UNIVERSITY OF CALIFORNIA
Santa Barbara

# Data-driven Graph Analysis

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Qingyun Liu

Committee in Charge:

Professor Ben Y. Zhao, Co-Chair

Professor Haitao Zheng, Co-Chair

Professor Xifeng Yan

September 2017

The Dissertation of
Qingyun Liu is approved:

Professor Xifeng Yan

Professor Haitao Zheng, Co-Chair

Professor Ben Y. Zhao, Co-Chair

March 2017

To my beloved

***grandparents***, ***parents***, ***husband*** and ***close friends***

Thanks for your love, warm hugs, support and company.

# Acknowledgements

My most sincere thanks go to my advisors, Prof. Ben Y. Zhao and Prof. Haitao Zheng. I greatly appreciate their efforts for mentoring me throughout the challenging PhD program. It is not often that one has the luck to find advisors that are always there for their students' needs, either in research or in life. I could always get timely and helpful feedbacks and advice from them. Sometimes they gave me direct advices on technical or editorial issues to help me avoid obvious roadblocks from their rich academic experience. And in most times, they tried to train me into an independent researcher, where they gave me guidance, encouragement and also freedom to think in my own way. I especially want to express my gratitude for being granted freedom in choosing research directions, where I could give trials in different topics and pursue what I am really fond of. Their continuous guidance, care and support were essential to the completion of my Ph.D. study. I have learnt innumerable lessons and gained great experience both academically and in life, which are valuable in both my career and life in general.

I also want to give my earnest gratitude to my Ph.D. committee member Prof. Xifeng Yan, for his guidance through my Ph.D. program. Prof. Xifeng Yan served on my MAE (Major Area Exam), Ph.D. proposal and Ph.D. defense committee, and gave me great advice and feedbacks.

I benefitted a lot from my four internships at different research labs and companies. I collaborated with Dr. Matti Hiltunen, Dr. Abhinav Srivastava and Dr. Yu Jin at AT&T Labs Research for a traffic monitoring project, worked on advertisement in online social networks with Dr. Smriti Bhagat and Dr. Anmol Sheth at Technicolor Research Center, tracked lateral movements across computers with Dr. Jay Stokes and Dr. Weidong Cui at Microsoft Research, and mentored by Dr. Li Yu and Dr. Junfeng Pan in the Feed Ads Ranking Team in Facebook. It was amazing experiences to work on such great projects with both interesting topics and big impact. I also felt quite lucky to work with such outstanding researchers and learnt from them. Especially I want to thank Dr. Jay Stokes, who is such a nice and kind person and always so pleasant to talk with. And he provided such enormous help for my project to get bigger impact in both application and publication.

My sincere thanks also go to all the members of our great SAND laboratory. I have enjoyed my Ph.D. life with you all, and greatly appreciate the collaboration with Shiliang Tang, Xiaohan Zhao, Xinyi Zhang and Megan McQueen. Thanks for our discussion and debates on research projects, which most of the ideas flourished. Thanks for working together towards the same goals, for being aside to share the joy when papers were accepted, and for encouraging each other when projects did not go well. I am also grateful for other former and current lab members, from whom I have learnt a lot: Ana Nika, Xia Zhou, Gang Wang, Yibo Zhu, Zengbin Zhang, Christo Wilson, Alessandra Sala, Bimal Wiswanath, Bolun Wang, Lin Zhou, Tianyi Wang, Yanzi Zhu, Zhijing Li, Yuanshun Yao, Zhujun Xiao, Yun Zhao, Jenna Cryan, Divya Sambasivan, Pritha D.N., Sujaya Maiyya, Kirti Bhandari and Abhay Chennagiri. Thanks for your encouragement, sharing my joy and sadness, and most of all, made my Ph.D. experience full of valuable memories.

I am also always thankful for all my collaborators. It is so lucky to have the chance to work with those outstanding researchers. I would like to thank Dr. Walter Willinger, Dr. Scott Counts, Dr. Apurv Jain and Xiao Wang, and also collaborators at my undergraduate study: Dr. Bin Cui, Dr. Junjie Yao and Zijun Xue.

Finally, and most importantly, I would like to express my special thanks to my intimate relationships. Thanks for my beloved grandparents Yunjin and Hao, you have taught me what is love. I will love and miss you forever. Thanks to my parents Lu and Jin , for your endless care, support and always being there for me. And Xin, my dear husband, I have always felt so lucky to have met you. We have walked down so many roads, seen so many views, and shared so many feelings. With your company, I do not feel alone in the journey of life. Also, I want to sincerely thank Isa for giving me the key to explore the world outside and inside, for letting me into a much broader, mysterious, amazingly beautiful world that I have ever seen. I also want to thank myself, for being brave enough to make mistakes and explore different possibilities in life, being acute enough to learn from the past and others' experiences, be persistent enough for the continuous efforts to learn about myself and life, and gradually become who I am. And finally, my closest friends, thanks for giving me the chance to share my life with you, and to let me accompany your journey. Thank you all, for letting me feel the richest and luckiest person in the world.

# Curriculum Vitæ

Qingyun Liu

## Education

2012-2017 Ph.D. in Computer Science
University of California, Santa Barbara, USA

2012-2016 Master of Science in Computer Science
University of California, Santa Barbara, USA

2008-2012 Bachelor of Science in Information Science and Technology
Peking University, Beijing, China

## Field of Study

Major Field Computer Science with Prof. Ben Y. Zhao and Prof. Haitao Zheng.

## Professional Experience

09/2012-03/2017 Research Assistant, University of California, Santa Barbara.

06/2016-09/2016 Software Intern, Facebook, Menlo Park, CA.

06/2015-08/2015 Research Intern, Microsoft Research, Redmond, WA.

06/2014-09/2014 Research Intern, Technicolor Research Center, Los Altos, CA.

06/2013-08/2013 Research Intern, AT&T Labs Research, Florham Park, NJ.

## Publications

ICWSM'17 Shiliang Tang, **Qingyun Liu**, Megan McQueen, Scott Counts, Apurv Jain, Haitao Zheng, and Ben Y.Zhao. "Echo Chambers in Investment Discussion Boards. " *International AAAI Conference on Web and Social Media*, May 2017.

IMC'16 **Qingyun Liu**, Shiliang Tang, Xinyi Zhang, Xiaohan Zhao, Ben Y.Zhao, and Haitao Zheng. "Network Growth and Link Prediction Through an Empirical Lens. " *In Proceedings of Internet Measurement Conference*, November 2016.

ToMPECS'16 **Qingyun Liu**, Xiaohan Zhao, Walter Willinger, Xiao Wang, Ben Y.Zhao, and Haitao Zheng. "Self-similarity in Social Network Dynamics. " *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, November 2016.

COSN'15 Xiaohan Zhao, **Qingyun Liu**, Haitao Zheng, and Ben Y.Zhao. "Towards Graph Watermarks." *In Proceedings of ACM Conference on Online Social Networks*, November 2015.

ICDE'12 Junjie Yao, Bin Cui, Zijun Xue, and **Qingyun Liu**. "Provenance-based Indexing Support in Micro-blog Platforms." *In Proceedings of IEEE International Conference on Data Engineering*, April 2012.

submitted **Qingyun Liu**, Shiliang Tang, Megan McQueen, Scott Counts, Apurv Jain, Haitao Zheng, and Ben Y.Zhao. "Herd Behavior and Inefficiency in Stock Markets. "

submitted **Qingyun Liu**, Jack W. Stokes, Rob Mead, Tim Burrell, Ian Hellen, John Lambert, Andrey Marochko and Weidong Cui. "LATTE: Tracking Malicious Lateral Movement Across a Computer Network. "

## Patent

09/2012-03/2017 Bin Cui, Junjie Yao, Hongzhi Yin, and **Qingyun Liu**. "A language model based expert recommendation method." China, Patent Number 102495860, October 2013.

# Abstract
# Data-driven Graph Analysis

Qingyun Liu

The ever-expanding demands for network utilities today have greatly changed people's lives. We are all around by various networks, from Internet, social networks, to World Wide Web. Graphs are fundamental abstraction for networks, which set the basis to systematically analyze and understand networks. Analyzing graphs is critical to provide insights on the fundamental process that drive the evolution of networks, and the essence for many real world applications, *e.g.*, social recommendations. Despite years of research in graph analysis, there has been little opportunity to study graphs from an empirical perspective. Prior studies are often limited by the size and granularity of public available datasets, which cannot accurately capture real graph complexity. In recent years, things are changing with the proliferation of online social networks (OSNs), which provides access to large traces of network dynamics.

In this dissertation, we take the opportunity by OSNs and seek to understanding graphs from a data-driven perspective. Following this goal, we address several graph problems which are of high impact, and also with great challenges in terms of scalability, high level of graph dynamics and privacy. We use empirical large-scale datasets to study new graph topics, step back to reassess how far we have come in analyzing fundamental graph problems, and also investigate how we can improve by leveraging real graph datasets.

Specifically, we first work on how to analyze and model graph dynamics, and we focus on the perspective of *self-similar* properties. Self-similarity is a fundamental property which defines hard limits on network modeling. Our work identifies the presence of self-similarity in the time dynamics of social graphs, and we incorporate the findings into a complete graph evolutionary model that can accurately capture key properties from both temporal and structural aspects. We validate our model against network dynamics in two real large-scale graphs, and show that it produced desired properties in both temporal patterns and graph structural features.

In our second work, we step back and reassess the space of the fundamental graph problem, *i.e.*, link prediction. Link prediction is the problem of predicting formation of new edges on a given graph, and applies to networking in numerous contexts. We perform an empirical study using different large traces of network growth to reassess the predictive power of current proposals, and augment them by leverage graph dynamic data.

Finally, we are concerned with graph privacy issues, *i.e.*, how to securely share large-scale datasets to trusted collaborators without data leakage. Current tools can provide limited protection, and we provide a new alternative in the form of *graph watermarks*. Graph watermarks are small graphs tailor-made for a given graph dataset, which are difficult to remove, and serve to associate to a particular user. In this work, we identify the goals and requirements of a graph watermark system, propose our basic and improved implementation, and evaluate the effectiveness and efficiency on various large graphs.

In summary, our research work demonstrates that data-driven graph analysis provides great insights, and is key to better understanding graph properties. We have addressed important graph problems in terms of scalability, high level of graph dynamics and graph privacy, and validated our proposals on large-scale real graphs.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The proliferation of networks have changed people's lives today more than ever before [73, 134]. There are ever-expanding demands for network utilities, from technology networks like the Internet and telephone networks, social networks like Twitter and Facebook where people share their everyday life, information networks like World Wide Web, to even biological networks like neural networks. As fundamental abstraction for networks, graphs model the connections in a network and set the basis to systematically analyze and understand networks. Graphs are applicable to various of networks, and many of today's datasets are captured in large dynamic graphs. Such datasets can include maps of autonomous systems in the Internet, social networks representing billions of friendships, connected records of patent citations, and interaction of proteins in personal health care.

Analyzing graphs is critical to provide insights on the fundamental process that drive the evolution of networks, and the basis for many real world applications. For example, studying dynamic social graphs is key to accurately predicting resource needs and system behavior in online social networks, and important for various applications including system design, resource allocation, anomaly detection, demand forecasting and advertising [29, 62, 131]. Another prominent example is that graph analysis has set the foundation for social recommendation systems [53, 89], which is widely used in social networks and applications, *e.g.*, ranging from Facebook, Twitter and LinkedIn, to photo sharing on Instagram and Pinterest, personal streaming on Periscope and Q&A sites like Quora.

Despite years of research in this space and hundreds of or maybe even thousands of publications [7, 10, 11, 17, 22, 31, 39, 53, 62, 89, 90, 91, 109, 111, 129, 130, 133, 150, 153] (only a small subset of which is cited in this dissertation), there has been little opportunity to study graphs from an empirical perspective, and prior studies are often hampered by limitations in the size and granularity of public available datasets, which cannot accurately capture real graph complexity. For example, most work in graph dynamics study graphs via *static* snapshots, which capture graph dynamics only at discrete points in time, and lack time information about events that occur between snapshots. Even for those that have analyzed graph traces in more fine granularity, they are often limited to moderate sized networks like co-authorship studies and patent citation graphs, which scale up to around 20K nodes and 200K edges [15, 130]. In contrast, graph algorithms/systemsdeveloped using and validated by these datasets are often targeting dynamic networks that are two or more orders of magnitude larger, with millions or billions of nodes and billions of edges [161].

Thankfully, things are changing with the arrival of network traces from online social networks (OSNs). We are taking advantage of this opportunity and availability to large traces of network dynamics to understand graphs from a data-driven perspective. We can use empirical datasets to study new graph topics, step back and reassess how far we have come in analyzing certain problems, and also investigate how we can improve by leveraging real graph datasets. At the same time, we are also faced with great challenges, since real large graph traces may have quite different properties. We list three primary challenges as follows:

1. *Scalability.* Many current graph tools have too high computational complexity to be scaled to empirical graphs [32, 119, 142], since they are developed or validated by small or moderate sized graphs. How can we adapt them or design new graph tools that are applicable to graphs with millions or billions of nodes is quite challenging.

2. *High level of dynamics.* Real graph traces often have high volume of dynamics, which prior studies have seldom dealt with. For example, Facebook has over 890 million login events and 4.5 billion likes every day [59], and Instagram has more than 80 million photos uploaded daily and 3.5 billion daily likes [118]. We need to understand and model such dynamics.

3. *Graph privacy.* There are huge concerns for graph privacy [14, 91, 107] since graph topology may represent very sensitive information from real world, *e.g.*, users' social relationship or even the strength of such relationship. It is a difficult and necessary task to fight against potential data leakage.

## 1.1 Dissertation Overview

Motivated by the new opportunities and challenges by large-scale empirical graph datasets, I have focused on graph research from data-driven perspective, and formalized the following statement of this dissertation:

**Using large traces of network dynamics, we can build graph tools to meet challenges from scalability, dynamics, and data privacy.**

Driven by this statement, we have tackled problems from three aspects. First, we work on how to analyze and model detailed graph dynamics, and we focus on the perspective of *self-similar* properties. Self-similarity is a fundamental property which defines hard limits on network modeling [75]. Our work identifies the presence of self-similarity in the time dynamics of social graphs, and we incorporate the findings into a complete graph evolutionary model that can accurately capture key properties from both temporal and structural aspects. Second, we step back and reassess the space of the fundamental graph problem, *i.e.*, link prediction. Link prediction is the problem of predicting formation of new edges on a given graph, and applies to networking in numerous contexts [68, 70, 53]. We perform an empirical study using large traces of network growth to reassess the predictive power of current proposals, and augment them by leverage dynamic data. Finally, we are concerned with graph privacy issues, *i.e.*, how to securely share large-scale datasets to trusted collaborators without data leakage. Current tools [91, 54, 128] can provide limited protection, and we provide a new alternative in the form of *graph watermarks*. Graph watermarks are small graphs tailor-made for a given graph dataset, which are difficult to remove, and serve to associate to a particular user. In this work, we identify the goals and requirements of a graph watermark system, propose our implementation, and evaluate the effectiveness and efficiency on various large graphs.

In the following, we briefly describe works included in this dissertation.

## 1.2 Analyzing and modeling graph dynamics

Analyzing and modeling social network dynamics are key to accurately predicting resource needs and system behavior in online social networks. The presence of statistical scaling properties, *i.e.*, self-similarity, is critical for determining how to model network dynamics. Self-similarity refers to the invariance behavior of a time series under rescalings, *i.e.*, the relative variance or volatility of traffic traces stays similar across different time scales [20, 75]. It is a fundamental statistical property which defines hard limits on traditional network models like Poisson [75].

In this work, we study the role that self-similarity scaling plays in a social network edge creation (*i.e.*, links created between users) process, through analysis of two detailed, time-stamped traces, a 199 million edge trace over two years in the Renren social network, and 876K interactions in a four year trace of Facebook.

We find that traditional R/S and Variance methods are unsuitable for measuring self-similarity in real traces of social graphs. Using the more advanced tool, *i.e.*, wavelet-based analysis, we find that the edge creation process in both networks is consistent with self-similarity scaling, once we account for periodic user activity that makes edge creation process non-stationary.

Using these findings, we build a complete model of social network dynamics that combines temporal and spatial components. Specifically, the temporal behavior of our model reflects self-similar scaling properties, and accounts for certain deterministic non-stationary features. The spatial side accounts for observed long-term graph properties, such as graph distance shrinkage and local declustering. We validate our model against network dynamics in Renren and Facebook datasets, and show that it succeeds in producing desired properties in both temporal patterns and graph structural features.

## 1.3 Reassessing current status of link prediction

In this work, we seek to understand current status of fundamental graph problem, *i.e.*, link prediction. Link prediction is the problem to of predicting formation of new edges on a given graph. It is a fundamental problem that applies to networking in numerous contexts, and has tons of applications [68, 70, 53]. However, there has been little opportunity to study various link prediction proposals from an empirical perspective due to the limitation of publicly available datasets. Until recently, validation of algorithms have been hampered by limitations in the size and realism of empirical datasets. More specifically, such public datasets are often limited to co-authorship studies and patent citation graphs, moderate sized networks that lack dynamic details [15, 130]. In contrast, algorithms developed using and validated by these datasets are targeting dynamic networks that are many orders of magnitude larger, and much more dynamic [161].

With our access to several large, detailed traces of dynamics in online social networks (Facebook, Renren, YouTube), we seek to revisit and reassess the value and accuracy of current prediction methods. Our goals are to understand the absolute and comparative accuracy of existing prediction algorithms, and to develop techniques to improve them using insights from analysis of network dynamics.

We implement and evaluate 18 link prediction algorithms, labeled as either "metric-based" (those that predict potential links using a single similarity or proximity metric) or "classification-based" (those that use machine learning classifiers with multiple metrics as input features). Despite poor performance in absolute terms, SVM classifiers consistently perform the best across all our traces. Its accuracy is occasionally matched by metric-based algorithms, but never consistently across datasets.

Finally, we dig sources for overall low accuracy of today's prediction algorithms, and use our observations of network dynamics to build "filters" that dramatically reduce the search space for link candidates. Augmenting current algorithms with our filters dramatically improves prediction accuracy across all traces and algorithms, even for algorithms that were already designed to capture network dynamics [37].

## 1.4 Secure graph sharing system

From network topologies to online social networks, many of today's most sensitive datasets are captured in large graphs. A significant challenge facing the data owners is how to share sensitive graphs with collaborators or authorized users, *e.g.*, ISP's network topology graphs with a third party networking equipment vendor. Current tools like building strong access control mechanisms, or modifying data to reduce the impact of potential leakages [91, 54, 128] can provide limited node or edge privacy, but significantly modify the graph reducing its utility.

In this work, we propose a new alternative in the form of graph watermarks. Graph watermarks are small graphs tailor-made for a given graph dataset, which are difficult to remove, and serve to associate to a particular user. When a data owner wants to share her graph with multiple users, she first generates a special subgraph, *i.e.*, a graph watermark, for each user. She then embeds each watermark into the original graph to get a watermarked graph separately, and distributes this graph to the corresponding user. If any watermarked graph is leaked, data owner uses the original graph to extract the embedded watermark, and identify who is responsible for the leakage. Then data owner can use this proof to seek potential damages against that user. Knowing the existence of such a way to track for responsibility, users would be more cautious with the shared data. Our graph watermarks serve both as a deterrent against data leakage and a method of recourse after a leak.

We provide robust schemes for embedding and extracting watermarks, and use analysis and experiments on large real graphs to show that they are unique and difficult to forge. We study the robustness of graph watermarks against both single and powerful colluding attacker models, then propose and evaluate mechanisms to dramatically improve resilience.

## 1.5 Contributions

In this dissertation, the key contribution is to carry out data-driven graph analysis from large-scale real datasets, where we have built graph tools to meet challenges from scalability, dynamics, and data privacy. We have provided novel algorithms, models, and systems to capture properties of large-scale real graphs, guided by our measurements. We have also revisited and evaluated the current status of fundamental graph problem to understand how far existing work has gone in this area. Besides, we have verified our solutions using rigorous theoretical analysis, experiments on a range of big real graphs and detailed simulations. In the following, we list our detailed contributions.

**Novel Algorithms, Models and System Designs.** We have designed novel algorithms/models/systems to address the key challenges from data-driven graph analysis on big real graphs. In Chapter 2, when we try to explore the existence of self-similarity property, we explored different detection algorithms and identified a reliable tool for dynamic social graphs. Based on our findings, we have proposed a novel dynamic graph model, which combines a temporal component that accounts for "when" and "how many" edges are created, and also a spatial component that captures "where" those edges are distributed. This is the first model to comprehensively capture the evolutionary dynamics of graphs from different aspects at fine granularity. Also, in Chapter 3, after we have evaluated the predictive power of existing link prediction algorithms and studied the causes for overall low prediction, we have provided "temporal filters" that can greatly improve the prediction, by pruning the search space for potential links. And in Chapter 4, to securely share graphs, we have overcome the limitations of current solutions by providing a new alternative in terms of graph watermarks. We are the first to identify the goals and requirements of a graph watermark system. We have also described an initial design that efficiently embeds watermarks into and extracts out of large graphs, and with low distortion of the graph data.

**Identifying Fundamental Challenges/Limitations.** We have revisited the fundamental graph problem, *i.e.*, link prediction, and explored the existence of fundamental statistical property in graph dynamics, *i.e.*, self-similarity. We have identified some fundamental/practical challenges faced by those problems. In Chapter 3, we are the first to evaluate existing link prediction solutions from an empirical angel, *i.e.*, on large-scale dynamic graph traces. We have found the overall low predictive power of all existing solutions, and dug sources of such low prediction accuracy from both structural and temporal aspects. We have found that one fundamental contributing factor is that current prediction algorithms take a purely static approach to network analysis, and do not take in account temporal patterns exhibited by an evolving network. In Chapter 2, we have proved when detecting self-similar property on real traces in social graph dynamics, how traditional tools produce inconclusive results because of the underlying deterministic trends by human behaviors. We have also validated the suitability of a more advanced and robust tool in such scenario.

**Theoretical and Experimental Validation.** We have validated our proposed solutions using theoretical analysis, experiments on various large-scale graph datasets, and also simulated attacks if needed. In Chapter 2, both our measurement for the existence of self-similarity property, and our proposed model for graph dynamics are based on two large-scale datasets: Renren, with 200M edges and 11M nodes at the end of the trace, and Facebook, with 877K edges and 47K nodes over a four-year period. By showing that our model produces dynamic traces that match key properties of the two original traces, we have proved that our model provides a practical method for generating realistic traces and has filled an existing void in this research community. In Chapter 3, we have applied our proposed "temporal filter" to existing solutions, and tested on different large-scale graphs, including Renren, Facebook and YouTube. We show that by leveraging temporal information on network dynamics using our filters, we can effectively improve link prediction accuracy. And we have also confirmed the generality of our filtering method. In Chapter 4, we have provided a strict theoretical proof of uniqueness of graph watermarks, showing that it is extremely difficult for attackers to forge watermarks. We have also tested the suitability of watermarking to 48 of today's real network graphs, which represent vastly different types of networks and a wide range of structural topologies with size ranging from 10K nodes and 39K edges, to 5M nodes and 48M edges. Results support our assertion that our watermarking mechanism is applicable to most of today's network graphs with low detection risk. Besides applicability, we have also validated our watermark system in terms of robustness agains attacks, distortion, uniqueness, and efficiency on several larger network graphs.

## 1.6 Thesis organization

The rest part of this dissertation is organized as follows:

In Chapter 2, we elaborate our efforts to analyze and model graph dynamics. Following the background and description of datasets, we carry out preliminary analysis with two traditional tools to detect the existence of self-similarity. From the initial analysis, we find the edge creation process in social graphs display a typical periodical pattern in user activity, and potential self-similarity properties at certain time scales. However, because of the existence of periodical patterns, both traditional tools fail to provide reliable detection results. We then apply a more rigorous method to systematically study potential self-similar properties, and validate its obvious existence over smaller time scales and fading away at larger time scales. Motivated by the self-similarity analysis, we next seek to build a complete model of graph dynamics. We propose a model with two components: a *temporal* component that produces a sequence of time-stamped events defining *when* and *how many* new edges are formed in a given time interval, and a *spatial* component defining *where* in the graph these new edge creations take place. Finally, we validate the proposed model on different datasets, where we calibrate the model using real data and use it to generate synthetic dynamics graphs, and then compare these synthetic graphs to the original data in terms of both temporal and spatial properties.

In Chapter 3, we present our work on revisiting the fundamental graph problem, *i.e.*, link prediction. We first introduce different categories of link prediction algorithms: *metric-based* and *classification-based* algorithms. We then present two key questions we want to ask in our efforts to study this problem, our proposed graph sequence based framework for evaluation, and 18 implemented algorithms. The empirical evaluation begins with metric-based prediction algorithms, where we seek to understand their prediction accuracy, and they key factors that lead to prediction errors. Later we evaluate classification-based algorithms in practical scenarios and compare their results to metric-based algorithms. Finally, we improve existing algorithms by integrating them with dynamic network analysis, where we propose *temporal filters* to drastically reduce the search space for link candidates. Our proposed solution can even augment algorithms that are already designed to capture network dynamics.

In Chapter 4, we introduce the design of graph watermark system. To set the context for the design, we first introduce the framework of graph watermark systems, define the attack models we target, and use them to guide our design goals. We then describe in details the basic design, which seeks to embed and extract watermarks on graphs to achieve watermark uniqueness while minimizing distortion on graph structure. Later we present detailed analysis on two fundamental properties: *watermark uniqueness*, where each watermark must be unique to the corresponding user, and *watermark detectability*, where the presence of a watermark should not be easily detectable by external users. Because in practice, attackers can seek to detect or destroy watermarked graphs, we propose advanced features to defend against corresponding attacks. We finally use real network graphs to evaluate the performance of the graph watermarking system in terms of key metrics: false positives, graph distortion, watermark robustness, and computational efficiency.

Finally in Chapter 5, we conclude the dissertation with future directions.

# Chapter 2

# Analyzing and Modeling Graph Dynamics

## 2.1 Introduction

[1] Studying graph dynamics, *i.e.* graph evolution including detailed timings of when nodes arrive and edges are created, is important for many network applications, from system design, resource allocation, anomaly detection, to demand forecasting. Prior studies of graph dynamics are typically based on randomized, generative graph models that produce sequences of events leading to an observed

[1] ©ACM, (2016). This is the authors version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The abbreviated version of content in this chapter was published in Journal ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) [93], http://doi.acm.org/10.1145/2994142.2342440.

network structure [65, 84, 10, 9, 109, 155]. Focusing primarily on producing a graph with some desired structural properties, they do not model or match the sequence of dynamic events that lead to that structure. With the proliferation of online social networks (OSNs) and thus access to real, large-scale traces of graph evolution, there are increasing number of literature in analyzing and modeling social graphs [133, 153, 62, 130, 39, 31, 22]. However, graph dynamics are still poorly understood. Current methods often study them via *static* snapshots, which capture graph dynamics only at discrete points in time, and lack time information about events that occur between snapshots.

Our work seeks to address this need by studying detailed dynamics in "time-stamped" traces of network growth. While most/all existing work analyze and model dynamics using *logical clocks*, we examine the relationship between network dynamics and real *physical clock* time. Specifically, the use of physical time allows us to tackle two significant challenges in the modeling of network dynamics. First, physical time allows us to determine if social network dynamics exhibit *self-similarity*, an invariance of behavior at different time scales. Self-similarity is a fundamental statistical property, that if discovered, defines hard limits on how such dynamics can be modeled using traditional means, *e.g.* Poisson. Its detection in contexts such as network traffic and web traffic has led to significant shifts in how such datasets were analyzed and modeled.

Second, analysis of a physical time trace allows us to build a model of OSN dynamics that captures not only structural properties of the network, but also the sequence of dynamic events leading to that structure. This type of dynamic graph model would address several practical OSN problems. First, the research community has repeatedly expressed a need for real dynamic graph traces. Using a real trace for calibration, our model can generate "realistic" dynamic graphs with a complete list of time-stamped network events. Next, our model can be used to perform "interpolation," *i.e.* construct complete dynamic graph traces that approximate the continuous network evolution between successive static snapshots of OSNs. Finally, our model can be used to detect abnormal events (attacks or changes in user behavior) in real networks, *i.e.* events that disrupt expected network dynamics.

In this chapter, we perform an empirical study of network dynamics by examining of network events over multiple years. For this Our work relies on two detailed, time-stamped traces of social networks, the *Renren* dataset [161] (complete, time-stamped trace of 199 million social links over 2 years) and the *Facebook wall post* dataset [72] (876K wall posts between users over 4 years in a Facebook regional network). To the best of our knowledge, these are the only datasets available today with sufficient granularity and event frequency to provide accurate analysis on network dynamics and self-similarity.

**Self-similarity based Network Analysis.** Self-similarity refers to the invariance behavior of a time series under rescalings, *i.e.* the relative variance or volatility of traffic traces stays similar across different time scales[2]. Successful detection of self-similar properties is a very meaningful result (for network modeling), because it defines fundamental limits on how such datasets can be modeled using traditional means. Due to its very different statistical properties, *e.g.* significantly higher burstiness, *self-similar* traffic cannot be easily captured or modeled by popular traffic models. In recent years, self-similarity has been found and led to changes in data modeling in variety of contexts, including local network traffic, wide-area network traffic, file system accesses, disk-level I/O, messaging and email communications and web traffic requests [75, 115, 36, 50, 122, 44, 126, 41]. In each case, the discovery of self-similar scaling properties led to a noteworthy shift in how such datasets were analyzed and modeled.

It is challenging to detect and quantify self-similar scaling properties in real network traces in a statistically rigorous manner. This is partially due to the likely presence of patterns (*e.g.* deterministic trends and diurnal or weekly cycles) that introduce non-stationarity. The edge creation process may be consistent with self-similar scaling over time scales ranging from seconds to hours. But patterns like

[2]Self-similarity can be used to describe scale invariance of certain properties of an object in space and/or time. In this paper, we adopt the *temporal* meaning, *i.e.* self-similarity along the time dimension.

diurnal or weekly user cycles likely dominate over larger time scales like days and weeks, and need to be accounted for before any self-similarity analysis. Intuitively, we seek to not only detect self-similar scaling properties in edge creation process, but also determine time scales where self-similarity is visible and can be quantified. Thus we use a range of techniques including R/S analysis, the variance fitting method, and a wavelet-based method. And our analysis focuses on edge creation, mainly because an exploratory analysis of the Renren data revealed no particular structure underlying the observed node creation events.

**A Model of Social Network Edge Dynamics.** We incorporate the findings from our self-similarity analysis into a complete evolutionary network model, including a *temporal* component that determines "when" new edge creations occur in time, and a *spatial* component that specifies "where" these new edges form. Together, this model produces a sequence of time-stamped events that uniquely define the formation and evolution of a social network or graph in time and space. By tuning a small number of parameters, our model can be calibrated to "fit" traces of measured graph dynamics exhibiting self-similar properties. We validate the model by comparing the model-generated edge creations to that of the real data (Renren and Facebook). Our results on both datasets show that the synthetic edge creation matches both the self-similar scaling behavior and the diurnal patterns exhibited by the real data. Furthermore, successive snapshots of the graph structure generated by our model match the corresponding snapshots of the original data on a variety of metrics, including average path length and average clustering coefficient.

We summarize our five key contributions in this chapter as follows:

First, we find that Renren's edge creation process is non-stationary over long-term periods. Even after removing the impact of node arrivals, traditional R/S and variance methods still produce inconclusive results on self-similar scaling. Thus, the two methods are unsuitable for measuring self-similarity in real traces in social networks. (see Section 2.3)

Second, by applying the more robust wavelet-based method for examining self-similarity, we find the edge creation process in Renren does exhibit properties consistent with self-similarity over time scales ranging from seconds to hours. We find the wavelet-based method to be highly robust detecting self-similarity in the presence of non-stationary trends. (see Section 2.4)

Third, We cross-validate our observations by repeating the above analyses on the Facebook wall post dataset, and confirm that it exhibits similar self-similarity properties observed from the Renren dataset. (see Section 2.5)

Forth, we propose a detailed model of social network dynamics that captures both the temporal properties of graph dynamics, in terms of self-similar scaling and deterministic non-stationary periodic patterns like diurnal or weekly cycles of user activity, and its spatial properties, including long-term graph distance shrinkage and reduction in local clustering. (see Section 2.6)

Finally, we validate our model by showing that it produces dynamic traces that match key properties of the original Renren and Facebook datasets, both temporally and spatially. Thus, by providing a practical method for generating realistic traces of time-stamped network events, our model fills an existing void in the research community. (see Section 2.7)

To the best of our knowledge, our work is the first to empirically study the presence of self-similarity in the time dynamics of online social networks (OSNs). Our findings highlight that instead of traditional Poisson models, the dynamics of real-world networks such as Renren social graph can often be adequately captured by a combination of a non-stationary component, *e.g.* long-term deterministic trends, and a stationary component, *e.g.* self-similar process. We believe that our model is the first to explicitly account for both temporal and spatial features in network dynamics and addresses an urgent need for accurate models of graph dynamics.

## 2.2 Background and Datasets

In this section, we introduce briefly the notion of self-similarity, and describe the Renren and Facebook dataset used in our study.

**Self-similarity.** For a time process, self-similarity refers to an invariance behavior, where certain statistical properties are similar under appropriately rescaled versions of the process [20, 75, 35]. Self-similarity has been observed in a variety of contexts in computing systems and networks, including web traffic [36], file system accesses [50], and traffic in both wide area networks [115] and local Ethernet networks [75]. For self-similar traffic, the aggregation of many bursty sources remains bursty across a wide range of time scales. This behavior is quite different from conventional Poisson processes that tend to produce traffic that smoothes out when observed over large time scales. While self-similarity can also be associated with geometry and describe the invariance in hierarchical structures [136], this work focuses on the temporal domain[3].

To formally define self-similarity, let $X = \{X_i : i = 1, 2, ...\}$ be a *covariance stationary* stochastic process whose autocorrelation function $r(k) \propto k^{-\beta}$ $(0 < \beta < 1)$ as $k \to \infty$. For each integer $m$ $(m > 0)$, we form a new process $X^{(m)}$ representing the averaged values of $X$ over disjoint blocks of size $m$. That is, the $j^{th}$ element of $X^{(m)}$ is:

$$X_j^{(m)} = \frac{1}{m}(X_{(j-1)m+1} + X_{(j-1)m+2} + ... + X_{jm}), j = 1, 2, ... \tag{2.1}$$

[3]Throughout the paper we refer to *temporal* self-similarity as self-similarity.

If $X$ is self-similar, then $r^{(m)}(k)$, the autocorrelation function of $X^{(m)}$, should satisfy [50, 75]:

$$r^{(m)}(k) = r(k), \quad \text{or} \quad r^{(m)}(k) \to r(k), \; m \to \infty. \tag{2.2}$$

An effective and commonly used metric to detect the existence or quantify the degree of self-similarity is the *Hurst parameter* $H$, measurable in multiple ways [5, 75]. Intuitively, $H$ helps to capture the "burstiness" of a covariance stationary process, where a higher $H$ corresponds to a process with more pronounced "bursts", *i.e.* large observations have a tendency to be followed by large observations, and small observations by small ones. Formally, $H = 1-\beta/2$, where $\beta$ is defined by the process $X$'s autocorrelation function $r(k) \propto k^{-\beta}$. A process exhibits self-similarity if $H$ falls in the range of $(0.5, 1)$.

Ideally, the finite-dimensional distributions of a self-similar process should stay invariant across all time scales. In reality, this property often exists at smaller time scales, but breaks down at large time scales due to non-stationary patterns and finite datasets [46, 50]. For example, diurnal user activity breaks stationarity and interferes with self-similarity at time scales larger than a few hours. Thus, analyzing for self-similarity requires determining the range of time scales over which it is visible [5, 46, 50].

**Datasets.** An Online Social Network (OSN) is an online platform to build social relations among people who share similar interests, opinions or have real-

| Graph | Trace Start Date | Trace End Date | Granularity | # of Nodes | # of Edges |
|---|---|---|---|---|---|
| Renren (Non-sampled) [161] | 11/21/05 | 12/31/07 | Seconds | 10,572,832 | 199,564,006 |
| Facebook (New Orleans) [72] | 09/14/04 | 01/22/09 | Seconds | 46,952 | 876,993 |

**Table 2.1:** Statistics of the two OSN datasets, with the start/end date of the traces, the granularity of time stamps in the traces, the total count of nodes that have been involved in edge creation, and the total count of edges that have been newly created in the traces.

life connections[4]. While many have diverse features, they typically share features that allow individuals to construct an own page or profile, and build connection with other users, *e.g.*, friending others. When modeling OSNs, an individual user is usually regarded as a "node", while the relationship between a pair of users as an "edge", or a "link".

Our analysis is based on OSNs: Facebook and Renren, where our work is the first to empirically study the presence of self-similarity in the time dynamics of OSNs. Facebook is the world's most popular online social network with over 1.5 billion users[5], while Renren is the Chinese version of Facebook, the largest and oldest OSN in China with more than 220 million users [62]. For both sites, a registered user can create her profile, add other users as "friends", and post messages on others' wall (called "wall posts"), an area on each user's own profile where others (usually friends) can make comments.

[4]https://en.wikipedia.org/wiki/Social_networking_service
[5]https://en.wikipedia.org/wiki/Facebook.

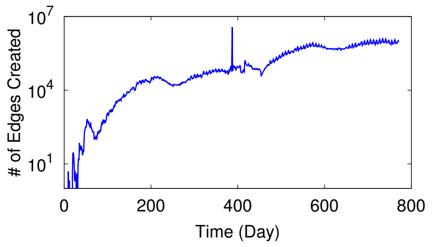


**Figure 2.1:** Daily edge growth in Renren.

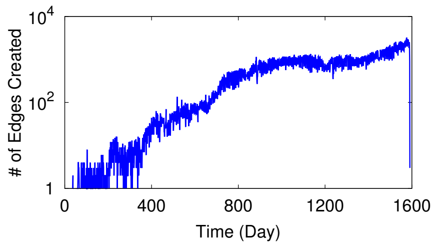


**Figure 2.2:** Daily edge growth in Facebook.

We show the summarized statistics of the two datasets in Table 3.2. The first and primary is an anonymized dataset from Renren [161], with a detailed time-stamped (down to the second) trace of the creation of all nodes (10,572,832) and all edges (199,564,006) over a 25-month period from Nov. 21, 2005 (the launch of Renren) to Dec. 31, 2007. Here an edge is created when two users become friends. To the best of our knowledge, this is one of the largest time-stamped datasets on social network evolution studied to date.

Figure 2.1 plots the daily edge growth of the Renren social network, where data points represent the number of *new* edges created on each day. This plot shows that the dataset covers both the initial explosive growth (from day 1 to around day 200) and the stabilized evolution of the Renren network [161]. Note the unusually large spike on day 386 (Dec. 12, 2006). This is the result of a merging event - Renren merged with *5Q*, its largest Chinese competitor at that time. The network doubled in size in a single day, growing from 624K users and 8.2M links to 1.3M users with 11.2M links. Given it is a one-time event, we exclude it from our analysis, and focus in our study on continuous data segments before or after the merge.

The second dataset is the Facebook wall post dataset[6] [72]. It contains wall posts produced by users from the Facebook New Orleans regional network, *i.e.* 46,952 users and 876,993 posts created over a four-year period from Sep. 14th, 2004 till Jan. 22nd, 2009. Each post is also time-stamped to the granularity of a second. Like [153], we consider each wall post as an edge representing an interaction between two users. Figure 3.6 plots the daily edge growth of the Facebook social network. Like Renren, this dataset also covers periods where edge creation events increase significantly at the beginning, and then stabilize (around day 750). Compared to Renren, this dataset is much more sparse.

## 2.3 Preliminary Analysis

Our goal is to determine if Renren and Facebook's network evolution display any property consistent with self-similarity, and if so, over what range of time scales. For clarity we first describe our analysis for Renren, which we repeat on the Facebook dataset in Section 2.5. Our analysis focuses on the edge creation process, since initial analysis showed no particular structure underlying the observed node creation events. The key challenge we face is how to identify and isolate the

[6]http://konect.uni-koblenz.de/networks/facebook-wosn-wall

impact of non-stationary patterns in the edge creation data. As a first step, we limit the impact of new node arrivals on edge creation, by focusing our analysis on edges created between members of a fixed user population. We remove this restriction and extend our analysis for all edge creation events in Section 2.4.3.

Next, we start by briefly describing how we sample the original dataset by removing certain node arrival and other obvious non-stationary events. We then discuss the methods for detecting self-similarity, our initial analytical findings and key insights.

### 2.3.1 Experiment Setup

**Data Sampling.** We begin our analysis with a conservatively sampled subset of our data to remove obvious non-stationary factors that may impede any direct analysis of self-similar scaling property. Specifically, we limit our sample to include only existing users as of December 1, 2007, and study all edge creation events between them during December 2007, *i.e.* days 741 to 771. This sampling eliminates three factors. First, by studying only edges created between members of a fixed user population, we minimize the impact of new node arrivals. Second, this month avoids the abnormal expansion of new edges around day 386 as a result of the one-time merge event of two social networks (Renren and 5Q). Finally, this time period is sufficiently late in the history of Renren that it avoids the initial exponential network growth experienced by most social networks [161]. This data sample represents a stable growth period in Renren, which contains $18,714,712$ edges created between $6,219,531$ existing users. In the following we refer to this sampled dataset as “sampled dataset of Renren” to differ from the entire dataset without sampling as “full Renren”.

**Estimating $H$.** The two most popular (and simple) methods to estimate $H$ are *variance analysis* and *R/S analysis* [46, 50, 75]. Our initial analysis efforts consist of applying these two methods in addition to directly visualizing the raw data.

Variance fitting method [75, 115] analyzes the decaying behavior of variances of the aggregated processes $X^{(m)}$ introduced earlier, with $m$ the block size. From Equation 2.2 in Section 2.2, a self-similar process $X$ satisfies:

$$\log(Var(X^{(m)})) \propto -\beta \log(m), m \to \infty \tag{2.3}$$

where $\beta = 2(1 - H)$. Thus by linearly fitting the plot of $\log(Var(X^{(m)}))$ versus $\log(m)$, this method can estimate $\beta$ and then $H = 1 - \beta/2$.

R/S analysis computes $H$ by measuring how apparent the variability of a time series changes with the length of the time-period being considered, which can be formally captured by the R/S statistic [50, 75]. To compute $H$, it divides the process $X$ into blocks of size $n$, and computes the corresponding R/S statistic

$R(n)/S(n)$. Because there is

$$E[R(n)/S(n)] \propto n^H, , n \to \infty \tag{2.4}$$

for self-similar processes [50], $H$ is estimated using the slope of $\log(E[R(n)/S(n)])$ versus $\log(n)$.

### 2.3.2 Measurement Results

We now present the results using three heuristics: *visualization of raw data*, *variance analysis* and *R/S analysis*.

**A Long-term Diurnal Pattern.** Figure 2.3 visualizes the edge creation process by plotting the number of new edges created in each second over the one month (Day 741-771). We can clearly observe a diurnal pattern in the edge creation process. This non-stationary behavior precludes any direct analysis of self-similarity. We confirm this from the results of the variance and R/S analysis. Figure 2.4 plots the values of $\log(Var(X^{(m)}))$ against $\log(m)$. The curve maintains a linear shape until $m$ reaches $10^4$ seconds ($\approx$ 3 hours), and then its slope changes significantly. Similarly, Figure 2.5 plots in log-log scale individual R/S statistics as a function of the block size $n$ (in seconds). The red straight line shows the best linear fit and its slope results in an H-estimate of $H = 1.19$, clearly outside the allowed range of $(0.5 < H < 1)$.

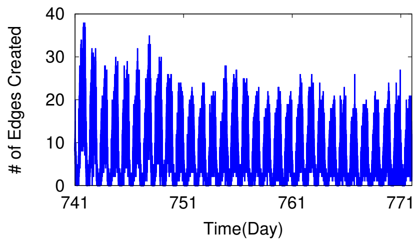
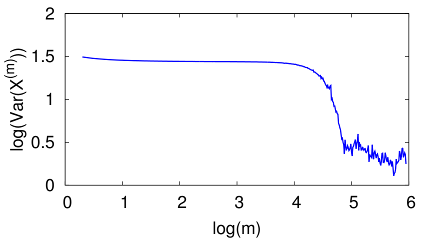


**Figure 2.3:** Edge growth in sampled dataset of Renren, in terms of the number of new edges created per second. It shows a clear diurnal pattern.



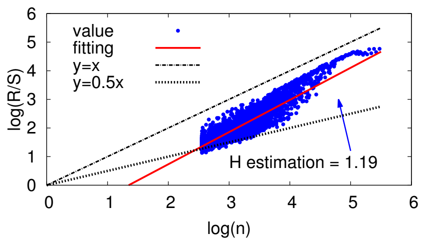


**Figure 2.4:** Variance analysis of sampled dataset of Renren: the slope changes greatly when $m$>$10^4$ seconds ($\approx$ 3 hours), preventing direct analysis on self-similarity.

**Figure 2.5:** R/S analysis of sampled dataset of Renren: $H$ estimation is beyond range of self-similarity, and data shape changes significantly for $n$>$10^4$ seconds ($\approx$ 3 hours).

The appearance of such a pronounced diurnal pattern has a direct impact on subsequent efforts to model our dataset. It suggests that models should include a component that accounts for this expected user-generated periodic behavior.

**Self-similar Fluctuations.** An interesting observation from Figure 2.3 is that the fluctuations on top of the diurnal component display a bursty behavior. Similarly, Figure 2.4 and 2.5 show that the curve only starts to lose its line shape when $m$ or $n$ exceeds $10^4$ seconds ($\approx$ 3 hours). Figure 2.6 shows the edge creation events of a randomly chosen 3-hour segment (6pm-9pm, December 16, 2007). It is

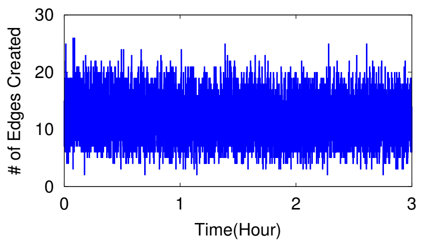


**Figure 2.6:** An example of edge growth of a randomly chosen 3-hour segment in the sampled dataset of Renren. It is highly bursty, appears stationary and suggests further exploration for self-similar scaling behavior.

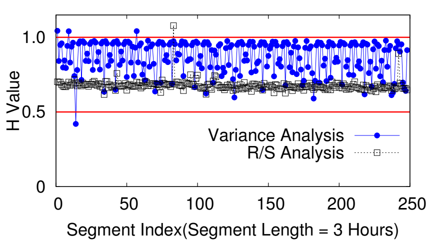


**Figure 2.7:** Estimates of $H$ by both Variance and R/S analysis on disjoint 3-hour segments in the sampled dataset of Renren, where 98%+ of $H$ estimates fall within (0.5,1).

highly bursty, appears stationary and could therefore exhibit self-similar scaling behavior. Together, these observations suggest that over time scales not significantly impacted by the observed diurnal patterns (*i.e.* a few hours and below), the edge creation process may be consistent with self-similar scaling behavior.

We confirm this intuition by performing variance and R/S analysis on each 3-hour log segment and computing its $H$ value. Figure 2.7 plots the results over the entire month as 248 disjoint 3-hour segments. $H$ estimates based on the variance analysis method vary across segments, with a mean of 0.89 and variance of 0.01, while R/S analysis remains stable, with mean of 0.68 and variance 0.001. For both methods, overwhelming majority of segments (98.4% for variance, 99.5% for R/S) estimate $H$ within $(0.5 < H < 1)$. These results suggest that the Renren edge creation process exhibits self-similarity over time scales ranging from seconds to hours.

### 2.3.3 The Reliability of our $H$ Estimates

In our analysis, we encountered potential issues regarding the reliability of $H$-estimates using the variance and R/S analysis methods. For some segments, the methods produced poorly-fitting linear regression lines, which in turn resulted in highly questionable estimates of $H$. Figure 2.8 shows an example of such a "problematic" segment (6-9am, December 6, 2007), where the line fitting is poor via variance analysis. We also plot as an inset in the figure the raw edge growth during the time period, which shows a clearly non-stationary event. We further study these events in Section 2.4.2.

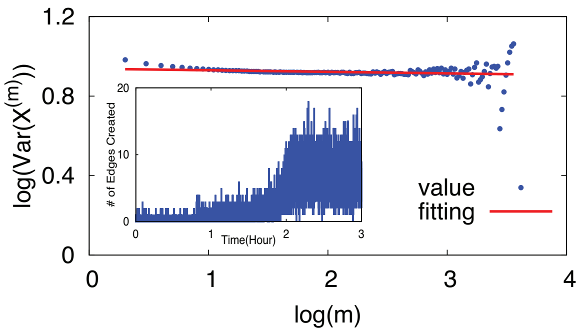


**Figure 2.8:** An example of poor line fitting in variance analysis, which has poor $R^2$=0.0458. This is also confirmed by the inset which displays the raw edge growth during the corresponding time period, and shows a clearly non-stationary event.

To quantify the impact of such poor data fitting on the obtained $H$ estimates, we compute the coefficient of determination $R^2$ for each segment. $R^2$ measures how well the observed data points are represented by a straight line. Like [50], we use the criterion of $R^2 > 0.9$ to indicate that the fitting is sufficiently good to provide a reliable $H$ estimate. 38.3% of all segments have unreliable $H$ estimates by R/S analysis vs. 71.0% by variance analysis! Prior studies have reported similar reliability issues [64, 141].

### 2.3.4 Summary of Observations

Our initial analysis led to three main findings. *First*, the Renren edge creation displays a typical diurnal pattern in user activity that makes the process inherently non-stationary, preventing a direct analysis of self-similarity. This suggests that any accurate model of Renren's edge creation process must include a component that explicitly account for this periodic behavior. *Second*, local fluctuations on top of the periodic component display behavior that indicates potential self-similarity. *Finally*, we find that two commonly-used methods, *i.e.* variance and R/S analysis methods, cannot provide reliable $H$-estimates for real data that displays non-stationary patterns.

Thus, our next step is to avoid most of the encountered problems by applying a more rigorous method for systematically analyzing data with potential scaling behavior that has strong robustness properties with respect to underlying non-stationary patterns, and results in $H$-estimates with known statistical properties (*e.g.*, confidence intervals).

## 2.4 Wavelet-based Analysis

Following our initial analysis, in this section we apply a more rigorous wavelet-based method to systematically study potential self-similar scaling behavior exhibited by our dataset. This method has strong robustness against underlying non-stationary patterns and can provide $H$-estimates with confidence intervals. To this end, we first briefly introduce the wavelet method and then present our findings.

### 2.4.1 The Wavelet Method

Estimation errors of the variance and R/S analysis methods can be attributed to their "eyeballing" approach when attempting to identify self-similarity in highly variable data. In contrast, the wavelet-based method offers a principled and rigorous analysis of a given dataset's scaling property by isolating characteristics of data via a combined scale-time presentation. In turn, it provides a more reliable self-similarity analysis [5].

In short, wavelet-based analysis represents a process $X$ by a sequence of subspaces $\{W_j\}_{j \in Z}$ where $W_j$ is at a finer scale than $W_{j-1}$ ($W_j \subset W_{j-1}$). This way, it can reveal detailed properties of $X$ at different time scales. If $X$ is self-similar, its projection on the $W_j$ subspace $\Gamma_j$, satisfies:

$$E[\Gamma_j] \sim |2^{-j}v_0|^{1-2H} \tag{2.5}$$

Here $2^{-j}v_0$ represents the reference frequency of the $j^{th}$ subspace $W_j$ while $v_0$ is the reference frequency of the root subspace $W_0$. One can estimate the Hurst parameter $H$ by plotting $E[\Gamma_j]$ vs. $j$ on a log-log scale and applying linear regression.

We estimate $H$ using the wavelet software developed [5] for self-similarity analysis. By carefully choosing the number of vanishing moments $N$ that controls $v_0$, the tool can systematically detect and remove the impact of various types of deterministic trends in the dataset. Furthermore, it also relies on known theoretical properties of the resulting $H$-estimate to provide confidence interval (CI) for $H$. In the analysis of our dataset, we choose the value of $N$ that produces both a good fit and the smallest confidence interval.

### 2.4.2 Measurement Results

We seek to confirm and substantiate our preliminary results that show properties consistent with self-similar scaling in our Renren dataset. We divide our sampled dataset into disjointed segments of lengths between 3 and 12 hours and apply the wavelet-based analysis to each segment. In our analysis, we refer to a segment as "abnormal" if its $H$ estimate (including its 95% confidence interval)

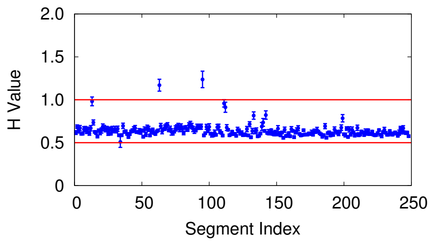


**Figure 2.9:** Wavelet analysis on data segments with **segment length = 3 hours** (sampled dataset of Renren).

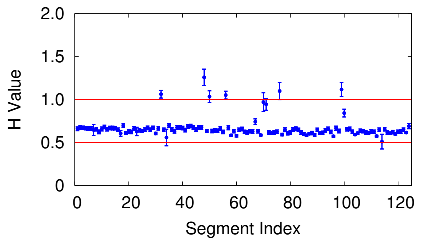


**Figure 2.10:** Wavelet analysis on data segments with **segment length = 6 hours** (sampled dataset of Renren).

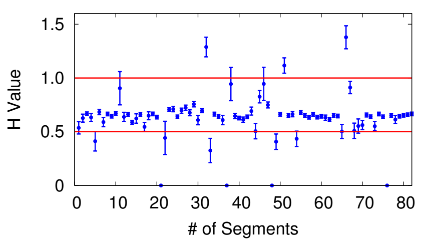


**Figure 2.11:** Wavelet analysis on data segments with **segment length = 9 hours** (sampled dataset of Renren).

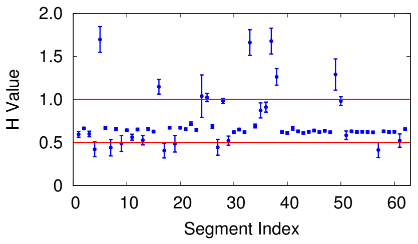


**Figure 2.12:** Wavelet analysis on data segments with **segment length = 12 hours** (sampled dataset of Renren).

does not completely fall within the self-similar range $(0.5, 1)$. Our analysis leads to two key findings.

**Self-similarity Over Time Scales from Seconds to Hours.** Our analysis confirms that over time scales ranging from seconds to a few hours, Renren's edge creation process exhibit properties consistent with self-similar scaling. As an example, Figure 2.9 shows the $H$-estimates with their 95% confidence interval for all 248 3-hour long segments. Only 5 segments are abnormal, while the rest

| Start Time Shift | Normal Segments | | Abnormal Segment |
|---|---|---|---|
| | H mean | H variance | Portion |
| 0 hour | 0.631 | 0.002 | 2.02% |
| 1 hour | 0.633 | 0.002 | 2.43% |
| 2 hours | 0.629 | 0.002 | 2.02% |

**Table 2.2:** Statistics of wavelet analysis on 3-hour segments with start time shifts. (sampled dataset of Renren)

(98%) consistently produces $H$-estimates within $(0.5, 1)$, and tightly clustered around $H = 0.63$.

To examine the robustness of our results, we check different segment compositions by shifting the start time of each segment by 0, 1, and 2 hours separately. From the summarized results in Table 2.2, we notice the stability in the mean (0.63) and variance (0.002) of $H$-estimates for normal segments, and also the portion of segments deemed abnormal ($2.02\% \sim 2.43\%$) These results provide further evidence that Renren's edge creation process behaves properties consistent with self-similar scaling over time scales from seconds to a few hours.

**Scaling Behavior over Larger Time Scales.** We observe that the number of abnormal segments increases as the segment size increases. Figure 2.10 to Figure 2.12 plot the $H$-estimates across all segments for segment lengths of

6, 9 and 12 hours. The ratio of abnormal segments increases to 8.1% for 6-hour segments, and up to 32.3% for 12-hour ones. It confirms our earlier conclusion that the properties consistent with self-similar scaling weaken in Renren's edge creation process, when viewed over larger time scales. This phenomenon is perhaps due to the presence of harder-to-account-for non-stationary patterns, such as heteroscedasticity (*i.e.* edge creation in Renren is more variable during peak hours than during low hours).

**Patterns of Abnormal Segments.** We also wish to understand patterns and potential causes for the observed abnormal segments. We find that these abnormal segments are randomly distributed across days, and within a day, around 60% of them appear during 6pm-9pm when Renren users are most active (the number of edges created account for 23% of the whole day).

We also find that abnormal segments are caused by sudden changes in the edge growth process. Based on the edge growth patterns, we are able to classify abnormal segments into three types, shown in Figure 2.13 to Figure 2.15. These include *level shift*, where the volume of edge growth suddenly increases (or decreases), *momentary drop* where the growth experiences a short period of extremely low activity, and *ramp up/down* where the edge activity quickly increases or decreases in the segment.

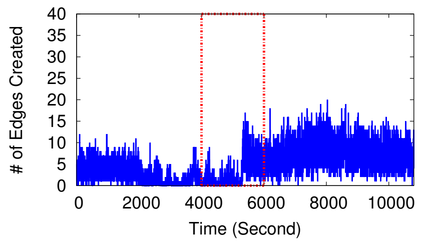


**Figure 2.13:** Examples of abnormal segments in terms of **level shift**, where the red dot boxes show the unusual edge creation events (sampled dataset of Renren).

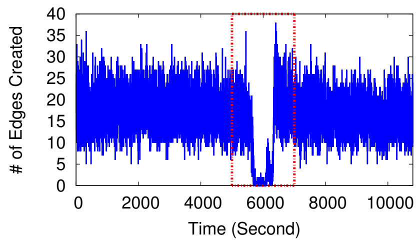


**Figure 2.14:** Examples of abnormal segments in terms of **momentary outage**, where the red dot boxes show the unusual edge creation events (sampled dataset of Renren).

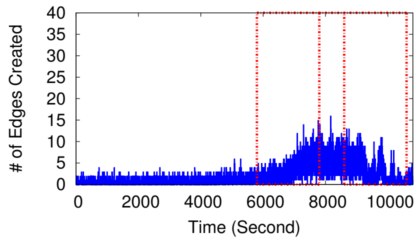


**Figure 2.15:** Examples of abnormal segments in terms of **ramp up/down**, where the red dot boxes show the unusual edge creation events (sampled dataset of Renren).

Our collaborators at Renren have confirmed that while per hour identification is difficult, it is possible that at least some of these abnormal events match changes to the site and its features. Intuitively, level shifts and momentary drops might be caused by new features or localized failures, and ramp up/down events might correspond to ad promotions to increase user membership. We are working with Renren to further confirm this.

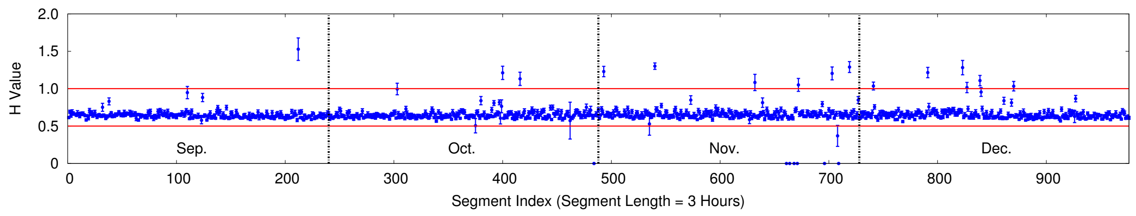


**Figure 2.16:** The $H$-estimates of all the disjoint 3-hour segments between September - December 2007 of the Renren dataset, after performing wavelet analysis on the entire dataset without sampling (full Renren). The results align with those with sampling (labeled as "sampled dataset of Renren" in caption).

### 2.4.3 Analysis Without Sampling

Finally, we expand our analysis to consider the full, unsampled dataset. This is to examine whether the observed property consistent with self-similar scaling on the sampled data still present after including new nodes with rapid (and non-stationary) edge growth.

We first consider the complete dataset from the month of December 2007. Interestingly, 97% of the 3-hour segments produce H estimates within the self-similar range, with mean $H = 0.65$. We show detailed $H$ estimates in Figure 2.16, which are highly consistent with our prior analysis on the sampled dataset (Figure 2.9). The only minor difference is two additional abnormal segments, possibly caused by non-stationary edge growth of the new nodes.

Next, we examine all edge events in the year of 2007. Again, we get consistent results: H-estimates of 97% of the 3-hour segments fall into the self-similar range, with mean $H = 0.64$. Figure 2.16 shows $H$-estimates for September-December

2007 (due to the space limit), which are representative of all other months. Together, these results suggest with high possibility, that the same self-similar property is present consistently throughout time. These results also confirm the high reliability of the wavelet method in self-similarity detection.

### 2.4.4 Summary

We apply the more reliable and accurate wavelet method to detect self-similarity at different time scales in Renren's edge creation process. The outcomes confirm prior observations from R/S and variance results, with high confidence, that the property consistent with self-similar scaling lasts to several hours. This property also holds for our full, unsampled dataset (after the network merge).

## 2.5 Validation via Facebook Dataset

One reasonable question is whether our results are strongly biased by our choice of dataset, *i.e.* property consistent with self-similarity is only present in Renren network. Here, we validate our findings using the Facebook wall post dataset [72]. Recall that for self-similar property to be detectable, a dataset must cover temporal events in fine granularity and have sufficient event frequency to provide

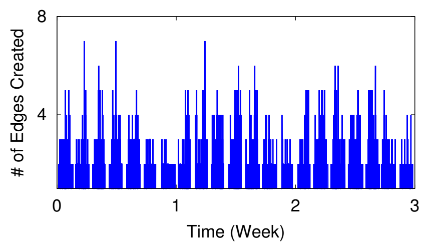


**Figure 2.17:** An example of edge growth of a randomly chosen 3-week data (Facebook).

meaningful statistics. To our knowledge, the Facebook wall post dataset [72] is the *only* dataset aside from our Renren dataset that meets these requirements.

As Figure 3.6 shows, like Renren, the number of edge creations in Facebook dataset increases significantly at the beginning and stabilizes around day 750. To eliminate the impact of this obvious non-stationary increasing trend, we focus on the edge creation process after day 750 (for a total duration of 841 days). Compared to Renren, this dataset is much more sparse, and per-second level analysis does not show any meaningful statistics (only 1.15% of non-zero data points). Thus we enlarge the time unit for analysis to 120 seconds, where the resulting ratio of non-zero data points (61.18%) is comparable to that of Renren (61.46%).

Following analysis in Section 2.3 and Section 2.4, we start by visualizing the raw data in Facebook dataset, and then apply the variance, R/S and wavelet analysis methods to see whether any property consistent with self-similar scaling exists across the whole time range. Figure 2.17 shows a random sample of three

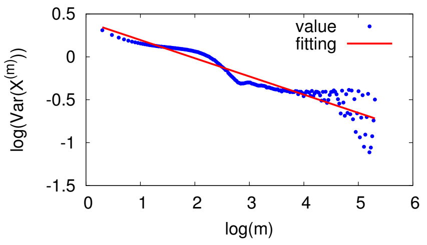


**Figure 2.18:** Variance analysis on the entire data: doubtable fitting with curves around $10^3$ units (Facebook).

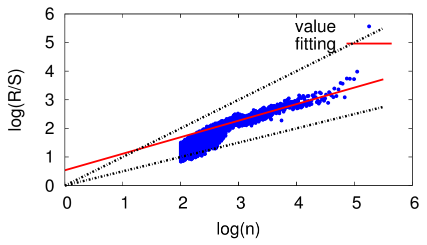


**Figure 2.19:** R/S analysis on the entire data: doubtable fitting since the shape changes greatly after $10^3$ units (Facebook).

successive weeks (from day 762 - day 782) in the edge creation process, which displays a clear weekly pattern. Similarly, this obvious non-stationary behavior precludes any direct analysis on self-similarity. We also confirm this using the R/S and variance analysis methods, where three successive weeks random the estimated $H$ values are 0.5779 and 0.8940 respectively. Although these H-estimates are within the self-similar range $(0.5, 1)$, Figure 2.18 and 2.19 show that the two methods have poor data fitting, resulting in unreliable estimations on $H$. On the other hand, the wavelet analysis produces a $H$ value of 1.11, indicating that there is no property consistent with self-similarity across the entire time range. Together, all these results suggest over the time scale up to years, we cannot reliably detect self-similarity properties in Facebook dataset.

Next, we explore whether the dataset displays self-similar scaling properties on shorter time scales. We split the entire dataset into fixed size segments of length varying between 1 to 7 days, and apply the wavelet analysis to each segment.

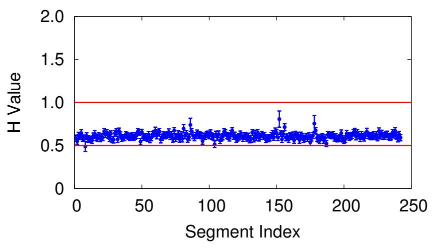


**Figure 2.20:** Wavelet analysis on data segments with **segment length** = **3.5 days** (Facebook).

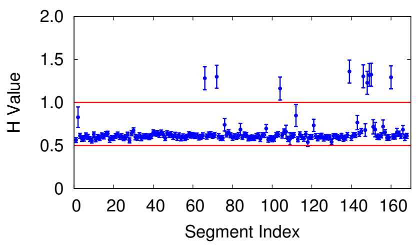


**Figure 2.21:** Wavelet analysis on data segments with **segment length** = **5 days** (Facebook).

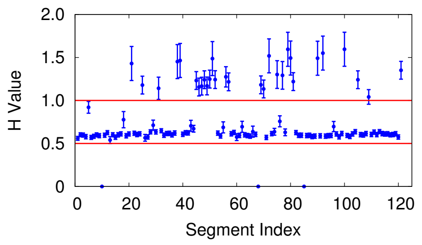


**Figure 2.22:** Wavelet analysis on data segments with **segment length** = **7 days** (Facebook).

Figure 2.20 to Figure 2.22 plot the H-estimates with their 95% CI at segment length of 3.5 days, 5 days and 7 days respectively.

We obtain two key observations. *First*, we observe strong self-similarity properties over the time scale between minutes and days. Figure 2.20 shows that 98.35% of 3.5-day segments have H values with 95% CI falling into range $(0.5, 1)$, centered around $H = 0.61$. And by shifting start times of segments, the consistent results in Table 2.3 further confirm this observation.

*Second*, the portion of abnormal segments (whose $H$ estimates are out of $(0.5, 1)$) increases with segment length, *i.e.* 1.65% for 3.5 days, 5.95% for 5 days

| Start Time Shift | Normal Segments | | Abnormal Segment |
|---|---|---|---|
| | H mean | H variance | Portion |
| 0 day | 0.612 | 0.001 | 1.65% |
| 1 day | 0.611 | 0.001 | 2.07% |
| 2 days | 0.611 | 0.001 | 2.07% |
| 3 days | 0.613 | 0.001 | 3.32% |

**Table 2.3:** Statistics of wavelet analysis on 3.5-day segments with start time shifts (Facebook).

and 26.45% for 7 days. A detailed analysis on the dataset shows that this is mostly caused by a weekly pattern (as shown in Figure 2.17) of user activities which dominates at larger time scales.

In summary, our results on the Facebook dataset align very well with our observations from the Renren dataset. Due to the existence of non-stationary patterns introduced by human behaviors, *e.g.* diurnal or weekly user activities, properties consistent with self-similar scaling exist, but only hold over certain time ranges, and gradually weaken at larger time scales.

## 2.6 A Model of Network Dynamics

Motivated by our self-similarity analysis of Renren and Facebook's edge creation process, we next seek to build a complete model of social network dynamics. Our proposed model includes two components: a *temporal* component that produces a sequence of time-stamped events defining *when* and *how many* new edges are formed in a given time interval, and a *spatial* component defining *where* in the graph these new edge creations take place (*i.e.* which nodes are involved). Ideally, the model should produce synthetic dynamic graphs whose edge creation will display deterministic non-stationary periodic patterns (*e.g.* diurnal or weekly user activities) and properties consistent with self-similarity, and whose graph structural changes match those observed from the original data and account for key spatial properties, *e.g.* graph densification, path shrinkage and local declustering [161]. Next, we explain the model in detail and provide validation in Section 2.7.

### 2.6.1 The Temporal Component

Our analysis in Section 2.3, Section 2.4 and Section 2.5 show that for both Renren and Facebook datasets, the edge creation process displays a combination of deterministic non-stationary periodic patterns, *i.e.* diurnal or weekly user activities, and property consistent with self-similarity. These observations motivate designing the temporal component of our model as a combination of two submodules: a *non-stationary* module that captures the predictable cycles in user activities, *e.g.* daily or weekly cycles, and a *self-similar* module that parsimoniously accounts for the inherent burstiness in user edge creations over certain time scales, *e.g.* from seconds to a few hours.

**The Self-Similar Module.** Prior work has demonstrated two effective methods for producing self-similar traffic. The first method aggregates many ON/OFF processes and under certain conditions, the superposition process displays a self-similar scaling [152, 50]. In particular, this construction requires statistical knowledge of the ON and OFF periods and assumes that either of those periods are modeled by a heavy-tailed distributions. The second method is based on the $M|G|\infty$ queuing model [35, 151]. Here, each source arrives according to a Poisson process, and the distribution of its active time is assumed to be heavy-tailed, *e.g.* the Pareto distribution. During its active time, each source is assumed to operate at a constant rate. Then the resulting count process $\{N_t, t = 0, 1, 2, ...\}$, where $N_t$ is the number of active sources at time $t$, is self-similar. In other words, by multiplexing sources with Poisson arrivals and heavy-tailed active times, one can produce a self-similar process.

Examining our two datasets in more detail shows that the $M|G|\infty$-based method provides an intuitive way and a good fit for modeling edge creation. For

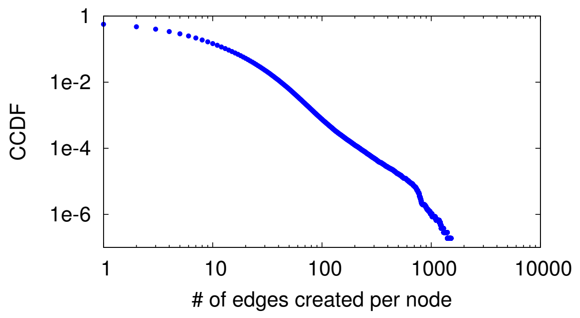


**Figure 2.23:** CCDF of # of edges created per user in Dec. 2007 in Renren dataset.

one, we observe that over time, the number of edges created per user follows a heavy-tailed distribution. For example, Figure 2.23 plots the distribution of the number of edges created per Renren user during December 2007, which can be approximated as a heavy-tailed pattern. Moreover, assuming each user creates edges at a constant rate, the active time of a user is directly proportional to the number of edges that user created. This in turn implies that each user's active time also follows a heavy-tailed distribution, consistent with the $M|G|\infty$-based construction of self-similar processes.

Based on this intuition, we build the self-similar module based on a standard $M|G|\infty$ process [35]. Users arrive according to a Poisson process with rate $\lambda$. Upon arrival, each user independently starts its active time duration $T_i$ (seconds) chosen from a Pareto distribution:

$$P(X > x) = (\frac{x_m}{x})^{\alpha}, \quad x \geq x_m, \quad 1 < \alpha < 2 \tag{2.6}$$

Assuming that each user creates edges at a constant rate $\gamma$/s, we can calculate the total expected number of edges created by user $i$ by $T_i \cdot \gamma$. Since an edge creation involves two users, we derive the number of edges $S_t$ created at time $t$ from the number of active users $N_t$:

$$S_t = \gamma \cdot N_t/2 \tag{2.7}$$

The time series $\{S_t, t = 0s, 1s, 2s, ...\}$ defines the self-similar module of the temporal component of our model.

**The Non-stationary Module.** We extract the deterministic non-stationary periodic component by subtracting the self-similar component from the original edge creation process. Suppose the number of original edge creation is $O_t$ at time $t$. Then the subtraction produces a process:

$$U_t = O_t - S_t, t = 0s, 1s, 2s, ... \tag{2.8}$$

Next we apply a sliding window over $U_t$ to obtain a smooth deterministic process and then fit it with a periodic function, *i.e.* Sine, to produce $D_t$, the non-stationary module of the temporal component of our model.

**Integrating the Two Modules.** We combine $S_t$ and $D_t$ and obtain our targeted edge creation process $E_t$:

$$E_t = S_t + D_t, t = 0s, 1s, 2s, ... \tag{2.9}$$

Since the non-stationary periodic component $D_t$ may generate negative values, we set a minimum for the sum to be 0. Note that we designed this temporal component to describe new edge creations aggregated across all the users.

Importantly, this temporal component only generates timestamps of new edges (in terms of the total number of edges created in each second), but does not associate any of these new edges to specific users. In other words, the temporal component will produce the total number of edges created in each second, but will not predict which nodes created these edges. This is because we design the temporal component to specifically capture the edge dynamics aggregated across all the users, *i.e.* property consistent with self-similar scaling and deterministic non-stationary periodic user patterns. The actual distribution or mapping of edge events across users is performed by the spatial component of our model, which we will describe in Section 2.6.2.

### 2.6.2 The Spatial Component

To determine where each new edge is created as part of the overall network evolution process, we first highlight two key observations made by our prior analysis on the Renren network [161]. *First*, after an initial bursty growth phase, new edge creation was dominated by existing nodes (>80%). This empirical result diverges from generative models, which assume that new node arrivals drive edge creation regardless of network size. *Second*, we observe three structural properties over time: graph densification, distance shrinkage, and high but decreasing clustering coefficient (CC). Existing graph models [9, 10, 25, 84] capture only a subset of them.

**Intuition.** We consider a stable social network in a state of ongoing growth[7]. After a fast initial period of explosive growth, the arrival rate of new users becomes relatively small compared to existing users. At this point, continuous friend discovery between existing users dwarfs the initial bursts of edge creations triggered by new user arrivals. Therefore, in our model, we use interarrival gaps between new users as iterations to drive the formation of new edges between existing users.

With these in mind, our model will focus on the creation of edges between existing users following the arrival of each new user. Specifically, we assume a new user $u_i$ creates an edge before the arrival of the next user $u_{i+1}$, and after this edge creation $u_i$ immediately becomes an "existing user." We hypothesize that existing users are often introduced to groups of friends, either discovering the presence of an offline friend (and other mutual friends), or creating new groups of friends via common interests or social applications. To capture this intuition, in each iteration, our model selects two existing nodes $u$ and $v$ at random, and connects $u$ repeatedly to multiple users in $v$'s neighborhood. Here $v$ can be an

[7]Note that our model explicitly targets the ongoing growth phase of a social network. We leave the measurement, analysis, and modeling of a network in decline for future work.

existing friend of $u$ or a previously unknown "stranger." The continuous formation of random connections between existing users shrinks average path lengths and lowers clustering coefficient by building shortcuts between nodes, while connecting friends of friends slows the rate of declustering.

**Model Details.** The spatial component is strongly dependent on the temporal component to determine the maximum number of edges created in any iteration (*i.e.* between two node arrivals). Let $F(n)$ represent the number of edges in the network when the network contains $n$ nodes. Then $F(i+1) - F(i)$ represents the total number of edges created between the arrivals of $u_i$ and $u_{i+1}$. With the knowledge of node arrival time statistics, *i.e.* $t_i$ and $t_{i+1}$, we can estimate the total number of edges $k$ created between $t_i$ and $t_{i+1}$ as $k = F(i+1) - F(i) = \sum_{t=t_i}^{t_{i+1}} E_t$.

Specifically, our proposed edge formation process is defined as follows. We drive the process using a parameter $p$, which defines the probability a node is selected in the recursive edge creation process between existing nodes.

**Algorithm 2.1** Spatial Component of Network Growth.

1. When a new node $u_i$ joins the network, $k = F(i+1) - F(i)$.

2. **Edge creation by the new node:** The new node $u_i$ randomly select an existing node $u_j$ to connect. Set $k = k - 1$. Now $u_i$ becomes an existing node.

3. **Edge creation between existing nodes:** Randomly select two existing nodes $u$ and $v$. If they are not connected, connect them and set $k = k - 1$. Then $u$ starts steps (a)-(c) to connect neighbors of $v$ and repeat them until all the required edges have been created (*i.e.* $k = 0$) or there are no more nodes to connect. Each time an edge is created, set $k = k - 1$.

   (a) Generate a random number $x$ following the geometric distribution with mean $(1 - p)^{-1}$.

   (b) Randomly select neighbors of $v$ that do not connect $u$ until reaching any of the three situations:

      i. $x$ neighbors are selected;

      ii. no more edges need to be created, *i.e.* k=0;

      iii. all available neighbors of $v$ are selected. Let $R$=$\{r_1, r_2, \ldots\}$ be the set of selected nodes.

   (c) For each node $r_i \in R$, $u$ connects $r_i$ and repeats steps (a)-(b) on $r_i$.

4. If more edges need to be created ($k \neq 0$), repeat step (3).

**Comparison to Existing Models.** The existing model most similar to our new model is the Forest Fire model [84], which simulates network growth by creating edges between each new node to a set of existing nodes. A new node joining the network randomly connects to an existing node and some of its neighbors; this repeats across the network, like a fire burning through a forest. This "burning process" and our recursive edge creation process between existing nodes both act to produce high clustering coefficient, by recursively connecting to neighbors of neighbors.

Three key differences separate our model from Forest Fire. *First*, our model captures the observation that existing nodes drive edge creation in a stable growth network. *Second*, our model produces decreasing clustering coefficient by connecting pairs of random existing nodes. Forest Fire does not capture this property because it always forms close triangles in each node's neighborhood, leading to relatively high clustering coefficient unlikely to decrease over time. *Third*, our model can be accurately calibrated to the observed dynamics of an existing network trace, by incorporating the network growth function from the temporal model. This additional flexibility makes it more attractive for generating realistic dynamic network traces.

## 2.7 Model Validation

Having described our proposed model for network edge dynamics in Section 2.6, we next validate the proposed network dynamic model. We calibrate the model using real data and use it to generate synthetic dynamic graphs, and then compare these synthetic graphs to the original data in terms of both temporal and spatial properties.

Since the temporal and spatial components are complementary and operate at different scales, we validate them sequentially to examine their contributions to network evolution. Because the output of the temporal component is used as an input to the spatial component, the validation on the spatial component also serves as validation of the complete model with both components.

Our validation results on the Renren and Facebook datasets lead to the same observations. For brevity, we present the Renren results in detail in Section 2.7.1 and 2.7.2, and summarize the Facebook results briefly in Section 2.7.3.

### 2.7.1 Validating the Temporal Component

Our validation is first based on the Renren dataset of the month of December 2007, the same datasets used in our self-similarity analysis (Section 2.3 and Section 2.4). We leave the validation of Facebook dataset to Section 2.7.3.

To validate our model, we first describe how we calibrate the model using the Renren dataset. As explained in Section 2.6.1, the temporal component consists of two sub-modules: a self-similar module (*i.e.* stationary stochastic process) and a non-stationary module (*i.e.* non-stationary deterministic function).

**Calibrating the Self-Similar Module.** We construct the self-similar module according to the $M|G|\infty$ model described earlier. That is, nodes arrive according to a Poisson process with rate $\lambda$, and the length of each node's active time is chosen independently from a Pareto distribution with parameters $\alpha$ and $x_m$.

Consider the Renren edge creation data collected in December 2007 where $7,246,621$ nodes have created edges. We estimate the corresponding value of rate $\lambda$ in the Poisson process of this period by the average active node count per second, *i.e.* $\lambda \approx 2.7$/s.

To derive the active time (in seconds) statistics, we leverage a proven relationship between $H$ and $\alpha$ [35, 75]: $H = (3 - \alpha)/2$. Since our measured $H$-estimate for the December 2007 data is around 0.65, we set $\alpha = 1.7$.

Finally, assuming a node creates edges at a constant rate of 1/s, the average number of edges created per node is then equal to the average active time across all the nodes, which can be calculated as the mean of the Pareto distribution, *i.e.* $x_m * \alpha/(\alpha - 1)$. By measuring the average edges created per node in December 2007, we get $x_m \approx 3.2$.

Using the $M|G|\infty$-based method with $\lambda = 2.7/\text{s}$, $\alpha = 1.7$ and $x_m = 3.2$, we generate a synthetic trace that represents the edge creation process contributed by the self-similar module. Figure 2.24 plots a randomly chosen 3-hour segment in the synthetic trace, which displays burstiness similar to the original data. By applying the earlier-described R/S, variance and wavelet analysis methods, we get $H$-estimates 0.68, 0.63 (both with a good line fitting) and 0.69 (with the 95% confidence interval 0.0099), respectively. The graphical fitting in Figure 2.25 and Figure 2.26 show that both variance analysis and R/S method have good fit. All these validate that the resulting trace is indeed consistent with the designed-for self-similar scaling behavior (*i.e.* $H = 0.65$).

**Calibrating the Non-stationary Module.** To calibrate the non-stationary module, we first subtract the synthetic trace generated by the self-similar module from the original edge creation data. We then apply a sliding window (with a window size of 1 hour and a step size of 1 second) to smooth the subtraction result over time. One sample of the smoothed data (for December 2007) is shown by the blue curve in Figure 2.27, displaying a daily pattern with almost 0 mean. The blue curve is well-fitted by the sine function: $9.70 \sin(7.27 \cdot 10^{-5}\, t + 3.56) - 0.003$, shown as the red curve.

**Validation Results.** We sum the synthetic traces produced by the above two sub-modules to build a single synthetic edge creation trace, and then compare this

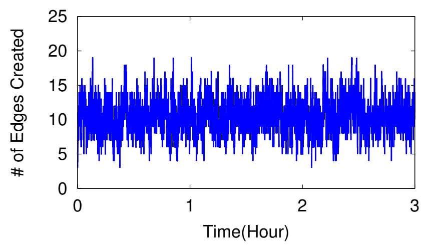


**Figure 2.24:** An example of edge growth of a randomly chosen 3-hour segment in the synthetic self-similar module (Renren).

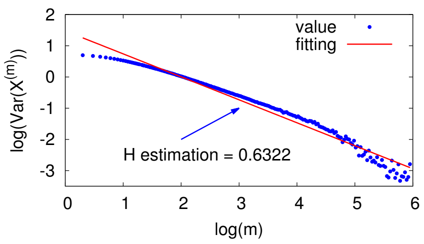


**Figure 2.25:** Variance analysis of synthetic self-similar module: $H$ estimation = 0.67, and in good linear fitting (Renren).

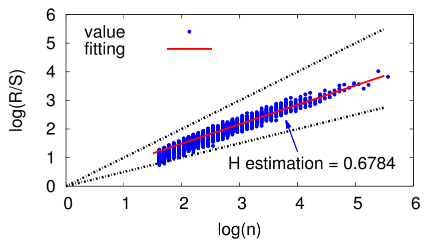


**Figure 2.26:** R/S analysis of synthetic self-similar module: $H$ estimation = 0.63, and in good linear fitting (Renren).

combined trace to the original data. Repeating the process *5 times* produces very consistent outcomes, *e.g.* the total edge counts are similar, with an average ratio between the synthetic and the original trace of 1.007 and variance $< 10^{-6}$. Figure 2.28 plots a sample of one synthetic trace together with the original trace (for December 2007) and illustrates that the synthetic data displays diurnal patterns similar to the original data.

We further compare the synthetic and original traces by performing on the synthetic trace the same self-similarity analysis that we applied in Section 2.3 and Section 2.4 on the original trace. Figures 2.29 (variance analysis) and 2.30

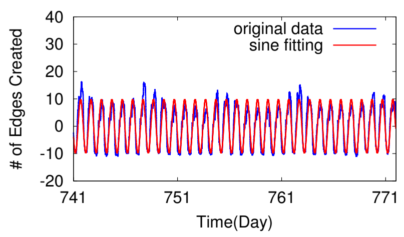


**Figure 2.27:** The synthetic non-stationary module (red curve) well captured the smoothed diurnal pattern in the original dataset (blue curve) (Renren).

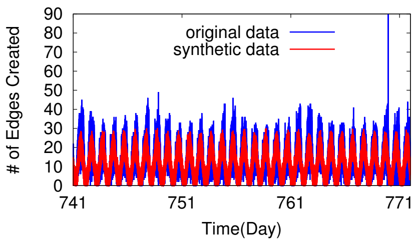


**Figure 2.28:** Synthetic trace by our temporal component (red) vs. original edge creation process (blue) (Renren).

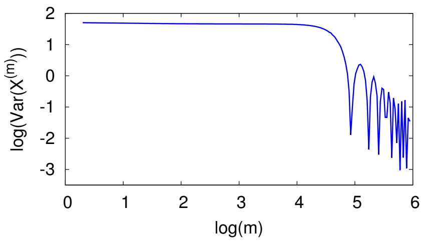


**Figure 2.29:** Variance analysis of the entire synthetic trace: like the original data, slope also changes for $m>10^4$ seconds ($\approx$ 3 hours) (Renren).

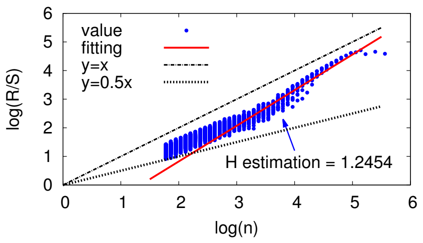


**Figure 2.30:** R/S analysis of the entire synthetic trace: like the original data, H-estimate is beyond the self-similar range, and data shape changes $n>10^4$ seconds ($\approx$ 3 hours) (Renren).

(R/S analysis) demonstrate that the synthetic trace exhibits the very same issues that plagued our preliminary analysis of the original data; *e.g.* scaling behavior changes drastically for time scales larger than a few hours, $H$ estimation is outside the theoretical range $(0.5, 1.0)$, and thus non-stationary diurnal patterns prevent a direct scaling analysis of the data.

Next we apply the wavelet-based analysis method to examine the self-similar nature of the synthetic trace over 3-hour segments. Figure 2.31 plots the resulting

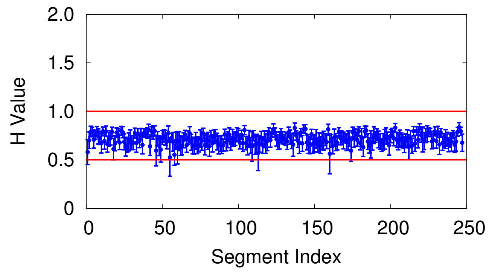


**Figure 2.31:** Wavelet analysis on 3-hour segments of synthetic trace. Like the original data, the vast majority of segments have estimated $H$ within (0.5,1) (Renren).

$H$-estimates for each segment with 95% CI. We see that the $H$-estimates for the synthetic trace also fall consistently between $(0.5, 1)$ with an exception of 4.03%, which closely matches the 3% exception seen from the original data. The average H value for the synthetic trace is around 0.75, again similar to that of the original trace (mean $H = 0.65$) as shown in Figure 2.9. Finally, we evaluate the robustness of our results by shifting the starting time of each segment by 0, 1, and 2 hours separately, and find both the abnormal segment ratios and $H$ estimates remain stable (we omit the results for brevity). Thus we conclude that the original trace and the synthetic traces are qualitatively and quantitatively similar.

Together, these results demonstrate that the temporal component of our model can accurately capture the diurnal patterns and self-similar scaling behavior displayed by the original Renren data. Furthermore, the contributions of the two sub-modules illustrate why and how the presence of deterministic non-stationary

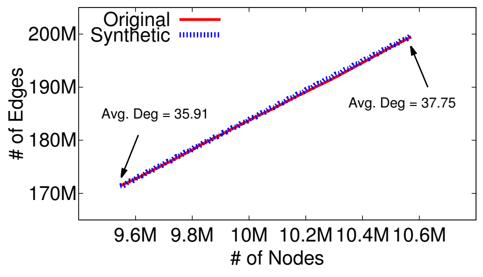


**Figure 2.32:** Network growth of the synthetic trace generated by the temporal component vs. the original data (Renren).

periodic trends like diurnal user activity patterns impacts any direct scaling analysis of such non-stationary data.

**Connecting the Temporal and Spatial Components.** Recall that the spatial component of our model uses the temporal component to compute the number of edges created between each pair of node arrivals. As a result, we need to be able to accurately estimate the arrival time of each node. From our exploratory analysis, we noticed no specific properties of the node arrival process other than that it is largely consistent with a Poisson process with rate $\lambda_{new}$, where $\lambda_{new}$ is estimated as the average number of new node arrivals per second.[8] Figure 2.32 shows that our solution can accurately predict the network edge growth in December 2007.

[8]This is analogous to the observation in [115] that while packet arrivals in network traffic appear better modeled using self-similar processes, Poisson effectively captures user session arrivals.

| Graph | # of Nodes | # of Edges | Avg. Deg | Avg. path | Avg. CC |
|---|---|---|---|---|---|
| 2006 Original | 624,364 | 8,258,266 | 26.45 | 4.16 | 0.159 |
| 2006 Synthetic | 624,364 | 8,721,927 | 27.93 | 4.46 | 0.183 |
| 2007 Original | 1,751,146 | 18,203,520 | 20.79 | 4.87 | 0.156 |
| 2007 Synthetic | 1,751,146 | 18,305,972 | 20.9 | 4.84 | 0.161 |

**Table 2.4:** Statistics of the original graph and the synthetic graph generated by our spatial component for Renren dataset. The 2006 graphs are built before December 12, 2006; the 2007 graphs are built for January - February, 2007.

### 2.7.2 Validating the Spatial Component

Next, we validate our spatial component. Ideally, we would calibrate the model using the entire Renren dataset (from November 21, 2005 to December 31, 2007) and produce synthetic traces for the entire 25-month period. However, using the entire dataset is impractical for two reasons. First, due to the size of the network at the end of the 25-month period (*i.e.* 10.6M nodes and 199M edges), the calibration process would be computationally prohibitive. Second, the merge event on December 12, 2006 introduced significant changes to the network, impacting any analysis of the network's dynamics.

As a viable practical alternative, we use two subsets of the Renren data for validation. The first segment (referred to as *2006 Original*) covers the period from the launch of the network (November 21, 2005) till right before the merge event

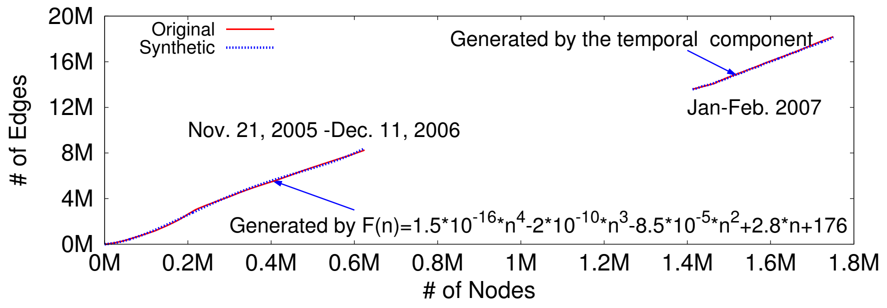


**Figure 2.33:** Fitting of network growth with the network edge growth function $F(n)$ (Renren).

(December 11, 2006). The corresponding last snapshot of the graph includes 624K nodes and 8M edges. This represents the "early" period of the network. The second segment (*2007 Original*) covers the first two months of 2007, with the snapshot on December 31, 2006 as the initial graph, and its last snapshot has 1.75M nodes and 18M edges. This represents the "stable growth" period of the network. Table 2.4 summarizes the observed network statistics for the two segments.

**Spatial Component Calibration.** We calibrate the component for the two segments separately. As discussed in Section 2.6.2, the spatial component has two parameters: *network edge growth function* $F(n)$ and *node selection probability* $p$. For the 2007 segment, we derive $F(n)$ from the temporal component. For the 2006 segment, however, we have to manually fit the network growth by a polynomial function. This is because our measurement shows that in 2006 the network is not stable and large enough to display significant temporal patterns. Figure 2.33 shows the $F(n)$ estimation results for both segments, which closely match the original data.

Next, we follow the methodology by [127] to determine $p$. We generate a series of synthetic graphs with $p$ varying between $(0.1, 0.9)$, and choose the best $p$ value that produces graphs with network distance and clustering coefficient most similar to the original data. The resulting $p$ values are different for the two segments: 0.7 for the 2006 segment and 0.5 for the 2007 segment.

**Validation Results.** Using the calibrated component, we generate synthetic dynamic graphs for the two data segments. As shown in Table 2.4, the synthetic graphs statistically match the original graphs in the corresponding last snapshot, in term of average degree, average path length and average clustering coefficient (CC). The emphasis of our validation is to understand whether synthetic graphs display the three dynamic properties observed from the Renren social network [161]: graph densification, average path length shrinkage and decreasing clustering coefficient.

Using the network growth function $F(n)$, Figure 2.33 confirms that the synthetic graphs can accurately capture the densification property. Thus in the following, we focus on evaluating dynamics of average path length and average clustering coefficient in synthetic graphs. As a reference, we also include the results

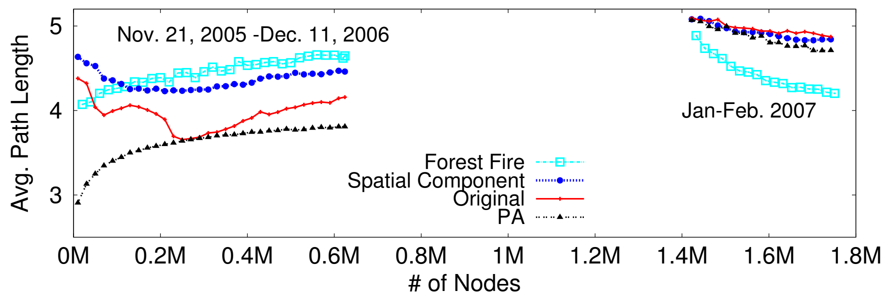


**Figure 2.34: Average path length** on generated synthetic graphs and the original Renren graph. Include two time periods from the very beginning to December 11, 2006 and in January - February, 2007 (to avoid the one-time merge event in Renren with another OSN). (Original: Renren graph; Spatial Component: graph generated by our spatial component; PA: graph generated by the preferential attachment model; Forest Fire: graph generated by the Forest Fire model).

using the Preferential Attachment model [17], which is the most popular static graph model, and the Forest Fire model [84][9].

We repeated our experiments *five times* for all three models, and obtained consistent results, with the variance across all runs at least three orders of magnitude smaller than the average value. Thus for brevity we only show the result for a single run.

*Average Path Length Evolution:* Figure 2.34 plots the average path length over time using our spatial component, the Preferential Attachment model, the Forest Fire model and the original data. For the 2006 segment, our spatial component

[9]Following a similar procedure described by [127], we modify the Forest Fire model to produce undirected graphs by creating undirected edges and allowing the "burning" process to proceed in both directions of an edge. To calibrate the model, *i.e.* determining the burning probability $p$, we sample values between $(0, 1)$ to find the best fit $p$ where the corresponding synthetic graphs match the original graph the most in terms of network distance and clustering coefficient.

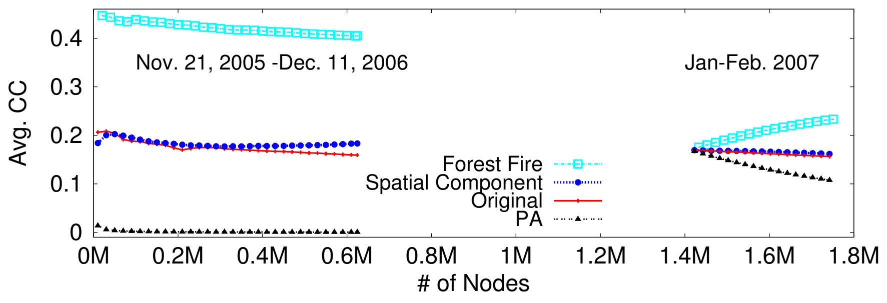


**Figure 2.35: Average Clustering Coefficient** on generated synthetic graphs and the original Renren graph. Include two time periods from the very beginning to December 11, 2006 and in January - February, 2007 (to avoid the one-time merge event in Renren with another OSN). (Original: Renren graph; Spatial Component: graph generated by our spatial component; PA: graph generated by the preferential attachment model; Forest Fire: graph generated by the Forest Fire model).

displays the most similar pattern to the original data, where the path length decreases first and then increases slightly, while the Preferential Attachment and Forest Fire models produce increasing path length. For the 2007 segment, while all four graphs display a decreasing pattern over time, our spatial component is the closest to the original graph. In this segment, behaviors of the Preferential Attachment and Forest Fire models change because the snapshot of the original data on December 31, 2006 is used as the initial graph, removing the long-term impact of preferential attachment [161] that produces increasing average path length over time.

*Average Clustering Coefficient Evolution:* Figure 2.35 plots the results for the average clustering coefficient from the three models and the original data. For the 2006 segment, only our spatial component behaves similarly to the original data, with an average clustering coefficient in $(0.15, 0.22)$, while that of the Preferential Attachment model stays closely to 0 and the Forest Fire model remains above 0.4. For the 2007 segment, again our spatial component produces nearly identical value of the original data, while the results of the Preferential Attachment and Forest Fire model deviate largely. Together, these results confirm three key findings. *First*, our spatial component can accurately capture the significant local connectivity and the slowly decreasing clustering coefficient. *Second*, the Preferential Attachment model is unable to maintain high clustering coefficient over time, even when growing from a highly clustered graph. *Finally*, as indicated by our earlier analysis, the Forest Fire model produces relatively high clustering coefficient, unable to capture the key properties of Renren such as decreasing clustering coefficient.

**Summary of Results.** Our validation confirms that the spatial component can accurately capture key dynamic features observed in Renren dataset. Since our 2007 synthetic trace takes input from the temporal component of our model, the spatial component validation also provides an overall validation of our proposed model.

### 2.7.3 Facebook Results

We now briefly summarize how we validate our model using the Facebook dataset, since the methodology is very similar to what is applied to the Renren dataset. Like Renren, our results on Facebook datasets also strongly validate the effectiveness of our model.

*First*, we validate the temporal component by calibrating the self-similar module and the non-stationary module, and then producing a synthetic edge creation trace (repeated 5 times). The total edge count matches the original data, *i.e.* the average ratio between the original and synthetic traces is 1.05 with variance $< 10^{-5}$. The wavelet-based analysis on the synthetic traces leads to results consistent with that of the original trace. Specifically, for 3-day segments, H estimates with 95% CI fall between $(0.5, 1)$ with an exception of $< 1\%$. The exception ratio (the portion of abnormal segments whose $H$ estimates are out of (0.5,1)) grows to 25% for 15-day segments, and 100% for 20-day segments.

*Second*, the spatial component for Facebook is slightly different from that of Renren because Facebook wall posts can lead to multiple edges between a node pair (while Renren only has one per node pair). Thus we modify our model, as well as the Forest Fire and Preferential Attachment models, to allow duplicated edges. We grow the three models from 0 node to the total number of nodes $46,952$ in the Facebook dataset. We compare the synthetic traces generated by the three models

| Graph | Avg. Degree | Avg. Path length | Avg. CC | Final avg. Degree | Final avg. Path length | Final CC | Final # of Edges |
|---|---|---|---|---|---|---|---|
| Facebook Original | 32.293 | 5.760 | 0.101 | 37.357 | 5.630 | 0.108 | 876,993 |
| **Synthetic (our model)** | 32.508 | 3.650 | 0.122 | 37.545 | 3.645 | 0.118 | 881,415 |
| Forest Fire | 70.273 | 3.836 | 0.469 | 69.509 | 4.030 | 0.446 | 1,631,792 |
| Preferential Attachment | 35.960 | 2.890 | 0.012 | 35.986 | 2.996 | 0.006 | 844,812 |

**Table 2.5:** Statistics of the original Facebook graph, the synthetic graph generated by our spatial component, by the Forest Fire model and by the Preferential Attachment model. Path length and clustering coefficient (CC) do not consider multiple edges between node pairs. All standard deviations are less than 4%. Columns 2-4 refer to averaged results for intermediate graph snapshots, Columns 5-8 refer to the final graph snapshot (Facebook).

and the original data (see Table 2.5). Again, our results show that our model can accurately capture the growth of the Facebook trace. Its average node degree and clustering coefficient, for both intermediate and final snapshots, are almost identical to the original data, while the Forest Fire and Preferential Attachment models produce large deviations.

### 2.7.4 Summary

Our results on the Renren and Facebook datasets consistently show that our model can successfully capture both the temporal properties of graph dynamics, in terms of self-similar scaling and deterministic non-stationary trends in terms of periodic patterns, and its spatial properties observed, including long-term graph distance shrinkage and reduction in local clustering.

## 2.8 Related Work

### 2.8.1 Self-similarity Measurements and Models.

Self-similarity describes the phenomenon where a property is preserved with respect to scaling in space and/or time. If an object is self-similar, its parts, when magnified, resemble the shape of the whole [114]. Previous works have studied *structural self-similarity* in networks [51, 136], *i.e.* the scale-invariance properties of physical structures of a graph (*e.g.* node degree or community size distribution) under coarse-graining of vertices. Our work differs by studying self-similar scaling properties on time dynamics, *i.e.* "temporal" self-similarity, which has not been studied in social networks.

**Self-similarity Measurements.** Temporal self-similarity describes the scaling properties of certain statistics (*e.g.*, variance, R/S, wavelet coefficients, finite-dimensional distributions) of a time series when computed at different time scales [114]. It has been detected in diverse contexts such as ecology, life sciences and stock markets [44], and was first introduced to network traffic for the purpose of modeling the bursty characteristics observed in Ethernet LAN traffic [76, 75]. Later studies show self-similarity has also been observed in other network traffic scenarios, including wide-area traffic [115], world wide web traffic [36], disk-level I/O [122], HTTP traffic traces [41], variable-bit-rate video [21, 46], blog posts [48], messages [126] and emails [44] in communication networks. Note that these empirical studies show that in practice, self-similar property is typically observed over a finite range of time scales [5, 46, 50], and is difficult to discern at both very small and very large time scales.

**Self-similarity Models.** Generally speaking, there are two classes of self-similar models. The first are purely mathematical models, *e.g.* fractional Gaussian noise [100], fractional Brownian motion (FBM) [100], fractional ARIMA processes [61] and b-model [149]. They are strictly descriptive and cannot explain the root cause underlying the formation of self-similarity. The second class seeks to provide physical reasons behind self-similarity. Inspired by the renewal reward process in economics [140], the superposition of many ON/OFF sources [152, 50] captures the observed self-similar nature of Ethernet LAN traffic, if the durations of the ON- or OFF-periods have a heavy-tailed distribution. The $M|G|\infty$ queuing model [35, 115, 114], where sources arrive according to a Poisson process and each source is active for a duration that is described by a heavy-tailed distribution, can also successfully explain self-similar phenomena.

### 2.8.2 Graph Models

In general, graph models can be classified as static graph models or dynamic graph models.

**Static Models.** We further classify static models into three sets. One set includes feature-driven models designed to capture one or more static graph features, *e.g.* small-world [150], power-law degree distribution [17, 60], and high clustering coefficients [60]. A second set includes intent-driven models that try to explain the underlying process of graph formation. Nearest neighbor models [145, 38, 143], Random walk models [24, 145] and copying models [71, 145] belong to this set. Finally, a third set of models generates graphs based on graph structural statistics instead of graph features. Kronecker graphs [80] apply Kronecker multiplication to generate graphs similar to real graphs. The dK-series model [99] uses subgraph degree distributions to capture increasingly detailed representations of graph structures. Finally, [127] proposes a general technique to produce "realistic" synthetic graphs by calibrating graph models using real graphs.

**Dynamic Models.** In contrast, dynamic models aim to capture dynamic features of graphs. [18] modifies preferential attachment model to capture graph densification. [84] proposes a Forest Fire model to capture both graph densification and diameter shrinking properties in networks. Later models [101, 155] captures similar properties. The dynamic copying model captures the property of decreasing clustering coefficients, but not the power-law degree distribution [25]. Based on graph structure statistics, [10] proposes a 3D Kronecker model. [9] is a model based on random typing statistics to capture several graph dynamic features. Unlike our work, [9] is not modeled after empirical data of graph dynamics. [78] designs a model of network evolution, but focuses on reproducing desired structural properties in the *final* snapshot. Finally, [109] tries to include capturing the network harshness into the model, *i.e.*, how likely a node will be lost, but also cares about the final structural statistics only.

## 2.9 Summary

Starting from the exploration of self-similarity properties, which is critical of determining how to model graph dynamics, our work in this chapter takes a concrete step towards studying the detailed dynamics of social networks. We focus on " time-stamped" traces of network growth, *i.e.*, network includes detailed timings of when nodes arrive and edges are created. By performing empirical studies of network dynamics over two detailed, time-stamped traces of social networks over multiple years, *i.e.*, Renren dataset and Facebook wall post dataset, we have detected that the edge creation process in both networks does perform properties consistent with self-similar scaling. We have also quantified that such properties hold from seconds to hours, and gradually weaken at larger time scales due to the existence of non-stationary patterns introduced by human behaviors, *e.g.*, diurnal or weekly user activities.

Specifically, we find that edge creation process in the two OSNs is non-stationary over long-term periods, and the two traditional techniques for self-similarity detection, *i.e.*, R/S and variance methods, produce inconclusive results and are unsuitable for measuring self-similarity in real traces in social networks. By applying the more robust wavelet-based method against underlying non-stationary trends, we find the edge creation process in both network traces does exhibit properties consistent with self-similarity over time scales ranging from seconds to hours.

We leverage this new result to propose a comprehensive model of graph dynamics, including a temporal component that defines when and how many new edges are formed across all the users, and a spatial component that defines where in the graph new edges form. Our temporal component captures the coexistence of long-term non-stationary periodic structure, *e.g.* diurnal or weekly patterns, and properties consistent with self-similarity at shorter time scales, while our spatial component is a dynamic graph model that simulates edge creation process driven primarily by existing users, and captures graph densification, shrinking network diameter, and decreasing local clustering.

Our detailed validation efforts on both datasets consistently show that our model accurately captures both the temporal properties of graph dynamics, in terms of self-similar scaling and certain deterministic non-stationary features, and also many key dynamic structural properties of the two social graphs over time.

By providing such a practical method for generating a realistic sequence of time-stamped events that uniquely define the formation and evolution of a social network in time and space, our model fills an existing void in network dynamics, and addresses an urgent need for accurate models that account for both temporal and spatial features in network dynamics.

# Chapter 3

# Reassessing Current Status of Link Prediction

## 3.1 Introduction

[1] The access to real, large-scale traces of network dynamics brings us great opportunity to rethink about some fundamental graph problems. Have they been well addressed, or how far have we come in understanding them in real-world scenario? What lessons can we draw from the successes (and failures) of current studies? Can we improve them by leveraging more data? In this dissertation, we

[1] ©ACM, (2016). This is the authors version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. Abbreviated version of content in this chapter can be found in paper "Network Growth and Link Prediction Through an Empirical Lens" [92], Proceedings of the ACM Internet Measurement Conference (IMC'16),http://doi.acm.org/10.1145/2987443.2987452.

focus on the link prediction problem, *i.e.*, the problem of predicting formation of new edges on a given network.

Link prediction is a fundamental problem that applies to networking in numerous contexts, including the Internet, the web, and online social networks. The sheer number of studies, including proposals for algorithms and models [7, 11, 32, 52, 53, 69, 89, 90, 95, 110, 111, 120, 142, 148], underscores the importance of the problem to a variety of applications, ranging from resource allocation in online services to offline efforts in counter-intelligence and counter-terrorism [68, 70].

As a technical problem, the efficacy of link prediction is generally not well understood. Today, link prediction algorithms are the basis for social recommendations in a wide range of social networks and applications, ranging from Facebook and Pinterest to personal streaming on Periscope and Q&A sites like Quora. The success of these sites and the sheer volume of prior literature lead many to believe the problem is well addressed. Only evidence to the contrary comes from anecdotes of failed recommendations that trigger potential privacy concerns [58].

Despite years of research in this space and hundreds of publications (only a small subset of which is cited here), there has been little opportunity to study these proposals from an empirical perspective. Until recently, public datasets of network dynamics have been limited to co-authorship studies and patent citation graphs, moderate sized networks which scale up to 20K nodes and 200K edges [15, 130]. In contrast, algorithms developed using and validated by these datasets are targeting dynamic networks that are two or more orders of magnitude larger, with millions or billions of nodes and billions of edges [161].

Thankfully, things are changing with the arrival of network traces from online social networks (OSNs). We are taking advantage of this opportunity (and availability to large traces of network dynamics) to step back and reassess the space of link prediction algorithms from an empirical perspective. We are motivated by questions such as:

- *How far have we come in understanding network growth and predicting the underlying processes that drive it? How far do we have to go?*
- *What lessons can we draw from the successes (and failures) of existing algorithms?*
- *Can we improve existing approaches by leveraging more data, e.g. detailed temporal network history?*

In this work, we perform an empirical study using large traces of network growth from three large OSNs, Facebook, Renren (Facebook equivalent in China), and YouTube. In each case, detailed timestamps capture the time when specific edges were created between nodes (users) in the network. To the best of our knowledge, these are the only publicly available datasets suitable for this study, both sufficiently large and with sufficiently detailed timestamps to capture graph dynamics. These traces cover substantial subsets of users in each network, and in the case of Renren, the entire user population at a time when the network included 10 million users. We discretize these traces into numerous temporal snapshots, and use them to drive the evaluation of 18 representative link prediction algorithms. Finally, we use our lessons and observations from analyzing these network dynamics to build "filters" that help prune the set of candidate nodes for edge creation. By applying these filters before link prediction, we can reduce the search space and focus on regions of likely growth.

To better understand and compare results across prediction algorithms, we classify them into two groups. First, *Metric-based prediction algorithms* define specific metrics that can be computed for all potential links, where a potential link with a higher score on the metric indicates a higher probability of formation. For our analysis, we implemented 14 distinct metrics that had scalable algorithms in existing literature. In contrast, *classification-based prediction algorithms* utilize machine learning classifiers that take multiple metrics as input features, and produce a prediction of likelihood of formation for each potential link. Some methods produce a detailed probability while others produce a binary result. Experiments in our study cover support vector machines (SVM), logistic regression, naive Bayesian networks, and random forests, each using all 14 metrics as input features. Our experiments show that more complex techniques, *e.g.* larger ensemble methods do not produce noticeable improvements in accuracy.

As our first result, we find that link prediction performance remains poor in absolute terms. Correctly predicting link formation within some timeframe is difficult, and the problem only grows harder, as each new node brings $\sim N$ more potential links to a network of size $N$. Second, we find that for each of our traces, metric-based prediction algorithms vary significantly in accuracy. In each case, a small subset of metric-based predictors do as well as (and occasionally outperform) the most accurate machine learning based classifier (SVM in all cases). We note that while a few metrics perform consistently well, no single metric predictor consistently performs as well as SVM across all networks. Instead, there appears to be a strong correlation between network structure and the relative success of specific metric-based algorithms. Machine learning methods do well in part because they automatically adjust weights across different metrics, emphasizing those that match the targeted network without *a priori* knowledge of its structure. Without such knowledge, we can either achieve "good" accuracy by choosing a consistently strong metric, or achieve "near optimal" accuracy by using a ML classifier (at the cost of higher computational and training costs).

Finally, we revisit existing prediction algorithms with the goal of augmenting them by leveraging knowledge of past network dynamics. Our insight is to provide "temporal filters" that significantly reduce the set of potential new links, reducing the search space and computational cost, while focusing predictors on more probable link candidates. Our filters are focused around trends in node activity and potential link d istances, both patterns observed in this and prior studies of network dynamics. Applying these filters produce very encouraging results, in many cases effectively doubling the predictive power of both metric-based and classifier-based algorithms. Not only do these filters outperform recent methods leveraging temporal information, but they can be combined with temporal methods to provide even better results.

In this chapter, we make three key contributions. First, we carry out a comprehensive analysis of a wide range of link prediction algorithms, studying not only their performance but also possible causes of low prediction accuracy. We apply decision tree classifiers to identify the best metric-based algorithms for different networks.

Second, we compare the two categories of link prediction methods, *i.e.* metric-based and classification methods, study their cost versus accuracy tradeoffs, and identify strategies for choosing between them.

Finally, we leverage insights from analysis of network growth to design filters that improve prediction accuracy by dramatically reducing the search space. In our tests, these filters significantly improve prediction power across all methods.

Further, they outperform recent proposals that integrate temporal information, and can be combined with them to produce even better results.

## 3.2 Background: Link Prediction

Link prediction identifies new edges that will likely form in the near future, by analyzing the structure of the current network [89]. Given a graph $G_t =<V_t, E_t>$ observed at time $t$, it seeks to predict new edges to be created between nodes $V_t$ at time $t'$ ($t' > t$).[2] Note that we focus on predicting *future* links at some time $t$, which is different from the detection of *missing* links, where given a partially observed graph, it identifies link status for unobserved pair of nodes [66, 103].

Existing link prediction algorithms naturally fall into two categories, which we refer to as *metric-* and *classification-based.* We list and classify all of the known popular prediction algorithms in Table 3.1, which are algorithms we focus on in this work, and their details will be introduced later in Table 3.3.

Metric-based algorithms estimate the likelihood of future connectivity between unconnected nodes, by generating a numeric score based on some graph-based heuristic [89] or models [32, 120]. All potential node pairs are sorted by score to determine the most likely future edges.

[2]This is the most common form of link prediction. It does not consider edges created by new nodes who join after $t$, which is referred to as the cold start link prediction problem [77], nor edges that might disappear after their creation.

| Metric-Based Prediction | | | Classification-Based Prediction |
|---|---|---|---|
| Heuristics | Learning Models | | *e.g.*, SVM, Logistic Regression, Naive Bayes, Random Forest |
| *e.g.*, CN, JC, AA, RA, LP, SP, PA, Katz, LRW, PPR | Probabilistic Models | Matrix/Tensor Based | |
| | *e.g.*, BCN, BAA, BRA | *e.g.*, Rescal | |

**Table 3.1:** Summary of link prediction algorithms, with details listed in Table 3.3.

In contrast, classification-based algorithms treat link prediction as a classification problem [11]. Using scores by metric-based algorithms and maybe other information as training features, these classifiers then "separates" the node pairs that will likely connect in the near future from those that will not. Some classifiers also produce a granular similarity score, which can be used to rank node pairs.

Next, we describe these two categories of algorithms in detail and highlight their differences.

**Metric-based Prediction.** Metric-based link prediction algorithms quantify and rank node pairs by their likelihood of forming new edges, based on specific metrics that capture similarity or proximity between nodes [89]. For simplicity, we refer to the entire group as "similarity metrics," and further divide them into *heuristics*, or more complicated *learning models*, as shown in Table 3.1.

Many popular metric-based algorithms are *heuristics* based on common intuitions of graph formation [89], *e.g.*, two currently unlinked nodes with the most commonly connected nodes are most likely to link in the future. Those hypotheses are driven by graph structural properties and do not require metadata. They generally focus either on node neighborhood information, where they capture properties of the common neighborhood between nodes of 2-hop distance, *e.g.* Common Neighbors [110] and Adamic Adar Index [7], or on path properties such as shortest path length [69].

Link prediction can also be performed by inferring the likelihood of two nodes forming an edge based on *learning models.* One way is to use probabilistic models calibrated by measurements on $G_t$. For example, [32, 52] assume a specific underlying structure of hierarchies or communities exists in the graph $G_t$, and model parameters are estimated using maximum likelihood.

Another approach is to extend the field of relational learning to link prediction [142, 148]. However, these underlying models either do not scale to large graphs (due to complexity in parameter learning) or rely on special conditions that do not generalize to common networks such as social networks [148]). Only the local naive Bayes model [95] meets the needs of large, generalized graphs.

Other metrics use matrix (tensor) techniques on matrix representations of graphs. They capture node similarity in a latent space, defined by different models [111, 120]. Among them, only *Rescal* [111] has been shown empirically to scale to large graphs of millions of nodes.

**Classification-based Prediction.** While metric-based algorithms are known for their simplicity [89], performance can vary significantly depending on the specific similarity metric used. Existing work has shown the best metric varies across datasets and there is no unified solution [89, 90].

The alternative is what we call classification-based methods. Instead of using a certain similarity metric, one can build automated classifiers to explore multiple similarity features [90]. Compared to single metrics, classifiers face the challenge of high computational complexity, (*e.g.* feature selection and training), especially for massive OSNs [53].

## 3.3 Datasets and Methodology

We now describe the datasets used for our study and our experimental methodology.

### 3.3.1 Datasets

Our study uses large traces of dynamic network growth from three different networks, Renren, Facebook, and YouTube. As far as we know, these are the only publicly available large-scale datasets suitable for this study, which have

| Graph | Trace Start | | | Trace End | | | Time | Snapshot | # of |
|---|---|---|---|---|---|---|---|---|---|
| | Date | Nodes | Edges | Date | Nodes | Edges | Granularity | Delta ($k$) | Snapshots |
| Facebook (New Orleans) [147] | 09/05/06 | 48,969 | 339,098 | 01/21/09 | 63,731 | 817,090 | Seconds | 15K | 31 |
| YouTube (Snowball Crawl) [105] | 02/09/07 | 1,406,188 | 3,466,440 | 07/23/07 | 3,223,589 | 9,376,594 | Days | 250K | 21 |
| Renren (Non-sampled) [161] | 01/01/07 | 1,413,731 | 13,616,792 | 12/31/07 | 10,572,832 | 199,564,006 | Seconds | 10M | 17 |

**Table 3.2:** Statistics of the three OSN datasets.

sufficiently detailed timestamps to capture graph dynamics, *i.e.*, the time when each edge (link) was created between nodes (users).

The Renren [161] data includes creation of every edge in the entire Renren network during a period of over 2 years (from its first edge, to 10 million users, 199 million edges when the trace ends). The Facebook trace [147] includes edges created in the New Orleans regional network over 2+ years. The YouTube trace [105] includes edges recorded from daily snowball crawls of a user community that grew from 1 million to 3 million users over a period of 5 months. To avoid disruptions from external events, *i.e.*, the network policy changes in Youtube, and a one-time network merge event for Renren (Renren merged with its largest competitor in December 2006), we use continuous subtraces that do not include the external events in question. Statistics on all three traces are summarized in Table 3.2.

We show each network's daily growth in nodes and edges in Figure 3.1.

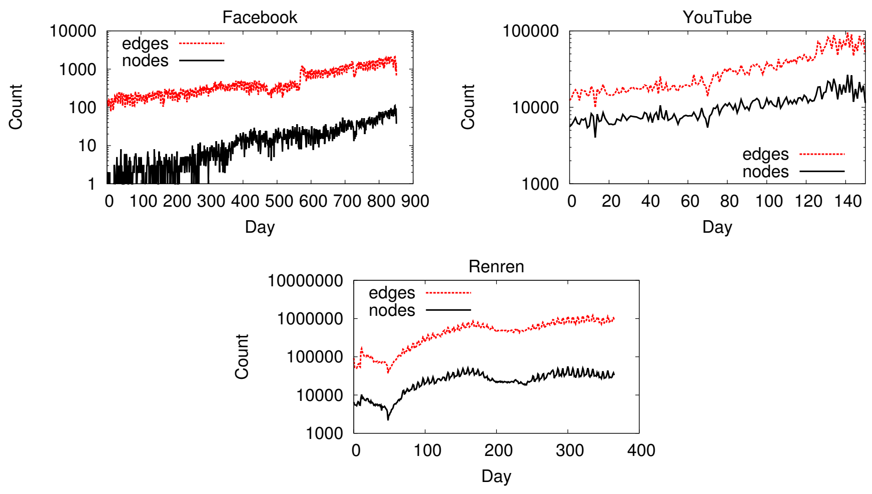


**Figure 3.1:** Daily new nodes and edges in the three networks.

While the three networks all continue on exponential growth trajectories (Facebook has a number of 49K users at the beginning while the other two has 1.4M users), we see Renren is on a much faster growing pace. This is because both the Facebook and YouTube datasets are sampled networks, *i.e.*, Facebook dataset is a regional network and YouTube dataset depicts the growth of a user community. Figures 3.2-3.4 provide a quick look at the change in basic network properties over its evolution, including average node degree, path length, and clustering coefficient. Unsurprisingly, average node degree for all three networks grows over time. In comparison, Renren and Facebook are much denser than YouTube. Unsurprisingly, networks grow and densify over time, and their average path length shrinks. YouTube has the largest path length due to its sparsity.

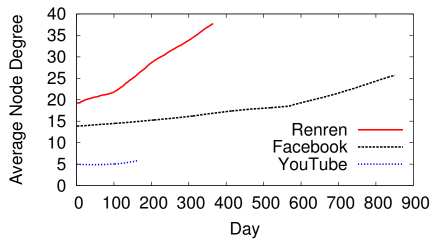


**Figure 3.2:** Average node degree.

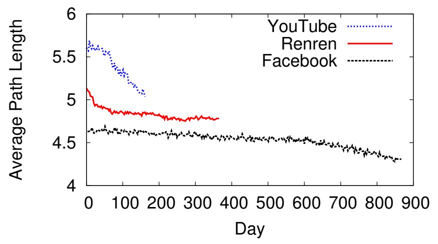


**Figure 3.3:** Average path length.

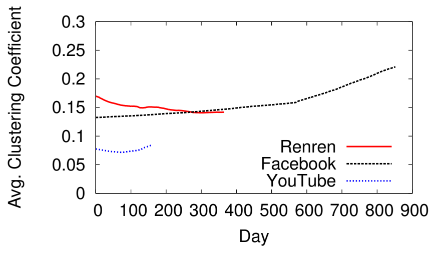


**Figure 3.4:** Average clustering coefficient.

### 3.3.2 Methodology

Existing link prediction studies focus on predicting edges between two static snapshots [89, 11, 45], and most do not capture the evolution of fast growing networks such as OSNs like Facebook, LinkedIn and Renren. In contrast, our work seeks to answer two key questions:

**Q1:** Can existing algorithms accurately predict the continuous edge (or link) growth of today's large, dynamic, online social networks?

**Q2:** Can we utilize temporal network data to improve prediction accuracy?

**Evaluating Link Prediction on Graph Sequences.** To answer these questions, we apply a *sequence-based* framework to evaluate existing link prediction algorithms as the network grows. We process each dataset to generate a sequence of graph snapshots $(G_1, G_2, ..., G_T)$ while keeping the number of new edges created in each snapshot constant. We refer to this number as the *snapshot delta*.

We run each algorithm in every graph snapshot $G_{t-1} (1 < t \leq T)$ to predict new edges (among existing nodes) that will appear in the next snapshot $G_t$, and compare them to the ground truth, *i.e.* the actual new edges found in $G_t$. We choose the snapshot delta value to ensure sufficient number of snapshots for analysis ($> 15$) while ensuring duration between two successive snapshots is not too long ($< 2$ weeks). Specific values and the resulting number of snapshots for each dataset are listed in Table 3.2.

To address Q1, we evaluate 14 different metric-based algorithms that can scale to our large datasets, and 4 widely used classification-based algorithms. For each algorithm, we study its prediction accuracy as the network grows and identify potential causes for any loss of accuracy.

We answer Q2 by analyzing the temporal properties in edge creation to identify and utilize trends as additional metrics to improve prediction accuracy.

**Implementing Metric-based Algorithms.** We first cover 10 most popular heuristics: Common Neighbors (CN), Jaccard's Coefficient (JC), Adamic/Adar Index (AA), Resource Allocation Index (RA), which focus on capturing properties of the common neighborhood between nodes of 2-hop distance; Preferential

Attachment (PA), which is based on node degree; Local Path (LP), Local Random Walk (LRW), Shortest Path (SP), Personalized PageRank (PPR), and Katz, which are driven by path properties. We also include 1 tensor-based algorithm, *i.e.*, Rescal [111], which works by condensing the interaction among nodes into a latent space. And finally we implement 3 probabilistic algorithms [95], which account for different roles by different common neighboring nodes between node pairs, *i.e.*, Local Naive Bayes based Common Neighbors (BCN), Local Naive Bayes based Adamic Adar (BAA), and Local Naive Bayes based Resource Allocation (BRA). These cover the metric-based approaches (see Table 3.1), and we summarize their detailed implementation in Table 3.3.

We also fine-tune our implementation by identifying the best parameters and approximation methods (if any) based on results of our own experiments and from prior studies. Specifically, LP requires a weight parameter $\epsilon$ for 3-hop paths, and $\epsilon = 0.0001$ provides the highest accuracy. For PPR, we configure the restart probability $\alpha = 0.15$ as suggested by prior work [16]. For Katz, we set $\beta = 0.001$ as suggested by [6], and implement two approximation methods: low rank approximation (Katz$_{lr}$) [6] and scalable proximity estimation (Katz$_{sc}$) [137]. Our experiments in §3.4 show that while more accurate than Katz$_{sc}$, Katz$_{lr}$ does not scale to larger networks, since it computes Katz score for all candidate node pairs[3].

[3]Even using 8 machines with 192GB memory each, calculating Katz$_{lr}$ for a Renren snapshot with 185M edges takes 27 days.

| **Metric-based Algorithms** | **Precise Formulation** |
|---|---|
| CN (Common Neighbors) [110] | $\lvert\Gamma(u) \cap \Gamma(v)\rvert$ |
| JC (Jaccard's Coefficient) [89] | $\frac{\lvert\Gamma(u)\cap\Gamma(v)\rvert}{\lvert\Gamma(u)\cup\Gamma(v)\rvert}$ |
| AA (Adamic/Adar) [7] | $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{\log(deg(w))}$ |
| RA (Resource Allocation) [165] | $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{deg(w)}$ |
| BCN [95] | $\lvert\Gamma(u) \cap \Gamma(v)\rvert \log(s) + \sum_{w\in\Gamma(u)\cap\Gamma(v)} \log(R_w)$, where $s = \frac{\lvert V\rvert(\lvert V\rvert-1)}{2\lvert E\rvert} - 1$,$R_w = \frac{N_{\triangle w}+1}{N_{\wedge w}+1}$ $N_{\triangle u}$ ($N_{\wedge u}$): # of triangles (non-triangles) involving $u$. |
| BAA [95] | $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{\log(deg(w))}(\log(s) + \log(R_w))$ |
| BRA [95] | $\sum_{w\in\Gamma(u)\cap\Gamma(v)} \frac{1}{deg(w)}(\log(s) + \log(R_w))$ |
| Katz [67] | $\sum_{l=1}^{\infty} \beta^l \cdot \lvert paths_{u,v}^{<l>}\rvert$ where $\beta > 0$, $paths_{u,v}^{<l>}$: all $l$-hop paths between $u$ and$v$ |
| LP (Local Path) [165] | $\lvert paths_{u,v}^{<2>}\rvert + \epsilon \cdot \lvert paths_{u,v}^{<3>}\rvert$ |
| PPR (Personalized PageRank) [16] | $\pi_{u,v} + \pi_{v,u}$ where $\pi_{u,v}$: probability of a random walk from $u$ to $v$ with a restart probability $\alpha \in [0, 1]$ |
| LRW (Local Random Walk) [94] | $\frac{deg(u)}{2\lvert E\rvert}\pi_{uv}(m) + \frac{deg(v)}{2\lvert E\rvert}\pi_{vu}(m)$ where $\pi_{uv}(m)$: probability from $u$ to $v$ after $m$ steps |
| SP (Shortest Path) | # of hops on shortest path between $u$ and $v$ |
| PA (Preferential Attachment) [17] | $deg(u) \cdot deg(v)$ |
| Rescal [111] | $XRX^T(u, v) + XRX^T(v, u)$ where adjacent matrix $A \approx XRX^T$ $X$: a $\lvert V\rvert \times r$ matrix, $R$: a $r \times r$ matrix |

**Table 3.3:** The 14 metric-based algorithms used for our study. Notations: given graph $G =< V, E >$, $u$ and $v$ are two graph nodes, $\Gamma(u)$ denotes the neighbors of node $u$, $deg(u)$ represents the node degree of $u$.

Thus for Renren and YouTube, we terminate the Katz$_{lr}$ experiments at snapshots of 65M edges and 5.5M edges, respectively.

In terms of computation cost, the local metrics (CN, JC, AA, RA, BAA, BCN, BRA) are easy to compute since we only need to compute each node's 2-hop neighbors. PA is also fast because one can optimize the implementation to only consider top-K node pairs. Even for our largest Renren graph, the computation for the above eight metrics finishes within a few minutes (we run the C++ implementation on 10 standard servers, each with 8 cores and 192GB RAM). The next three metrics (LRW, PPR and LP) take a few hours to compute because LRW and PPR require random walk computation while LP requires reaching 3-hop neighbors. Finally, the most complex metrics (Rescal, Katz and SP) take a few days to complete since they require node embedding. We also note that for the classifier-based methods, the computation complexity is dominated by feature calculation, *i.e.* computing the above similarity metrics.

## 3.4 Metric-Based Prediction

Our empirical evaluation begins with metric-based prediction algorithms. We seek to understand their prediction accuracy, and the key factors that lead to prediction errors.

### 3.4.1 Experimental Setup

Given a sequence of snapshots $\{G_1, G_2, ..., G_T\}$, we predict the new edges to appear in $G_t$ based on observed $G_{t-1}$. For each of the 14 metric-based algorithms, we compute the similarity metric score for each unconnected node pair, and select the top $k$ node pairs with the highest score. While the choice of $k$ may affect prediction accuracy, we use the ground truth value, *i.e.* $k$ equals the number of new edges among $V_{t-1}$ nodes appeared in $G_t$ but not in $G_{t-1}$. This allows us to focus on the effectiveness of similarity metrics. As a baseline for comparison, we also implement a *random prediction* algorithm, which uniform-randomly picks $k$ unconnected node pairs from $V_{t-1}$ as the predicted new edges in $G_t$.

**Performance Metrics.** We follow the established practice of evaluating each link prediction algorithm by comparing results to those from random prediction, *i.e.* in terms of the factor improvement over random [89]. Specifically, given a similarity metric $M$, let $E_t^M$ represent the set of correctly predicted node pairs that become connected in $G_t$, *i.e.* the overlap between the predicted top $k$ node pairs to connect and those that actually connect in $G_t$. Let $E_t^R$ be the set of correctly predicted edges using random prediction, with an expected size of $\overline{|E_t^R|}$. Thus the performance metric is the improvement factor or *accuracy ratio* [89]:

$$|E_t^M| / \overline{|E_t^R|} \tag{3.1}$$

If the ratio is larger than 1, prediction using metric $M$ is more accurate than random prediction (by predicting $k$ edges). Note that we choose to use *accuracy ratio* rather than the area under the receiver operating characteristic curve (AUC) because AUC evaluates link prediction performance according to the entire list of the predicted node pairs [97], while our goal is to evaluate the accuracy of top $k$ predicted node pairs. This allows us to focus on examining the effectiveness of similarity metrics.

### 3.4.2 Metric-based Prediction Accuracy

**Absolute Prediction Accuracy.** We start by first looking at the raw prediction accuracy results in absolute terms, *i.e.* ratio of correctly predicted edges that match real new edges. For each consecutive pair of snapshots $G_{t-1}$ and $G_t$, we apply each prediction algorithm on $G_{t-1}$ generate the next $k$ links likely to form, and compute the overlap in the result with the $k$ links actually formed in $G_t$:

$$|E_t^M|/k \tag{3.2}$$

Prediction accuracy was quite low across the board, for all algorithms on all snapshots across all of our datasets. To highlight these accuracy results, we show in Table 3.4 the highest absolute accuracy results obtained by each algorithm over any snapshot pair across our datasets.

| Network | JC | BCN | BAA | BRA | LP | LRW | PPR | SP | Katz$_{lr}$ | Katz$_{sc}$ | Rescal | PA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Renren | 1.72 | 2.40 | 3.22 | 3.52 | 1.75 | 1.06 | 2.44 | 0.053 | 0.82 | 0.018 | 0.091 | 0.0068 |
| Facebook | 1.21 | 6.17 | 6.82 | 4.43 | 5.53 | 2.11 | 1.06 | 0.10 | 9.41 | 1.85 | 4.45 | 0.21 |
| YouTube | 0.22 | 0.59 | 0.53 | 0.44 | 0.60 | 0.58 | 0.23 | 0.0021 | 0.98 | 1.44 | 1.75 | 0.38 |

**Table 3.4:** Best possible absolute accuracy (%) of all prediction methods on each dataset.

It is clear to see that in absolute terms, link prediction performs poorly in practice. While some methods consistently do better than others, the best they can do is accuracy in the single digits in percentages, *e.g.* 5–6%. The best results tend to come from the Facebook dataset, likely because it's significantly smaller (33 times fewer nodes) than the Youtube and Renren datasets. The single best result is Katz$_{lr}$, which reaches 9.41% on Facebook, but fails to reach even 1% on the larger datasets. Note that our definition of "accuracy" is loose, in that it only requires a predicted link to appear within some range of $k$ new links (see Table 3.2), where $k$ represents all links created in a time period ranging from one week (YouTube) to four weeks (Renren).

These numbers are likely to be significantly lower for real networks, which contain orders of magnitude more nodes (and therefore many orders of magnitude more potential new links) than our datasets, *e.g.* Facebook, WhatsApp, Pinterest etc. While our results are limited by reliance on only network structure (existing links), these results highlight the fact that link prediction is far from a solved problem. These results explain why link prediction literature typically uses the *accuracy ratio* [89], which compares results to a purely random algorithm. We will use the accuracy ratio metric for the rest of our analysis.

**Accuracy Ratio Results.** We present prediction results of our 14 metric-based algorithms from Figure 3.5 to Figure 3.7, as the accuracy ratio over the sequence of snapshots for each OSN (marked by their total edge count). We omit the results of CN, AA and RA because they perform similarly (slightly worse) than their Local Naive Bayes versions, *i.e.* BCN, BAA and BRA. We include two implementations of Katz: $\text{Katz}_{lr}$ and $\text{Katz}_{sc}$, where $\text{Katz}_{lr}$ almost consistently outperforms $\text{Katz}_{sc}$, but is difficult to scale on Renren and Youtube. For the rest of the paper we only show analysis of $\text{Katz}_{lr}$ and refer to it as *Katz*.

We make two key observations from Figure 3.5 to Figure 3.7. *First*, as expected, all metric-based algorithms outperform random prediction over each entire sequence of snapshots. The largest improvement on accuracy ratio is more than 100,000 times for Renren and YouTube, and 6000 for Facebook. A major contributor to this magnitude of differential is the large network sizes, where the accuracy of random prediction quickly decreases as network size grows, resulting in a much higher accuracy ratio.

*Second*, while the best algorithm varies across the three networks, there are algorithms, *i.e.*,SP and PA, which consistently perform poorly. SP gives all 2-

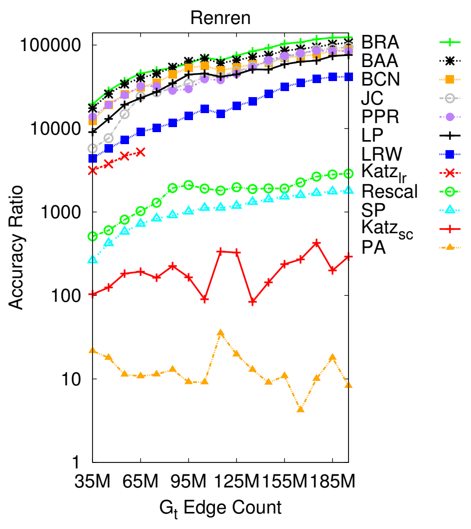


**Figure 3.5:** Link prediction performance for **Renren** dataset.

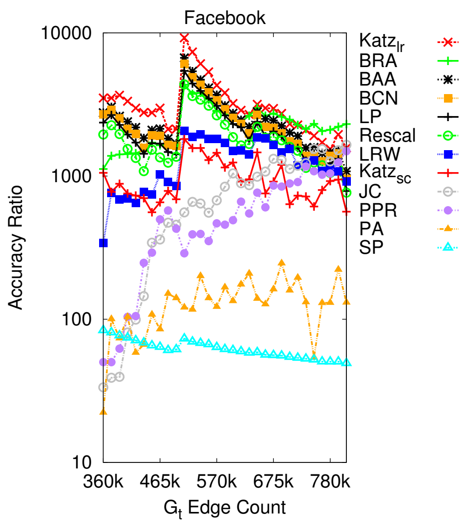


**Figure 3.6:** Link prediction performance for **Facebook** dataset.

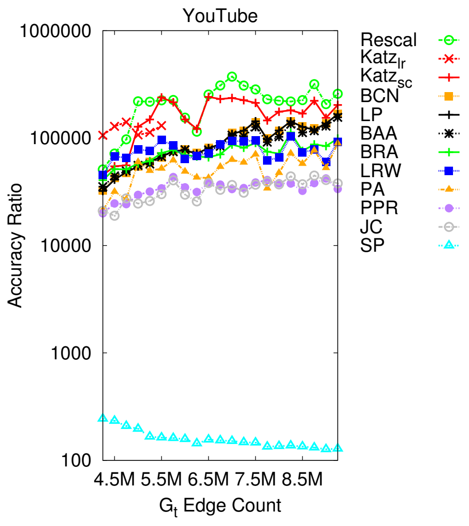


**Figure 3.7:** Link prediction performance for **YouTube** dataset. From Figure 3.5 to Figure 3.7 we show accuracy ratio of all metric-based prediction algorithms. We omit the results of CN, AA and RA because they perform similarly (slightly worse) than their Local Naive Bayes versions, *i.e.* BCN, BAA and BRA. The results for Katz$_{lr}$ in Renren and YouTube are capped to 65M and 5.5M edges due to computation complexity.

hop node pairs the highest score, thus its prediction is actually random choice over all such pairs. PA tries to capture "the rich get richer" property, which is not dominant in friendship creation networks (*i.e.*, Renren and Facebook), where joint efforts from both users are required [161]. PA achieves marginally better accuracy ratio in YouTube, which is more of a subscription network where popular users attract more followers.

**Impact of Network Structures.** As mentioned before, Renren and Facebook are more similar in underlying structures since they are both traditional social networks. Our results from Figure 3.5 to Figure 3.7 align with this observation that top algorithms are similar on Renren and Facebook, *i.e.*, both include common neighbor based algorithms BRA, BAA and BCN. Renren is slightly different from Facebook in that it is a non-sampled graph, and therefore captures higher connectivity between nodes compared to the subsampled regional network in Facebook. Thus Katz is hard to scale on Renren and JC and PPR perform much better. JC and PPR prefer pairs with both low degree nodes, which are usually inactive (more in Section 3.4.4) and are most common in the early phase of our Facebook trace, and decrease as the Facebook network grows over time. We can see their clearly increasing accuracy ratio in Figure 3.6.

In contrast, YouTube is more of a subscription network, where many super nodes with extremely high degrees remain super active in link creation. Thus

YouTube has much higher node heterogeneity and lower network assortativity. We find that more than 40% new edges involve the top 0.1% nodes with highest degrees in YouTube, while only less than 3% for Facebook and Renren. Also, among edges created by super nodes, most are low degree nodes (80% with degree $< 20$). We confirm this by measuring the assortativity for each network. It stays consistently negative for YouTube, and generally positive for the other two.

The difference in network structures produces significantly different link prediction results in YouTube. Because they prefer node pairs with both high degrees, BRA, BAA and BCN do not rank highly amongst metrics (note that the y-axis from Figure 3.5 to Figure 3.7 is in logscale). PPR and JC perform very poorly because most nodes have very low degree ($\sim$80% nodes with degree $\leq 3$). The outperformer is Rescal, which achieves extremely good performance. Rescal works by condensing the interaction among nodes into a much smaller latent space, where it models the interaction between latent components instead, and assigns nodes corresponding weights for each latent component. Intuitively, super nodes are critical in many roles thus have much higher weights, leading to a higher final score. Our results also confirm that super nodes are weighted extremely highly while the rest share similar weights. In this way Rescal best captures the negative assortativity in YouTube.

**Correlation of Accuracy with 2-hop Edge Ratio.** We observe that most algorithms increase in accuracy ratio with network growth, but only for Renren and YouTube, not Facebook. Our analysis shows that this could be explained by a dependence on link creation between 2-hop neighbors, i.e. $\lambda_2$, the percentage of 2-hop node pairs in $G_{t-1}$ who form edges in $G_t$. A plot of $\lambda_2$ shows that it increases with network growth in Renren/YouTube, but decreases (after a matching spike) in Facebook. This is explained by the trend towards "densification" over time [81]. This is disrupted in the Facebook trace, because subsampling over the regional network breaks an increasing number of cross-regional edges as the network grows. We compute the average Pearson correlation of the top-performing 6 metrics for each graph to $\lambda_2$. The results are 0.95 for Renren, 0.83 for YouTube and 0.81 for Facebook.

**Summary.** Our results produce two key takeaways. *First*, the underlying network structure heavily impact prediction accuracy of metric-based algorithms (in terms of accuracy ratio). The more similar network structures in Renren and Facebook (links in which are both the abstraction of friendship between users, while YouTube is more of a subscription network) means their prediction results show consistent relative performance. *Second*, prediction accuracy of most metric-based algorithms strongly correlate with the ratio of 2-hop edges in network evolution, because their predictions are dominated by 2-hop edges.

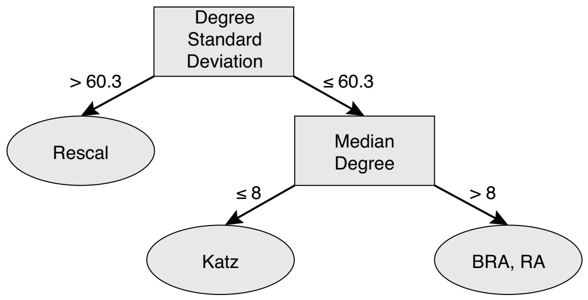


**Figure 3.8:** Visualization of classification results on choosing the best metric-based algorithm.

### 3.4.3 Choosing Metric-based Algorithms

Since network structures heavily impact the performance of metric-based algorithms, a natural question is: "given a network, can one *predict* the best link prediction algorithm?" And similarly: "given an algorithm, can we *characterize* the kind of networks on which it provides the most accurate link prediction?"

We answer the first question by training a multi-class classifier (decision tree), where the input features are the network properties and each class represents a (winning) link prediction algorithm (14 classes in total). We treat each graph snapshot as a data point, and create 69 data points across our three datasets. We consider the following features (computed from each snapshot): node count, edge count, node degree distribution (average, standard deviation, $x$-percentile), clustering coefficient, average path length, and network assortativity.

Figure 3.8 shows the resulting decision tree, where Rescal, Katz and BRA (RA) are among the best performing algorithms (consistent with Figure 3.5 to

Figure 3.7). We see that the heterogeneity of node degrees in the network (captured as degree standard deviation) is the highest impact feature. It specifies that networks with high node degree heterogeneity should use Rescal, which aligns with our analysis in Section 3.4.2 that Rescal prefers node pairs with higher degree heterogeneity. The next factor is the median node degree where lower values ($\leq$8) marks Katz due to its limited scalability and higher values points to BRA (RA) which prefer high-degree node pairs.

Note that this result is not meant as a definitive guide to choosing link prediction approaches for different types of graphs. Our training set for the decision tree is relatively small, and only covers three distinct types of networks. A more "robust" result would require data from a wide range of networks with varying characteristics, with even more snapshots per network. We only use the results here to demonstrate general trends between key features, which are consistent with our detailed experimental results (Figure 3.5 to Figure 3.7).

To answer the second question, we train a binary classifier (decision tree) for each algorithm where the inputs are the same set of network properties. We consider an algorithm to provide "good" prediction (*i.e.* positive) if its prediction accuracy ratio is within 90% of the optimal algorithm. The classification results are shown as below: (we omit algorithms for which there are few or no positive results):

- Rescal: standard deviation of node degree$> 60.3$
- Katz: # of edges$\leq 4.5M$
- BRA (RA): median node degree$> 7$

The results are consistent: Rescal is best for networks with high node degree heterogeneity, Katz is suitable for networks of limited scale and BRA (RA) is best for high-density networks.

**Summary of Observations.** We train classifiers to explore the correlation between the networks and metric-based link prediction algorithms. While we are limited to our three large network traces, we believe our results do provide some insights on today's metric-based link prediction algorithms:

- On sparse and small networks, Katz is a good choice.
- On dense and large networks, BRA (RA) performs well.
- On networks with high node heterogeneity, Rescal is likely the best solution.

### 3.4.4 Sources of Low Prediction Accuracy

While metric-based prediction largely outperforms random prediction, accuracy is still low in absolute terms. For example, the best similarity metric (BRA) on Renren boosts prediction accuracy over random prediction by more than 40,000 times (at 55M edges). Yet it only achieves 3% accuracy when predicting the next

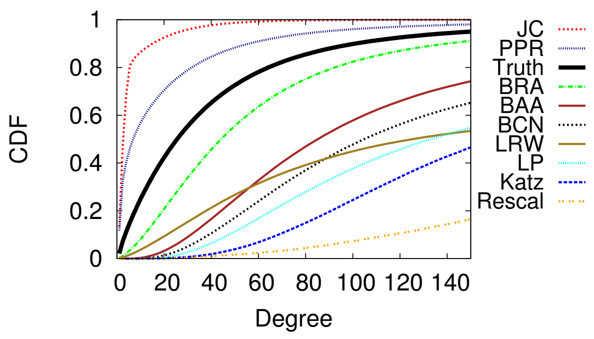


**Figure 3.9:** Degree distribution of nodes in predicted edges (Renren, 55$M$ edges).

edge. To understand the key reasons behind such low accuracy, we investigate both structural and temporal aspects of each metric-based algorithm, with the exceptions of PA and SP, the worst performing metrics which we discussed in Section 3.4.2. We later take our findings into account when designing complementary prediction mechanisms in Section 3.6. Our analysis shows consistency over time and across networks. For brevity we focus our discussion on a sample of results (Renren, 55M edges).

**Structural factors.** We notice that all these similarity metrics are strongly biased by node degree. Figure 3.9 plots the degree distribution of nodes associated with the predicted edges (by each metric) and the ground truth distribution. We see that PPR and JC are heavily biased towards low-degree nodes, while the rest focus more on high-degree nodes. Such bias often comes from the construction of the similarity metric. Take for example BCN (and CN). In a small-world network,

| Metric | Predicted Edges | Real Edges |
|---|---|---|
| Rescal | 99.5% | 0.5% |
| LRW | 66.7% | 0.6% |
| Katz | 39.7% | 0.6% |
| LP | 33.3% | 0.5% |
| BCN | 24.2% | 0.5% |
| BAA | 16.4% | 0.5% |
| BRA | 4.7% | 0.8% |

**Table 3.5:** Ratio of predicted and actual created edges that involve 0.1% most frequently predicted nodes (Renren snapshot with $55M$ edges).

two nodes with high degree likely share more common neighbors, and are more likely to be chosen by the common neighbor algorithm.

We also observe that for metrics biased towards high-degree nodes, their results are dominated by a small number of nodes. To illustrate this, we find the 0.1% nodes most frequently predicted to create a new edge, and show their ratio of predicted and real edges in Table 3.5. We see that except for BRA, all other similarity metrics overpredict the involvement of a small group of nodes in edge creation. It makes sense that the worst offender, Rescal, is much better suited for a supernode-driven network like YouTube. There, its frequent link predic-

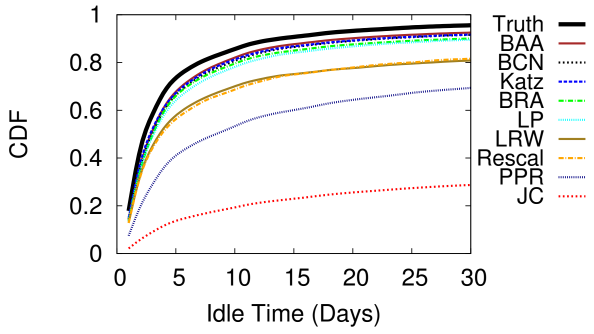


**Figure 3.10:** CDF of node idle time in predicted edges (Renren, $55M$ edges).

tions around supernodes matches the network structure and produces much more accurate results.

**Temporal factors.** Our analysis also shows that these metrics tend to predict links between less active nodes. In particular, for each snapshot $G_t$, we measure the *idle time* for a node $v$ in $G_t$ as the time gap between $t$ and the most recent time when $v$ creates an edge. Figure 3.10 shows that the idle time of nodes in predicted edges by all metrics are larger than that of ground truth, meaning that they are all biased to nodes that are dormant recently, which are less likely to form new edges.

## 3.5 Classification-based Prediction

Classification-based algorithms apply supervised learning to predict links using multiple similarity metrics as features. The key challenge is how to scale to large

OSNs, *i.e.* being able to predict edges among all possible node pairs. Prior works limit the prediction coverage to a very small subset of node pairs [130, 137]. Another challenge is that social networks are highly sparse, translating into highly "imbalanced" positive (connected) and negative (disconnected) subsets. Prior work cites data imbalance as a major cause of low prediction accuracy [57]. In our 55M-edge Renren snapshot for example, the ratio of positive to negative links is $1 : 179K$, and decreases further as the network grows.

In this section, we evaluate classification-based link prediction in practical scenarios, using our large OSN datasets with high data imbalance. To do so, we develop a scalable measurement mechanism for implementing and evaluating classification-based algorithms. We also study how they perform on imbalanced data and compare their results to metric-based prediction algorithms.

## 3.5.1 Evaluation Configuration

Classification-based algorithms first train models (classifiers) using labeled data and their corresponding features, then apply the trained classifiers to test data to predict their labels. Link prediction only requires binary classification ("+" for creating an edge and "-" for no edge).

The key challenge in evaluating these algorithms is how to train and make prediction on all possible node pairs – this requires computing all the features for

$O(|V^2| - |E|)$ node pairs and making a classification decision ( $|V|$ and $|E|$ the graph node and edge count). Even for a “small” Renren snapshot (2.3M nodes, 25M edges), it takes 88 days to compute features!

**Snowball Sampling.** To address this challenge, we consider limiting our evaluation using snowball sampling [49], which has been shown to effectively reduce computation cost while preserving network structure and statistical representativity.

Specifically, for a snapshot $G_{t-2} = \{V_{t-2}, E_{t-2}\}$ we first randomly select a node $v$ as the seed, then run a breadth-first-search from node $v$ until a fixed percentage $p$ of nodes are visited. These visited nodes $V^S_{t-2}$ are the sampled nodes in snapshot $G_{t-2}$. We repeat the process on the next snapshot $G_{t-1} = \{V_{t-1}, E_{t-1}\}$ using the same seed $v$, producing $V^S_{t-1}$. The choice of sampling percentage $p$ must balance between computation cost and data representativity. We configure $p$ based on the network size. Since our Facebook network is reasonably small, $p$=100%. For Renren and YouTube, $p$=2%.

Next, we apply common classification methods on these sampled node sets. During the training process, we measure the similarity features of all node pairs among $V^S_{t-2}$ in $G_{t-2}$, labeling each node pair as either positive or negative depending on whether they are connected in $G_{t-1}$, and training a classifier using this labeled set.

| Graph | | $G_{t-2}$ | | $G_{t-1}$ | | Snowball |
|---|---|---|---|---|---|---|
| | | Nodes | Edges | Nodes | Edges | Sampling $p$ |
| Facebook | small | 49K | 345K | 49K | 360K | 100% |
| | large | 56K | 600K | 57K | 615K | |
| YouTube | small | 1.63M | 4M | 1.74M | 4.25M | 2% |
| | large | 2.63M | 7M | 2.70M | 7.25M | |
| Renren | small | 2.3M | 25M | 2.7M | 30M | 2% |
| | large | 6.2M | 95M | 6.7M | 105M | |

**Table 3.6:** Data instances for evaluating classification algorithms.

In the testing process, we collect features between node pairs among $V_{t-1}^S$ in $G_{t-1}$, feed them into the trained classifier to compute prediction scores, and then choose the top $k$ node pairs with the highest scores as the new edges for the next snapshot $G_t = \{V_t, E_t\}$. As in Section 3.4, we set $k$ to the actual number of new edges created among node pairs in $V_{t-1}^S$ for $G_t$, and use the accuracy ratio to evaluate prediction accuracy. To minimize the impact of seeds, we randomly select 5 nodes as seeds, repeat classification methods on them, and measure the average and standard deviation of prediction accuracy ratios.

Given the computation complexity, we limit our evaluation to two instances (listed in Table 3.6) of different sizes (small and large) for all three networks. Again, because these instances produce highly consistent results and space constraints, we only discuss the results for the large networks.

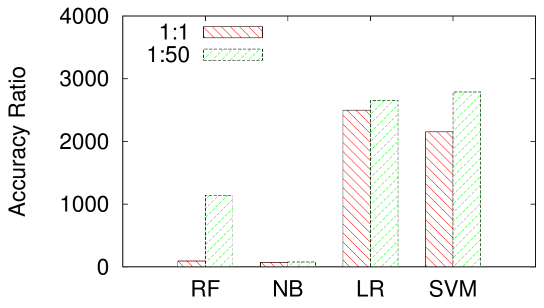


**Figure 3.11:** Accuracy ratio of four classifiers with undersampling ratio $\theta$ 1:1 and 1:50 (Facebook, 345K edges).

**Features and Classifiers.** We use scores from all 14 similarity metrics listed in Table 3.3 as features, and experiment with 4 well-known classifiers: Support Vector Machine (SVM), Logistic Regression, Naive Bayesian (NB), and Random Forests (RF)[4]. We also considered but ultimately rejected Decision Trees, because they can only produce binary recommendations, and are effectively subsumed by Random Forests.

We ran experiments on a wide range of network snapshots, and found the classifiers were consistent in their relative performance. RF and NB always performed poorly, and LR performed generally on par with SVM. In addition, we found that SVM outperformed LR with imbalanced training sets, which has been also shown in prior work [57]. Since our data is highly imbalanced, SVM's results are uniformly the best of the bunch, and we use SVM results for the remainder of our

[4]We use the implementation in an open source library [116] with default parameters for all classifiers in this paper.

discussion. As an example of the relative accuracy results, we plot in Figure 3.11 prediction accuracy ratio for all 4 classifiers on a Facebook network snapshot of 345K edges.

### 3.5.2 Link Prediction Accuracy

To evaluate classification-based prediction, we must first understand the impact of data imbalance within training sets. Recall that link formations in social networks are extremely imbalanced, *i.e.* far fewer connected node pairs than disconnected. This imbalance has been shown to contribute to classification errors [57].

To deal with challenges from data imbalance, we apply the well-known *under-sampling* technique to build training data: keeping all positive node pairs while varying the number of negative node pairs [57]. Here positive(negative) node pairs refer to those which will(not) connect in the prediction timeframe. Figure 3.12 plots prediction accuracy ratio while varying the under-sampling ratio, $\theta$=(# of positive node pairs : # of negative node pairs), from (1:1) to (1:10000)[5]. For our three OSNs, the true (unsampled) positive vs. negative ratio is around (1:100000). Note that existing classification-based prediction algorithms generally use balanced node pairs, or a ratio of (1:1).

[5]We stop at (1:10000) for YouTube and Facebook and (1:5000) for Renren because this is the largest training size we can support on our memory-heavy servers (192GB RAM each).

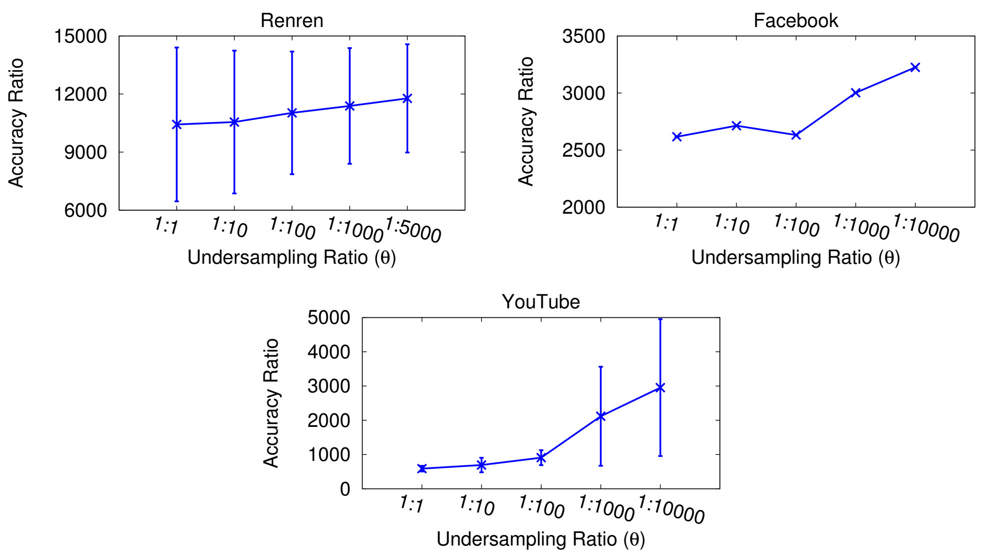


**Figure 3.12:** Performance of classification-based prediction as a function of the under-sampling ratio $\theta$ used during classifier training.

These results show that classification-based prediction algorithms are significantly better than random prediction for all 5 sampling ratios. For Renren and Facebook, accuracy ratio improves as the sampling ratio $\theta$ approaches the actual positive vs. negative ratio (1:100000). Compared to conventional balanced sampling (1:1), a lower under-sampling ratio produces significantly more accurate results, and improvements in accuracy ratio, and also the accuracy, can be as high as a factor of 5.

The above results confirm the effectiveness of classification-based link prediction. More importantly, we show that the performance of these algorithms depends on the configuration of training data. Conventional methods of using

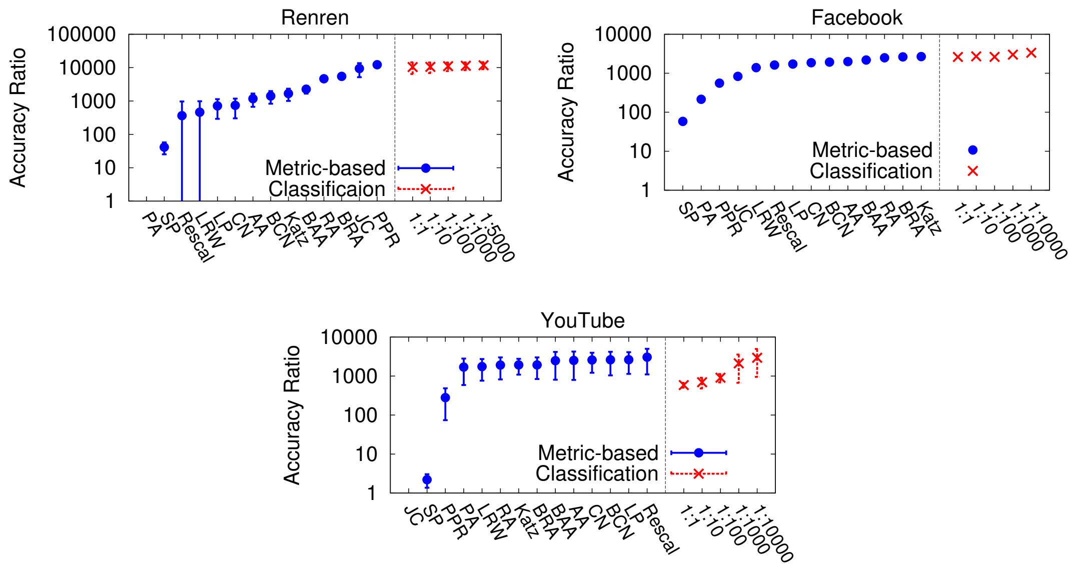


**Figure 3.13:** Comparing the prediction performance of metric- and classification-based prediction algorithms.

balanced training data can lower prediction accuracy by as much as a factor of 5. To minimize such loss, we need to invest efforts on finding the right level of undersampling ratio ($\theta$).

### 3.5.3 Comparing to Metric-based Algorithms

For a fair comparison, we run the metric-based methods again on the same sampled data ($V_{t-1}^S$). We plot the accuracy ratio for each algorithm (blue circle on the left) in Figure 3.13, and rank them in descending order from right to left. We see that the top (most accurate) similarity metrics are generally consistent on both the sampled data and the entire network (see Section 3.4) across different datasets. Also note that the test dataset $V_{t-1}^{S}$ is smaller than $V_{t-1}$ and better connected, the accuracy ratio of the metric-based algorithms is lower than results previously shown in §3.4 (accuracy ratio is lower because random prediction does better on this smaller dataset).

**Comparing Accuracy.** Figure 3.13 plots the accuracy ratio of SVM (red cross on the right) and metric-based methods (blue circle on the left). With a well-chosen $\theta$, SVM consistently performs as well as, or outperforms the best metric-based algorithms.

This outperformance stems from two factors: combining multiple similarity metrics to broaden coverage, and using under-sampling to address the issue of data imbalance. Overall, these results show that among existing algorithms, the SVM classifier provides consistently strong results. However, we also note that some similarity metrics, namely RA and BRA, provide consistently "good" results across all of our networks. In scenarios where the computational or training costs of SVMs were undesirable, RA and BRA provide reasonable alternatives with much lower computational complexity.

**Similarity Metric Ranking vs. SVM Feature Weight.** We seek to understand whether a good similarity metric in the metric-based method (identified from Figure 3.13) also becomes a dominant feature for the classification method. For this we use the feature coefficient provided by SVM, where a larger abso-

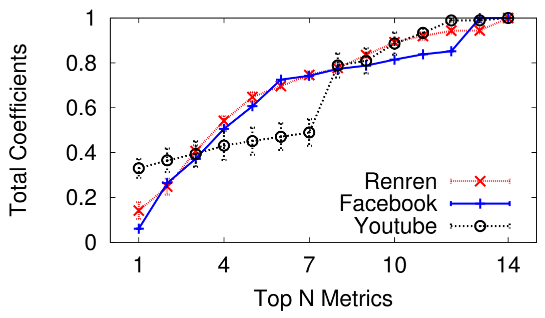


**Figure 3.14:** The relationship between top similarity metrics and top SVM features, shown as the total normalized SVM coefficient of top $N$ similarity metrics, $N$=1,2,...,14.

lute value means the feature is more important. To make a fair comparison, we normalize the coefficients (using absolute values) within each classifier.

We take two steps to study the relationship between top similarity metrics and top SVM features. First, we directly compare the rankings of the two. For both Renren and Facebook, the rankings are very similar between the similarity metrics and SVM features, *i.e.* top similarity metrics are also top features in SVM. For YouTube, the orders are less consistent, except that Rescal always ranks first.

Next, we study how top similarity metrics contribute to SVM by comparing their feature coefficients. Specifically, for each graph we pick the top $N$ similarity metrics and calculate their total normalized SVM coefficients, where $N = 1, 2, ..., 14$. Figure 3.14 presents the results for the large data instance listed in Table 3.6 with the largest $\theta$. Results of small data instances and other values of $\theta$ are consistent and omitted for brevity.

We see that for Renren and Facebook the similarity metrics make similar contributions to the machine learning process. The top 6 similarity metrics have a slightly higher weight than the rest. For YouTube, the top first similarity metric (Rescal) and a lower ranked metric (Katz) are the key contributors while the rest make similar contributions.

Together, these results suggest that in general the metric-based and classifier-based methods share similar preferences on similarity metrics. But the classifier-based methods can combine prediction power of multiple metrics to achieve a higher accuracy and robustness across different datasets. Finally, the difference between Renren/Facebook and YouTube aligns with our earlier observation that as a subscription network YouTube's link prediction pattern differs from those of Renren and Facebook.

## 3.6 Improving Link Prediction

While our results show that today's prediction algorithms significantly outperform random prediction, they are still limited in their prediction accuracy. A fundamental contributing factor is that current prediction algorithms take a purely static approach to network analysis, and do not take in account temporal patterns exhibited by an evolving network. While recent studies seek to extend link prediction to support dynamic networks, they either do not scale [130], or are restricted to single model or metric [144] where performance vary significantly across datasets.

In this section, we improve existing link prediction algorithms by integrating them with dynamic network analysis. Specifically, we identify key patterns on network dynamics, and use them to build *temporal filters* that drastically reduce the search space for link prediction. Our proposed filters effectively augment existing link prediction algorithms, providing a significant boost in prediction accuracy. This is even true for algorithms that were already designed to capture network dynamics, *e.g.* [37].

### 3.6.1 Temporal Properties on Edge Creation

Using our dynamic OSN datasets, we investigate how different properties of network dynamics affect edge creation. These include node activeness, neighborhood structure evolution, neighborhood activeness, and arrival of common neighbor. We conclude that *node activity* and *arrival of common neighbor* are the key factors for all three networks, and thus we omit analysis for other explorations here. We have consistent observations across different snapshots and over different networks, and due to space limitation we only show figures for Renren snapshot at 55M edges in this subsection. We will briefly summarize our observations for other networks in the next subsection.

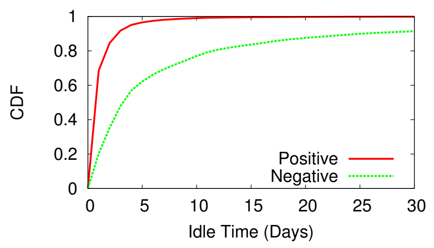


**Figure 3.15:** CDF of active node idle time in a Renren snapshot.

**Node Activeness.** Intuitively, a node that has recently actively created edges is more likely to create edges in the near future. We validate this by measuring node activity on both positive and negative node pairs (*i.e.* those with edges and those without). For each node pair, we mark the node with longer idle time (defined in Section 3.4.4) as the inactive node and the other as the active node. We measure activity by the idle time of the active node, the idle time of the inactive node, and the number of edges created by the active node in the past $d$ days.

We found that for positive node pairs, *i.e.* those who will connect in the prediction timeframe, the idle times of both active and inactive nodes are significantly smaller. Figure 3.15 plots the CDF of the active node's idle time for the Renren snapshot at 55M edges. More than 90% of positive node pairs have <3 days idle time while only 40% of negative pairs do so. This 3-day threshold can effectively distinguish positive and negative node pairs. Similar patterns can be found when comparing the inactive nodes' idle times, with a 20-day idle threshold.

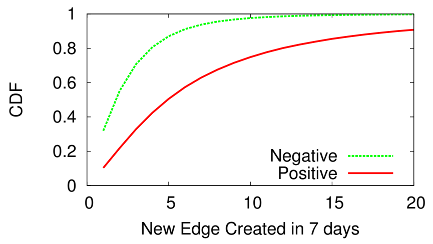


**Figure 3.16:** CDF of new edges created in the past 7 days by a node in a Renren snapshot.

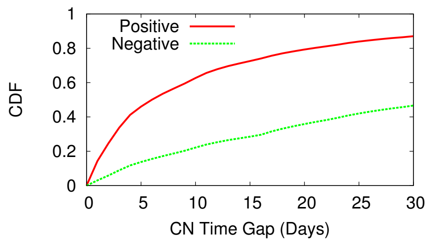


**Figure 3.17:** CDF of CN time gap of positive and negative node pairs in a Renren snapshot.

Furthermore, active nodes in positive node pairs tend to create more edges in a recent time. Using the same Renren snapshot, Figure 3.16 shows the CDF of new edges in the past week for both positive and negative sets. For more than 60% of positive node pairs, the active node creates more than 3 edges while only 20% of negative node pairs do so. This "3-edge in past 7 days" can also be used to help identify potential new links.

**Arrival of Common Neighbor.** We show in Section 3.4 that most similarity metrics focus on predicting edge formation between 2-hop neighbors. For these, the recent arrivals of common neighbors can often trigger the completion of a triad [167] and thus be critical in predicting edges.

We test this hypothesis by measuring, for each node pair, the gap between the most recent time when they connect to a common neighbor and the current snapshot time, referred to as the *CN time gap*. Our results show that the CN time gap of positive set is much smaller than that of negative set. Figure 3.17 shows the result for the same Renren snapshot, where more than 60% of positive pairs create their last common neighbors in the last 10 days, while 20% of negative pairs do so.

### 3.6.2 Temporal Filtering

We propose to use these observations, which are consistent across networks, to develop "temporal filters" to drastically reduce the search space of new links by filtering out node pairs that are unlikely to create edges. Specifically, we remove any potential node pair from the candidate list if it fails to meet any of the following four criteria:

- Idle time of active nodes $< d_{act}$ days.
- Idle time of inactive nodes $< d_{inact}$ days.
- $d$-day new edges $\geq E_{new}$.
- CN time gap $< d_{CN}$. [6]

Our threshold values are listed in Table 3.7, which hold across different snapshots for each corresponding network. While each parameter is network specific, the methodology to discover them is general.

**Prediction Accuracy after Filtering.** We now present the improvement in link prediction accuracy (in terms of accuracy ratio) after adding temporal filter-

[6] For node pairs beyond 2 hops, we do not apply this criterion.

| Graph | Node Idle Time | | d-day New Edges | | $d_{CN}$ |
|---|---|---|---|---|---|
| | $d_{act}$ | $d_{inact}$ | d | $E_{new}$ | |
| Facebook | 15 | 40 | 21 | 2 | 40 |
| YouTube | 3 | 30 | 7 | 3 | 20 |
| Renren | 3 | 20 | 7 | 3 | 10 |

**Table 3.7:** Parameters of the temporal filters.

| Network | JC | BCN | BAA | BRA | LP | LRW | PPR | SP | Katz | Rescal | PA | 1:1 | 1:10 | 1:100 | 1:1000 | 1:10000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Renren | 2.2 | 5.8 | 4.1 | 2.3 | 9.7 | 3.2 | 1.7 | **14.9** | 1.5 | 2.4 | - | 1.9 | 1.9 | 1.8 | 1.8 | 1.8* |
| Facebook | **5.7** | 1.2 | 1.2 | 1.4 | 1.3 | 1.3 | 5.3 | 4.4 | 1.3 | 1.2 | 2.1 | 1.3 | 1.4 | 1.5 | 1.3 | 1.2 |
| YouTube | - | 1.2 | 1.2 | 1.2 | 1.2 | 1.1 | 3.1 | **15.7** | 1.5 | 1.1 | 1.2 | 2.2 | 2.2 | 2 | 1.2 | 1.1 |

**Table 3.8:** Ratio of accuracy values after filtering vs. before filtering for all metric-based and classification methods. Bold value in each row is the maximum improvement for that network; "-" means the accuracy before filtering is "0". *Ratio in Renren is 1:5000.

ing. We experiment with the same data instances used to evaluate classification-based algorithms (see Table 3.6) and present the result for the large instance. Results from the smaller instance show even more significant benefits and are omitted for brevity.

Table 3.8 lists the normalized improvement from applying the filter, *i.e.* the accuracy ratio of prediction with filtering divided by the accuracy ratio of prediction without filters. The improvement is quite significant for many cases, and somewhat incremental for others. For classification-based algorithms, our filtering raises the accuracy by 10%$\sim$120%. For metric-based algorithms, the gain can be as much as a factor of 15.7.

We observe that filtering affects certain algorithms more than others. For metric-based algorithms, applying temporal filters changes the "best" prediction algorithm. For example in Facebook, JC was the weakest metric before the filters, but becomes the best metric after filtering. This is because temporal filters effectively identify and remove the unlikely-to-connect node pairs, *i.e.* inactive, low-degree nodes that JC is unable to identify.

### 3.6.3 Comparing to Other Temporal Methods

Recent works have exploited temporal information to improve prediction accuracy [27, 37, 144]. We compare our filtering design with the time series based prediction [37], a popular method that can also scale to our network datasets. For each potential node pair, this method computes its similarity metrics at multiple past time points, and aggregates these scores to produce a final score of the pair. We implement two aggregation approaches, Moving Average (MA) and Linear Regression (LR), shown by [37] as the two best approaches, and perform aggregation on equally spaced past time points (the space equals to the number of days between $G_t$ and $G_{t-1}$). We observe that MA consistently outperforms LR, and thus omit the LR result for brevity.

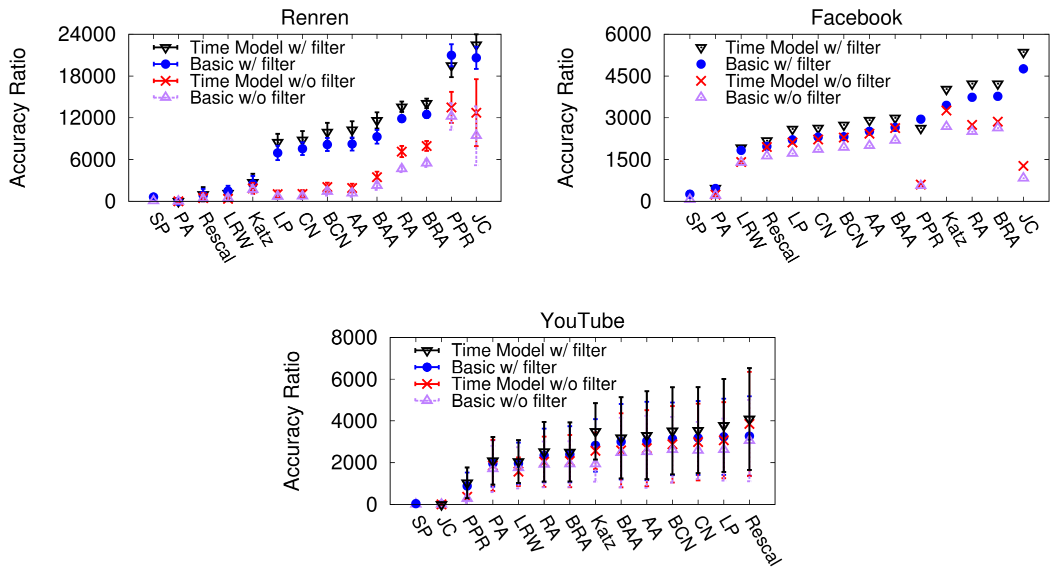


**Figure 3.18:** Our proposed temporal filtering method outperforms time-based models.

Similar to Section 3.6.2, we also present the result for the large instance for each network in Table 3.6 and ignore the similar results from the smaller instance. Figure 3.18 shows prediction accuracy ratio for original similarity metrics (marked as Basic) and those enhanced with MA (marked as Time Model), both with and without our filtering method. For each metric, our filtering consistently improves the accuracy far more than the time series based prediction, especially for Renren and Facebook. Furthermore, even after applying the time series based prediction, our filter can still consistently improve prediction accuracy.

**Summary.** We show that by leveraging temporal information on network dynamics, we can effectively improve link prediction accuracy. Using these temporal filters, we can prune the set of candidate node pairs for edge creation, allowing link prediction algorithms to focus on regions of likely growth. By comparing to other temporal methods, we further confirm the effectiveness and generality of our filtering method.

## 3.7 Related Work

We perform an in-depth study on two types of link prediction algorithms (metric- and classification-based). Prior works have evaluated metric-based algorithms using co-author networks [89], classification-based methods using balanced data [11, 45], and both methods using a small subset of Twitter (155K nodes) [23]. Our work differs by studying both methods using datasets of large, dynamic online social networks that recently became available. By discretizing these datasets into numerous temporal snapshots, we study the evolution of link prediction over fine time intervals, identify potential factors behind prediction errors, and propose filters that improve prediction power for all algorithms.

Recent studies have leveraged temporal information for link prediction. The key approach is to extend existing algorithms in the temporal domain, *e.g.* adding a temporal dimension [6], assigning more weights to new links [135, 144], integrating graph structure information over time [37]. Another approach applies past observations for prediction [27, 130], by identifying subgraphs that are similar to the target subgraph and use their time-evolving behaviors to help predict the target. Unfortunately, each of these methods suffers from at least one of the following limitations:

- The proposed approach is of high complexity and cannot scale to large networks
- The proposed approach is limited to a single model/metric, whose performance varies significantly across networks
- The proposed approach does not capture (and leverage) temporal patterns of the network

In contrast, our approach not only provides a general and scalable link prediction solution that supports a wide variety of similarity metrics, classifiers and network graphs, but also utilizes insights of network evolution to boost prediction accuracy and reduce complexity.

Finally, our work targets link prediction that only require graph topology information, *i.e.* nodes and edges. Additional information, such as edge weights [96], node connections on other social networks [106], and link direction [158], can improve prediction performance. We plan to consider these factors in future work.

## 3.8 Summary and Discussion

Using real traces of large dynamic networks, our work in this chapter takes a concrete step towards objectively quantifying the predictive power of today's link prediction algorithms. By implementing a wide range of algorithms, we have already identified concrete challenges and issues with multiple algorithms, from high computational complexity that limits scalability, to binary classification results that lack granularity.

For metric-based approaches, we have shown the futility of some metrics (shortest path) and validated the scalability of others, *e.g.* scalable Katz heuristics [137]. At the same time, we have shown that it is indeed possible to scale some classifiers to large, multi-million node networks, and that classifiers such as SVMs can produce consistently strong results.

More surprisingly, we find that the best metric-based predictors (vary across different networks) perform on par with the most accurate classifier (SVM in all cases), and we derive potential guidelines for choosing metrics based on network structure. We also take a deeper look at current link prediction algorithms for the source of low accuracy, in terms of both structural and temporal aspects. Furthermore, we provide "temporal filters" that can greatly improve prediction accuracy (across different methods and networks) by leveraging knowledge of prior network dynamics, even for predictors that have already integrated temporal information.

Finally, our results underscore the fact that current prediction algorithms still perform poorly at the fine granularity of individual link predictions, even with our proposed temporal filters. While this confirms link prediction is still an unsolved problem, it is important to calibrate expectations depending on specific applications. For example, while current algorithms focus primarily on predicting nearby neighbors, a significant number of new links connect "distant" nodes. Overcoming these empirical limitations requires a better understanding of underlying network structures and dynamics.

This work only scratches the surface of a much larger problem space. Using datasets from just three networks, we already observe significant variance in accuracy for single metrics across different networks. Much more experimental and analytical work is necessary before we can identify specific properties of each network that make them more or less predictable by certain metrics. Our evaluation is limited by our reliance on network structural data, whereas deployed link prediction systems are likely to combine multiple information sources [53], *e.g.*, user profiles and behavioral data, which can boost prediction accuracy empirically.

# Chapter 4

# Secure Graph Sharing System

## 4.1 Introduction

[1] Graphs are capable to capture many of today's most sensitive datasets, including maps of autonomous systems in the Internet, social networks representing billions of friendships, or connected records of patent citations. Privacy concerns rise as controlling access to these datasets is a difficult challenge. More specifically, it is often the case that owners of large graph datasets would like to share access to them to a fixed set of entities without the data leaking into the public domain. For example, an ISP may be required to share detailed network topology

[1] ©ACM, (2015). This is the authors version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. Abbreviated version of content in this chapter can be found in "Towards Graph Watermarks" [162], Proceedings of the 2015 ACM on Conference on Online Social Networks (IMC'15), http://doi.acm.org/10.1145/2817946.2817956.

graphs with a third party networking equipment vendor, with a strict agreement that access to these sensitive graphs must be limited to authorized personnel only. Similarly, a large social network like Facebook or LinkedIn may choose to share portions of its social graph data with trusted academic collaborators, but clearly want to prevent their leakage into the broader research community.

One option is to focus on building strong access control mechanisms to prevent data leakage beyond authorized parties. Yet in most scenarios, including both examples above, data owners cannot restrict physical access to the data, and have limited control once the data is shared with the trusted collaborator. It is also the case that no matter how well access control systems are designed, they are never foolproof, and often fall prey to attacks on the human element, *i.e.* social engineering. Another option is to modify portions of the data to reduce the impact of potential data leakages. This has the downside of making the data inherently noisy and inaccurate, and still can be overcome by data reconstruction or de-anonymization attacks using external input [107]. Finally, these schemes are hard to justify, in part because it is very difficult to quantify the level of protection they provide.

In this chapter, we propose a new alternative in the form of *graph watermarks*. Intuitively, watermarks are small, often imperceptible changes to data that are difficult to remove, and serve to associate some metadata to a particular dataset.

They are used successfully today to limit data piracy by music vendors such as Apple and Walmart, who embed a user's personal information into a music file at the time of purchase/download [13]. Should the purchased music be leaked onto music sharing networks, it is easy for Apple to track down the user who was responsible for the leak. In our context, graph watermarks work in a similar way, by securely identifying a copy of a graph with its "authorized user." Should a shared graph dataset be leaked and discovered later in public domains (on BitTorrent perhaps), the data owner can extract watermark from the leaked copy and use it as proof to seek damages against the collaborator responsible for the leak. While not a panacea, graph watermarks can provide additional level of protection for data owners who want to or must share their data, and perhaps encourage risk-averse data owners to share potentially sensitive graph data, *e.g.* encourage LinkedIn to share social graphs with academic collaborators.

To be effective, a graph watermark system needs to provide several key properties:

- Graph watermarks should be relatively *small* compared to the graph dataset itself. This has two direct consequences: the watermark will be difficult to detect (and remove) by potential attackers, and adding the watermark to the graph has minimal impact on the graph structure and its utility.

- Watermarks should be difficult to forge and should not occur naturally in graphs, ensuring that the presence of a valid watermark can be securely associated with some user, *i.e.* non-repudiation.
- Both the embedding and extraction of watermarks should be efficient, even for extremely large graph datasets with billions of nodes and edges.

Since we also want to design a watermark system that works in any application context involving graphs, we make no assumptions about the presence of metadata. Instead, our system must function for "barebones" graphs, *i.e.* symmetric, unweighted graphs with no node labels or edge weights.

In this chapter, we present initial results of our efforts towards the design of a scalable and robust graph watermark system. Highlights of our work can be organized into the following key contributions.

First, we identify the goals and requirements of a graph watermark system. We also describe an initial design of a graph watermark system that efficiently embeds watermarks into and extracts them out of large graphs. Graph watermarks are uniquely generated based on a user private key, a secure graph key, and the graph they are applied to. We describe constraints on its applicability, and identify examples of graphs where watermarks cannot achieve desirable levels of key properties such as uniqueness.

Second, we provide a strict proof of uniqueness of graph watermarks, showing that it is extremely difficult for attackers to forge watermarks.

Third, we evaluate our watermarks in term of distortion, uniqueness, and efficiency on several large graph datasets.

Fourth, we identify two attack models, describe additional features to boost robustness, and evaluate them under realistic conditions.

To the best of our knowledge, our work is the first practical proposal for applying watermarks to graph data. We believe graph watermarks are a useful tool suitable for a wide range of applications from tracking data leaks to data authentication. Our work identifies the problem and defines an initial groundwork, setting the stage for follow-up work to improve robustness against a range of stronger attacks.

## 4.2 Background and Related Work

In this section, we provide background and related work on graph privacy and watermark techniques in applications.

**Graph Privacy.** Graph privacy is a significant problem that has been magnified by the arrival of large graphs containing sensitive data, *e.g.* online social graphs or mobile call graphs. Recent studies [14, 107] show that deanonymization attacks can defeat most common anonymization techniques.

A variety of solutions have been proposed, ranging from anonymization tools that defend against specific structural attacks, or more attack-agnostic defenses. To protect node- or edge-privacy against specific, known attacks, techniques utilize variants of *k-anonymization* to produce structural redundancy at the granularity of subgraphs, neighborhoods or single nodes [91, 164, 56, 168]. Alternatively, randomization provides privacy protection by randomly adding, deleting, or switching edges [54, 160]. Others partition the nodes and then describe the graph at the level of partitions to avoid structural re-identification [55]. Finally, a different approach is taken by producing model-driven synthetic graphs that replicate key structural properties of the original graphs [128]. One extension of this work utilizes differential privacy techniques to provide a tunable accuracy vs. privacy tradeoff [129].

Our goals are quite different from prior work on graph anonymization, meant to protect data before its public release. We are concerned with scenarios where graph data is shared between its owner and groups of trusted collaborators, *e.g.* third party network vendors analyzing an ISP's network topology, or Facebook sharing a graph with a group of academic researchers. The ideal goal in these scenarios is to ensure the shared data does not leak into the wild. Once data is shared with collaborators, reliable tools that can track leaked data back to its source serve as an excellent deterrent. Watermarking techniques have addressed similar problems in other contexts, and we briefly describe them here.

**Background on Digital Watermarks.** Watermarking is the process of embedding specialized metadata into multimedia content [74]. The embedded *watermark* is later extracted from the file and used to identify the source or owner of the content. These systems include an embedding component and an extraction component. The embedding component takes three inputs: a watermark, the original data, and a key, aiming to embed the watermark with minimum impact on the data. The key is used as a parameter to generate a unique watermark for a specific user, and is kept confidential by the data owner. Extraction takes as input the watermarked data, the key, and possibly a copy of the original data. Extraction can directly produce the embedded watermark or a confidence measure of whether it is present.

Watermarking is widely used today to protect intellectual property. Significant work has been done in digital watermarking, particularly image watermarking [138, 98, 19, 125, 156]. Watermark techniques [112, 113] have been studied to protect the abuse of digital vector maps. Watermarks have also been used to protect software copyrights [166, 33], by adding spurious execution paths in the code that would not be triggered by normal inputs [146]. Moreover, watermark

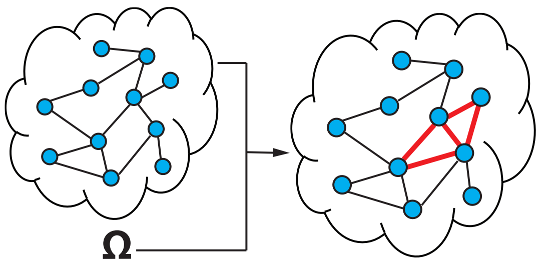


**Figure 4.1:** Embedding graph watermarks. Ω is a secret random generator seed produced using the secure graph key and user's private key.

algorithms have been proposed for relational datasets [8, 88, 63]. Much of this has focused on modifying numeric attributes of relations, using the primary key attribute as an indicator of watermark locations, assuming that the primary key attribute does not change. Finally, watermarks, in the form of minute changes, have been applied to protect circuit designs in the semiconductor industry [117, 154].

## 4.3 Goals and Attack Models

To set the context for the design of our graph watermark system, we need to first clearly define the attack models we target, and use them to guide our design goals.

**Graph watermarks at a glance.** At a high level, we envision the graph watermark process to be simple and lightweight, as pictured in Figure 4.1 and Figure 4.2. Embedding a watermark involves *overlaying* the original graph dataset

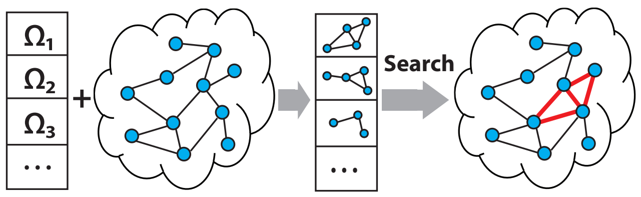


**Figure 4.2:** Extracting graph watermarks. $\Omega_i$ is a secret random generator seed produced using the secure graph key and the private key of user $i$.

$(G)$ with a small subgraph $(W)$ generated using the original graph and a secret random generator seed $(\Omega)$. Embedding the watermark simply means adding or deleting edges between existing nodes in the original graph $G$, based on the watermark subgraph $W$. Each authorized user $i$ receives only a watermarked graph customized for her, generated using a random seed $\Omega_i$ securely associated with her. The seed is generated through cooperation of her private key and a key securely associated with the original graph.

If and when the owner detects a leaked version of the dataset, the owner takes the leaked graph, and "extracts the watermark," by iteratively producing all known watermark subgraphs $W_i$ associated with $G$ and each of the seeds $\Omega_i$ associated with an authorized user. The "extraction" process is actually a matching process where the data owner can conclusively identify the source of the leaked data, by locating the matching $W_i$ in the leaked graph.

In our model of potential attackers and threats, we assume that attackers have access to the watermarked graph, but not the original $G$. Clearly, if an attacker is able to obtain the unaltered $G$, then watermarks are no longer necessary.

**Attack Models.**

The attackers' goal is to destroy or remove graph watermarks while preserving the original graph. Watermarks are designed to protect the overall integrity of the graph data. Thus we do not consider scenarios where the attackers sample the graph or distort it significantly to remove the watermark. Under these constraints, we consider two practical attack models below.

- *Single Attacker Model.* For a single attacker with access to one watermarked graph, it will be extremely difficult to detect the embedded watermark. Without the key associated with another user, forging a watermark is also impractical. Instead, their best attack is to disrupt any potential watermarks by adding or deleting nodes or edges.
- *Collusion Attack Model.* If multiple attackers join their efforts, they can recover the original graph by comparing multiple watermarked graphs, identifying the differences (*i.e.* watermarks), and removing them.

**Design Goals.** The attack models help us define the key characteristics required for a graph watermarking system.

- *Low distortion.* The addition of watermarks should have a small impact on overall structure of the original graph. This preserves the utility of the graph datasets.
- *Robust to modifications.* Watermarks should be robust to modification attacks on watermarked graphs, *i.e.* watermarks should remain detectable and extractable with high probability, even after the graph has been modified by an attacker.
- *Low false positives.* It is extremely unlikely for our system to successfully identify a valid watermark $W_i$ in an unwatermarked graph or a graph watermarked by $W_j$ where $i \neq j$. When we embed a single watermark (Section 4.4), we also refer to this property as *watermark uniqueness*.

Within the constraints defined above, designing a graph watermark system is quite challenging, for several reasons. First, the subgraph that represents the watermark must be relatively "unique," *i.e.* it is highly unlikely to occur naturally, or intentionally through forgery. A second, contrasting goal is that the watermark should not change the underlying graph significantly (low distortion), or be easily detected. Walking the fine line between this and properties of "uniqueness" likely means we have to restrict the set of graphs which can be watermarked, *i.e.* for some graphs, it will be impossible to find a hard to detect watermark that does not occur easily in graphs. Finally, since any leaked graph can have all metadata stripped or modified, watermark embedding and extraction algorithms must function without any labels or identifiers. Note that the problem of subgraph matching is known to be NP-complete [34].

## 4.4 Basic Watermark Design

We now describe the basic design of our graph watermarking system. The basic design seeks to embed and extract watermarks on graphs to achieve watermark uniqueness while minimizing distortion on graph structure. Our design has two key components:

- **Watermark embedding**: The data owner holds a graph key $K^G$ associated with a graph $G$ known only to her. Each user $i$ generates its public-private cryptographic key pair $< K^i_{pub}, K^i_{priv} >$ through a standard public-key algorithm [102], where $K^i_{pub}$ is user $i$'s public key and $K^i_{priv}$ is its corresponding private key. To share the graph $G$ with user $i$, the system combines input from user $i$'s digital signature $K^i_{priv}(T)$ and graph key $K^G$ to form a random generator seed $\Omega_i$, and use $\Omega_i$ to generate a watermark graph $W_i$ for graph $G$. The system embeds $W_i$ into $G$ by selecting and modifying a subgraph of $G$ that contains the same number of nodes as $W_i$. The resulting graph $G^{W_i}$ is given to user $i$ as the watermarked graph.

- **Watermark extraction**: To identify the watermark in $G'$, we use $\Omega_i$ to re-generate $W_i$ and then search for the existence of $W_i$ within $G'$, for each user $i$.

In this section, we focus on describing the detailed procedure of these two components. We present detailed analysis on the two fundamental properties of graph watermarks, *i.e.* uniqueness and detectability in Section 4.5.

### 4.4.1 Watermark Embedding

The most straightforward way to embed a watermark is to directly attach the watermark graph to the original graph. That is, if $W_i$ represents the watermark graph for user $i$, and $G$ represents the original graph, the embedding treats $W_i$ as an independent graph, and adds new edges to connect $W_i$ to $G$. However, this approach has two disadvantages. *First*, direct graph attachment makes it easy for external attackers to identify and remove $W_i$ from $G$ without using graph key $K^G$ and user $i$'s signature $K^i_{priv}(T)$. New edges connecting $W_i$ and $G$ must be carefully chosen to reduce the chance of detection, which is very challenging. *Second*, attaching a (structurally different) subgraph $W_i$ directly to a graph $G$ introduces larger structural distortions.

Instead, we propose an alternative approach that embeds the watermark graph "in-band." That is, the embedding process first selects $k$ nodes ($k$ is the number of nodes in $W_i$) from $G$ and identifies $S$, the corresponding subgraph of $G$ induced by these $k$ nodes. It then modifies $S$ using $W_i$ without affecting any other nodes in $G$. Because the watermark graph $W_i$ is naturally connected with the rest of the graph, both the risk of detection and amount of distortion induced on the original graph $G$ are significantly lower than those of the direct attachment approach.

We now describe the details of "in-band" watermark embedding, which consists of four steps:

1. Generating a random generator seed $\Omega_i$ from user $i$'s signature $K^i_{priv}(T)$ and graph key $K^G$
2. Generating the watermark graph $W_i$ from the seed $\Omega_i$
3. Selecting the placement of $W_i$ on $G$ by picking $k$ nodes from $G$ and identifying the corresponding subgraph $S$ induced by these $k$ nodes
4. Embedding $W_i$ into $G$ by modifying $S$ to match structure of $W_i$.

We introduce each step in details in the following paragraphs:

**Step 1: Generating a random generator seed $\Omega_i$.** To generate an unforgettable watermarked graph, we form a random generator seed $\Omega_i$ [47] using user $i$'s signature $K^i_{priv}(T)$ and graph key $K^G$.

Suppose the system intends to generate a watermarked version of graph $G$ at time $T$ to share with user $i$. We begin by first sending user $i$ with the timestamp

$T$. User $i$ responds with its signature $K^i_{priv}(T)$, by encrypting the timestamp with its private key $K^i_{priv}$. Before proceeding further, we validate $K^i_{priv}(T)$ to ensure it is from user $i$, by decrypting it with user $i$'s public key $K^i_{pub}$. If the timestamps match, we combine the signature $K^i_{priv}(T)$ and the graph key $K^G$ to form the random generator seed $\Omega_i$ for user $i$. A mismatch may indicate that user $i$ is a potential malicious user.

Note that $\Omega_i$ cannot be formed alone by the data owner who only holds the graph key $K^G$, or by user $i$ who only owns its private key $K^i_{priv}$. Thus, results computed using $\Omega_i$, including the random graph $W_i$ generated (Step 2) and the choice of graph nodes to mark (Step 3), cannot be derived independently by the data owner or identified by user $i$.

**Step 2: Generating the watermark graph $W_i$.** We generate $W_i$ as an Erdos-Renyi random graph with edge probability of $p$ and node count $k$ ($k \ll n$, where $n$ is the number of nodes in $G$). The random edge generator uses $\Omega_i$ as the seed [47]. The $k$ nodes of $W_i$ are ordered as $\{v_1, v_2, ..., v_k\}$.

The key factor here is choosing the node count $k$ and the edge probability $p$. To ensure watermark uniqueness, Section 4.5.1 shows that the two parameters must satisfy:

$$k \geq (2+\delta)\log_q n \tag{4.1}$$

where $q = \frac{1}{\max(p, 1-p)}$, $\delta$ is a constant $> 0$.

Furthermore, it is easy to prove that $p = \frac{1}{2}$ minimizes the node count $k$ and the average edge count $p \cdot \binom{k}{2}$ of the watermark graph $W_i$. Intuitively, using a compact watermark graph not only reduces the amount of distortion to $G$, but also improves its robustness against malicious attacks. Thus, we configure $p = \frac{1}{2}$ and therefore $k = (2 + \delta) \log_2 n$. This produces a reasonably sized watermark graph ($k$ <100) even for extremely large graphs, *e.g.* the complete Facebook social graph (∼1 billion nodes in 2014).

**Step 3: Selecting the watermark placement on graph $G$.** Next, we identify $k$ nodes from $G$ and its corresponding subgraph $S$ to embed the watermark graph. To ensure reliable extraction, we must choose these $k$ nodes carefully, meeting these two requirements. *First*, using $\Omega_i$ generated in Step 1, the $k$ nodes must be chosen deterministically and remain distinguishable from the other nodes of $G$. *Second*, the set of the $k$ nodes chosen for different watermarks (or different $\Omega_i$ values) must be easily distinguishable from each other to reinforce watermark uniqueness.

Our biggest challenge in meeting these requirements is that we cannot use node IDs to distinguish nodes from each other. Node IDs or any type of metadata can be easily altered or stripped by attackers before or after leaking $G'$, thereby making extraction impossible.

We address this challenge by using local graph structure around each node as its "label." Specifically, we define a *node structure description* (NSD) as a descriptive feature of each node. A node $v$'s NSD is represented by an array of $v$'s sorted neighbor degrees. For example, if node $v$ has three neighbors with node degrees 2, 6, 4, respectively, then $v$'s NSD label is "2-4-6." We then hash $v$'s NSD label into a numerical value using a secure one-way hash *e.g.* SHA-1 [123], and refer to the result as node $v$'s *NSDhash*.

Next, we use $\Omega_i$ as the seed to randomly generate $k$ hash values, and use each as an index (*e.g.* using a mod function) to identify a node in $G$. It is possible that multiple nodes have the same NSDhash, *i.e.* a collision. If this happens, we resolve the collision by using $\Omega_i$ again as an index into a sorted list of these nodes with the same NSDhash. The nodes can be sorted by any deterministic order, *e.g.* node IDs in the original graph. Note that this process is only required for embedding (and not extraction), so any deterministic order chosen by the graph owner will suffice.

At the end of this step, we obtain $k$ ordered nodes from $G$, $X = \{x_1, x_2, ..., x_k\}$, and the corresponding subgraph $S = G[X]$ induced by the node set $X$ on $G$.

**Step 4: Embedding the watermark graph $W_i$ into graph $G$.** In this step, we embed the watermark graph $W_i$ by modifying the subgraph $S = G[X]$ to match $W_i$. Specifically, we match each (ranked) node in $W_i$, $\{v_1, v_2, ..., v_k\}$ with the corresponding node in $S$ (or $X$), $\{x_1, x_2, ..., x_k\}$:

$$W \to S, f(v_i) = x_i \tag{4.2}$$

Once the nodes are mapped, we then apply an XOR operation on each edge of the two graphs. That is, we consider the connection between $(v_i, v_j)$ or $(x_i, x_j)$ as one bit, *i.e.* an edge between $(v_i, v_j)$ or $(x_i, x_j)$ means 1 and no edge between $(v_i, v_j)$ or $(x_i, x_j)$ means 0. If an edge $(v_i, v_j)$ exists in $W_i$, we modify the corresponding edge value in $S$ from $(x_i, x_j)$ to $(x_i, x_j) \oplus 1$; and if no edge $(v_i, v_j)$ exists in $W_i$, we modify the edge value $(x_i, x_j)$ to $(x_i, x_j) \oplus 0$. When the above edge modification process ends, we also explicitly create edges between nodes $x_i$ and $x_{i+1}$ to maintain a connected subgraph. As a result, we transfer the subgraph $S$ into $S^{W_i}$ with the watermark graph $W_i$ embedded. The reason for choosing the XOR operation is that it allows the same watermark to be embedded in the graph multiple times (at multiple locations), thus reducing the risk of the watermark being detected and destroyed by attacks such as frequent subgraph mining. We will discuss this in more details in Section 4.6.

At the end of this step, we obtain a watermarked graph $G^{W_i}$ for user $i$. Before we distribute it to user $i$, we anonymize $G^{W_i}$ by completely (randomly) reassigning all node IDs. Such anonymization not only helps to protect user privacy, but also minimizes the opportunity for colluding attackers with multiple watermarked graphs to identify the embedded watermark (see Section 4.6).

### 4.4.2 Watermark Extraction

The watermark *extraction* process determines if a watermark graph $W_i$ is embedded in a target graph $G'$. If so, then $G'$ is a legitimate copy distributed to user $i$. The extraction process faces two key challenges. *First*, the target graph $G'$ can easily be modified by users/attackers during the graph distribution process. In particular, all node IDs can be very different from that of the original $G$. Thus extraction cannot rely on node IDs in $G'$. *Second*, identifying whether a subgraph exists in a large graph is equivalent to a subgraph matching problem, known to be NP-complete. To handle large graphs, we need a computationally efficient algorithm.

Our design addresses these two challenges by leveraging knowledge on the structure of the subgraph where the watermark was embedded. This eliminates the dependency on node IDs while significantly reducing the search space during the subgraph matching process. We describe our proposed design in detail below.

**Step 1: Regenerating the watermark.** The owner performs the extraction, and has access to the original graph $G$, graph key $K^G$, and user's signature $K^i_{priv}(T)$. For each user $i$, we combine its signature $K^i_{priv}(T)$ and graph key $K^G$ to form its random generator seed $\Omega_i$. Then, we follow step $2-4$ described in Section 4.4.1 to regenerate the watermark graph $W_i$, identify the $k$ ordered nodes from $G$ and their NSD labels, and finally the modified subgraph $S^{W_i}$ that was placed on a "clean" version of the watermarked graph $G^{W_i}$.

**Step 2: Identifying candidate watermark nodes on $G'$.** Given the $k$ nodes $X = \{x_1, x_2, ..., x_k\}$ identified from the original graph $G$, in this step we need to identify for each $x_j$, a set of candidate nodes on the target graph $G'$ that can potentially become $x_j$. We accomplish this by identifying all the nodes on $G'$ whose NSD labels are the same of $x_j$ in the "clean" version of the watermarked graph $G^{W_i}$. Since multiple nodes can have the same NSD label, this process will very likely produce multiple candidates. To shrink the candidate list, we examine the connectivity between candidate nodes of $X$ on $G'$ and compare it to that among $X$ on $G^{W_i}$. If two nodes $x_m$ and $x_n$ are connected in $G^{W_i}$, we prune their candidate node lists by removing any candidate node of $x_m$ that has no edge with any candidate node of $x_n$ on $G'$ and vice versa. This pruning process dramatically reduces the search space. After this step, we obtain for each $x_i$ the candidate node list $C_i$ on the target graph $G'$.

**Step 3: Detecting watermark graph $S^{W_i}$ on $G'$.** Given the candidate node list of each node in $X$, we now search for the existence of $S^{W_i}$ on the target graph $G'$. For this we apply a recursive algorithm to enumerate and prune the combinations of the candidate sets, until we identify $S^{W_i}$ or exhaust all the node candidates. The detailed algorithm is listed below. In this algorithm, we use a node list $Y$ to record the nodes in $G'$ which we have already finalized as the corresponding nodes in $S^{W_i}$, *i.e.* $Y = \{y_1, y_2, ..., y_m\}$ $(m \leq k)$. When the process starts, $Y = \emptyset$, $m = 0$.

**Algorithm 4.1** Recursive Algorithm for Detecting $S^{W_i}$ on $G'$.

```
1. Function: SubgraphDetection($G'$, $S^{W_i}$, $\{C_1, C_2, ..., C_k\}$, $Y$, $m$)
2. Input: Graph $G'$, watermark graph $S^{W_i}$, candidate node list $C_i$ for each
   node $x_i$ in $X$, identified node list $Y = \{y_1, y_2, ..., y_m\}$ $(m < k)$
3. Output: Identified node list $Y = \{y_1, y_2, ..., y_{m+1}\}$
4. for each node $c \in C_{m+1}$ do:
   (a) if $c \notin Y$ and each edge $(c, y_t)$ in $G'$ $(t = 1..m)$ is the same as the edge
       $(x_{m+1}, x_t)$ in $S^{W_i}$ $(t = 1..m)$ then
   (b) $Y = Y \cup c$, $m = m + 1$
   (c) if $m == k$ then
   (d) Return $Y$
   (e) else
   (f) SubgraphDetection($G'$, $S^{W_i}$, $\{C_1, C_2, ..., C_k\}$,$Y$, $m$)
   (g) end if
```

```
        (h) $Y = Y \setminus c$, $m = m - 1$
        (i) **end if**
5. **end for**
6. Return $Y$
```

**Discussion.** The above design shows that our watermark extraction algorithm simplifies the subgraph search problem by restricting it to a small number of selected nodes from a graph, thus avoiding the NP-complete subgraph matching problem.

To illustrate the efficiency of our algorithm, we now show an estimation of the computational complexity. Assume that the number of candidates for each watermark node $x_i$ is $|C_i|$, and the probability that an edge between node $c_{im} \in |C_i|$ and node $c_{jn} \in |C_j|$ is $p_{ij}$. Moreover, since we prove that the probability of an edge between node $x_i$ and node $x_j$ is $\frac{1}{2}$ in Section 4.5.1, the probability that the connectivity between $(c_{im}, c_{jn})$ matches the connectivity between $(x_i, x_j)$ is $\frac{1}{2} \cdot p_{ij} + \frac{1}{2} \cdot (1 - p_{ij}) = \frac{1}{2}$. We can show that to identify a node list with $m$ nodes in Algorithm 4.1, we need to match $\binom{m}{2}$ node pairs. Thus, the probability to identify a node list with $m$ nodes is $\frac{1}{2}^{\binom{m}{2}}$, and the expected number of node combinations is $\prod_{i=1}^{m} |C_i| \cdot \frac{1}{2}^{\binom{m}{2}}$. Thus, the computational complexity of Algorithm 4.1 is proportional to the sum of node combinations at each step, *i.e.* $O(\sum_{m=2}^{k} \prod_{i=1}^{m} |C_i| \cdot \frac{1}{2}^{\binom{m}{2}})$. Note that we do not consider the fixed $k-1$ edges between $(x_{i-1}, x_i)$ for simplicity.

This result shows that as more nodes are identified in Algorithm 4.1, fewer node combinations exists, which approximates to 0 (as shown in Section 4.5.1). This means the major computation cost of our algorithm comes from the initial few steps and is dominated by the size of their candidates. Note that we target real graphs with very high level of node heterogeneity, *e.g.* small-world, power-law or highly clustered graphs, which leads to small candidate size in most cases. In other words, the computational complexity of our algorithm is low in real graphs. In practice, our system can efficiently extract watermarks from real, million-node graphs, and do so in a few minutes on a single commodity server (Section 4.7.3).

## 4.5 Fundamental Properties

Having described the basic watermark system, we now present detailed analysis on its two fundamental properties: *watermark uniqueness* where each watermark must be unique to the corresponding user, and *watermark detectability* where the presence of a watermark should not be easily detectable by external users without the knowledge of the seed $\Omega_i$ associated with user $i$.

### 4.5.1 Watermark Uniqueness

As a proof of ownership, each embedded watermark should be unique for its user. That is, given the original graph $G$ and the seed $\Omega_i$ associated with user $i$, the embedded watermark graph $S^{W_i}$ should not be isomorphic to any subgraph of $G^{W_j}$ $(i \neq j)$ where $G^{W_j}$ is the watermarked graph for user $j$. Meanwhile, $S^{W_i}$ should not be isomorphic to any subgraph of the original graph $G$. The following proof shows that with high probability, our proposed graph watermark system produces unique watermarks for any graph $G$.

**Theorem 1.** *Given a graph $G$ with $n$ nodes, let $k \geq (2+\delta)\log_2 n$ for a constant $\delta > 0$. We apply the following process to create a watermarked graph $G^{W_i}$ for user $i$:*

- *We create $k$ nodes, $V = \{v_1, v_2, ..., v_k\}$, and generate a random graph $W_i$ on $V$ with an edge probability of $\frac{1}{2}$.*
- *We randomly select $k$ nodes, $X = \{x_1, x_2, ..., x_k\}$ from $G$, and identify the subgraph corresponding to these $k$ nodes $S = G[X]$.*
- *Using $W_i$, we modify $S$ as follows: we first map each node $x_i$ in $X$ to a node $v_i$ in $V$. Let $e(u, v) = 1$ denote an edge exists between node $u$ and $v$ and $e(u, v) = 0$ denote otherwise. We modify each $e(x_i, x_j)$ in $S$ to $e(x_i, x_j) \oplus e(v_i, v_j)$. We then*

*explicitly connect nodes $x_i$ and $x_{i+1}$,* i.e. *$e(x_i, x_{i+1}) = 1$. The resulting $S$ now becomes $S^{W_i}$, and the resulting $G$ becomes $G^{W_i}$.*

*Let $G^{W_l}$ denote a watermarked graph for user $l$ ($l \neq i$), built using a different seed $\Omega_l$.*

*Then with low probability, any subgraph of $G^{W_l}$ or $G$ is isomorphic to $S^{W_i}$.*

*Proof.* We first show that with low probability, any subgraph of $G^{W_l}$ is isomorphic to $S^{W_i}$. Let $Y = \{y_1, y_2, ...y_k\}$ be a set of ordered nodes in $G^{W_l}$, where each $y_i$ maps to a node $x_i$ in $X$. We define an event $\mathcal{E}_Y$ occurs if the subgraph $G^{W_l}[Y]$ is isomorphic to $G^{W_i}[X]$ or $S^{W_i}$. Then the event $\mathcal{E}$ representing the fact that there exists at least one subgraph on $G^{W_l}$ that is isomorphic to $S^{W_i}$ is the union of events $\mathcal{E}_Y$ on all possible $Y$, *i.e.* $\mathcal{E} = \cup_Y \mathcal{E}_Y$.

Next, we compute the probability of event $\mathcal{E}$ by those of individual event $\mathcal{E}_Y$. Specifically, we first show that the probability of an edge exists between node $x_i$ and $x_j$ ($j \neq i+1$) in $S^{W_i} = G^{W_i}[X]$ is $\frac{1}{2}$. This is because each edge in the random graph $W_i$ is independently generated with probability $\frac{1}{2}$. After performing the XOR operation between $W_i$ and $S$, the probability of an edge exists between $x_i$ and $x_j$ ($j \neq i+1$) on $S^{w_i}$ is:

$$\frac{1}{2} \cdot p_{ij} + (1 - p_{ij}) \cdot \frac{1}{2} = \frac{1}{2} \tag{4.3}$$

where $p_{ij}$ is the probability that an edge exists between $x_i$ and $x_j$ on $S$. Thus the result of XOR between $W_i$ and $S$ is also a random graph, and its edge generation is independent of that in $G^{W_l}, l \neq i$. Furthermore, it is easy to show that our design applies XOR operations on $\binom{k}{2} - (k-1)$ node pairs on the $k$ nodes, and each node pair has an edge with a probability of $\frac{1}{2}$. Thus, the probability of a subgraph $G^{W_l}[Y]$ being isomorphic to $S^{W_i}$ is:

$$P(\mathcal{E}_Y) = \frac{1}{2}^{\binom{k}{2}-(k-1)} \cdot \beta \tag{4.4}$$

where $\beta \leq 1$ is the probability that every $(y_i,\, y_{i+1})$ pair in $G^{W_l}[Y]$ is connected. Thus $P(\mathcal{E}_Y) \leq \frac{1}{2}^{\binom{k}{2}-(k-1)}$.

Since $\mathcal{E} = \cup_Y \mathcal{E}_Y$ and there are less than $n^k$ possible sets of $k$ ordered nodes in $G^{W_l}$, we use the Union Bound to compute the probability of event $\mathcal{E}$ as follows:

$$\begin{aligned} &P(\mathcal{E}) < n^k \cdot P(\mathcal{E}_Y) \leq n^k \cdot \frac{1}{2}^{\binom{k}{2}-(k-1)} \\ &= 2^{\frac{k^2}{2+\delta}} \cdot \frac{1}{2}^{\frac{k^2-3k}{2}+1} = \frac{1}{2}^{\frac{\delta k^2}{2(2+\delta)} - \frac{3k}{2}+1} \end{aligned} \tag{4.5}$$

The above equation shows that the probability $P(\mathcal{E})$ reduces exponentially to 0 as $k$ increases.

Finally, we can apply the same method to show that with low probability, any subgraph of $G$ is isomorphic to $S^{W_i}$. This is because the XOR operations between $W_i$ and $S$ produce a random graph that is independent of $G$. □

### 4.5.2 Watermark Detectability

In addition to providing uniqueness, a practical watermark design should also offer low detectability, *i.e.*, with low probability each watermark gets identified by external users/attackers. This means that without knowing the seed $\Omega_i$ associated with user $i$,

the embedded watermark graph $S^{W_i}$ should not be easily distinguishable from the rest of the graph $G^{W_i}$. Therefore, the detectability would depend heavily on the topology of the original graph $G$, *i.e.* a watermark graph can be well hidden inside a graph $G^{W_i}$ if its structural property is not too different from that of $G$.

In the following, we examine the detectability of watermarks in terms of *a graph's suitability for watermarking.* This is because directly quantifying the detectability is not only highly computational expensive[2], but also lacks a proper metric. Instead, we cross-compare the key structural properties of $S^{W_i}$ and $G$, and define $G$ as being suitable for watermarking if its structure properties are similar to that of $S^{W_i}$, implying a low watermark detectability.

**Suitability for Watermarking.** To evaluate a graph's suitability for watermarks, we first study the key structural property of the embedded watermark graph $S^{W_i}$. To guarantee watermark uniqueness and minimize distortion, the wa-

[2]Each embedded watermark graph is similar to a random graph with $\frac{1}{2}$ edge probability. Thus the detectability is low if certain subgraphs of $G$ are also random graphs with similar edge probabilities. Yet identifying these subgraphs (and the embedded watermark graph) on a large graph incurs significant computation overhead.

termark graph $S^{W_i}$ needs to be a random graph with an edge probability of $\frac{1}{2}$ (except for the fixed edges between $x_i, x_{i+1}$ node pairs), and include $k = (2+\delta)\log_2 n$ nodes. Thus its average node degree is at least $(k+1)/2$ and its average graph density is $(\binom{k}{2} + k - 1)/2$.

Given these properties of the embedded watermark, we note that watermark node degree and density can be higher than those of many real-world graphs. Intuitively, to ensure low detectability of such a watermark graph, suitable graphs should include a set of nodes ($D$) that are difficult to distinguish from the watermark nodes in term of node degree and subgraph density. Specifically, a suitable graph dataset needs to contain a set of nodes $D$ with degree comparable or higher than the watermark graph node degree; and the density of the subgraph on $D$ is at least comparable to the watermark graph density. If these two properties hold, the embedded watermark graph cannot be easily distinguished from $D$ in the graph, and therefore cannot be detected by attackers.

To capture the above intuition, we define that a graph $G$ is suitable for watermarking if its node degree and graph density satisfy the following two criteria. First, the minimum and maximum node degree of $G$, denoted as $N_{min}(G)$ and $N_{max}(G)$ respectively, need to satisfy $N_{min}(G) \leq (k+1)/2 \leq N_{max}(G)$. Second, across all $k$-node subgraphs of $G$ whose node degree expectation is greater than $(k+1)/2$, the minimum and maximum graph density need to satisfy $D_{min}(k) \leq$

$(\binom{k}{2}+k-1)/2 \leq D_{max}(k)$ [3]. Together, these two criteria ensure that the embedded watermark graph can be "well hidden" inside $G^{W_i}$.

Table 4.1: Statistics of 48 of today's network graphs. $k$ is the watermark size.

| Graph Category | Graph | # of Nodes | # of Edges | Avg. Deg. | $k$ |
|---|---|---|---|---|---|
| Facebook | Russia | 97,134 | 289,324 | 6.0 | 39 |
| | L.A. | 603,834 | 7,676,486 | 25.4 | 45 |
| | London | 1,690,053 | 23,084,859 | 27.3 | 48 |
| Other Social Networks | Epinions (1) | 75,879 | 405,740 | 10.7 | 38 |
| | Slashdot (08/11/06) | 77,360 | 507,833 | 13.1 | 38 |
| | Twitter | 81,306 | 1,342,303 | 33.0 | 38 |
| | Slashdot (09/02/16) | 81,867 | 497,672 | 12.2 | 38 |
| | Slashdot (09/02/21) | 82,140 | 500,481 | 12.2 | 38 |
| | Slashdot (09/02/22) | 82,168 | 543,381 | 13.2 | 38 |
| | GPlus | 107,614 | 12,238,285 | 227.5 | 39 |
| | Epinions (2) | 131,828 | 711,496 | 10.8 | 40 |
| | Youtube | 1,134,890 | 2,987,624 | 5.3 | 47 |
| | Pokec | 1,632,803 | 22,301,964 | 27.3 | 48 |
| | Flickr | 1,715,255 | 15,555,041 | 18.1 | 48 |
| | Livejournal | 5,204,176 | 48,942,196 | 18.8 | 52 |
| Citation | Patents | 23,133 | 93,468 | 8.1 | 34 |

[3] To avoid computationally prohibitive subgraph enumeration, we apply a sampling method to estimate them with full details in [163].

| | | | | | |
|---|---|---|---|---|---|
| Networks | ArXiv (Theo. Cit.) | 27,770 | 352,304 | 25.4 | 34 |
| | ArXiv (Phy. Cit.) | 34,546 | 420,899 | 24.4 | 35 |
| | ArXiv (Phy.) | 12,008 | 118,505 | 19.7 | 32 |
| Collaboration | ArXiv (Astro) | 18,772 | 198,080 | 21.1 | 33 |
| Networks | DBLP | 317,080 | 1,049,866 | 6.6 | 43 |
| | ArXiv (Condense) | 3,774,768 | 16,518,947 | 8.8 | 51 |
| Communication | Email (Enron) | 36,692 | 183,831 | 10.0 | 35 |
| Networks | Email (Europe) | 265,214 | 365,025 | 2.8 | 42 |
| | Wiki | 2,394,385 | 4,659,565 | 3.9 | 49 |
| | Stanford | 281,903 | 1,992,636 | 14.1 | 42 |
| Web | NotreDame | 325,729 | 1,103,835 | 6.8 | 43 |
| graphs | BerkStan | 685,230 | 6,649,470 | 19.4 | 45 |
| | Google | 875,713 | 4,322,051 | 9.9 | 46 |
| Location based | Brightkite | 58,228 | 214,078 | 7.4 | 37 |
| OSNs | Gowalla | 196,591 | 950,327 | 9.7 | 41 |
| | Oregon (1) | 11,174 | 23,409 | 4.2 | 31 |
| AS | Oregon(2) | 11,461 | 32,730 | 5.7 | 32 |
| Graphs | CAIDA | 26,475 | 53,381 | 4.0 | 34 |
| | Skitter | 1,696,415 | 11,095,298 | 13.1 | 48 |
| | Gnutella (02/08/04) | 10,876 | 39,994 | 7.4 | 31 |
| | Gnutella (02/08/25) | 22,687 | 54,705 | 4.8 | 34 |
| P2P networks | Gnutella (02/08/24) | 26,518 | 65,369 | 4.9 | 34 |
| | Gnutella (02/08/30) | 36,682 | 88,328 | 4.8 | 35 |

| | | | | | |
|---|---|---|---|---|---|
| | Gnutella (02/08/31) | 62,586 | 147,892 | 4.7 | 37 |
| Amazon Co-purchasing Networks | Amazon (03/03/02) | 262,111 | 899,792 | 6.9 | 42 |
| | Amazon (2012) | 334,863 | 925,872 | 5.5 | 43 |
| | Amazon (03/03/12) | 400,727 | 2,349,869 | 11.7 | 43 |
| | Amazon (03/06/01) | 403,394 | 2,443,408 | 12.1 | 43 |
| | Amazon (03/05/05) | 410,236 | 2,439,437 | 11.9 | 43 |
| Road Networks | Pennsylvania | 1,088,092 | 1,541,898 | 2.8 | 47 |
| | Texas | 1,379,917 | 1,921,660 | 2.8 | 47 |
| | California | 1,965,206 | 2,766,607 | 2.8 | 49 |

Table 4.2: Suitability of watermarking for 48 of today's network graphs, determined by comparing their node degree distribution $[N_{min}(G), N_{max}(G)]$ and $k$-node subgraph density $[D_{min}(k), D_{max}(k)]$ to those of the embedded watermark graphs. 35 out of these 48 graphs are suitable for watermarking.

| Graph Category | Graph | Node Degree Criterion | | $k$-node Subgraph Density Criterion | | Suitability |
|---|---|---|---|---|---|---|
| | | $(k+1)/2$ | $[N_{min}(G), N_{max}(G)]$ | Watermark | $[D_{min}(k), D_{max}(k)]$ | |
| Facebook | Russia | 20 | [1, 748] | 390 | [45, 701] | **Yes** |
| | L.A. | 23 | [1, 2141] | 517 | [44, 975] | **Yes** |
| | London | 24 | [1, 1483] | 588 | [47, 1128] | **Yes** |

| | | | | | | |
|---|---|---|---|---|---|---|
| Other Social Networks | Epinions (1) | 19 | [1,3044] | 370 | [47,649] | **Yes** |
| | Slashdot (08/11/06) | 19 | [1, 2540] | 370 | [38, 668] | **Yes** |
| | Twitter | 19 | [1, 3383] | 370 | [44, 703] | **Yes** |
| | Slashdot (09/02/16) | 19 | [1, 2546] | 370 | [38, 669] | **Yes** |
| | Slashdot (09/02/21) | 19 | [1, 2548] | 370 | [38, 669] | **Yes** |
| | Slashdot (09/02/22) | 19 | [1, 2553] | 370 | [38, 673] | **Yes** |
| | GPlus | 20 | [1, 20127] | 389.5 | [53, 741] | **Yes** |
| | Epinions (2) | 20 | [1, 3558] | 409.5 | [51, 780] | **Yes** |
| | Youtube | 24 | [1, 28754] | 563.5 | [47, 815] | **Yes** |
| | Pokec | 24 | [1, 14854] | 587.5 | [47, 979] | **Yes** |
| | Flickr | 24 | [1, 27236] | 588 | [51, 1128] | **Yes** |
| | Livejournal | 26 | [1, 15017] | 689 | [51, 1326] | **Yes** |
| Citation Networks | Patents | 17 | [1, 280] | 297 | [37, 373] | **Yes** |
| | ArXiv (Theo. Cit.) | 17 | [1, 2468] | 297 | [36, 534] | **Yes** |
| | ArXiv (Phy. Cit.) | 18 | [1, 846] | 314.5 | [36, 544] | **Yes** |
| Collaboration Networks | ArXiv (Phy.) | 16 | [1, 491] | 263.5 | [45, 496] | **Yes** |
| | ArXiv (Astro) | 17 | [1, 504] | 280 | [37, 528] | **Yes** |
| | DBLP | 22 | [1,343] | 472.5 | [43,903] | **Yes** |
| | ArXiv (Condense) | 26 | [1, 793] | 663 | [50,1063] | **Yes** |
| Communication Networks | Email (Enron) | 18 | [1,1383] | 314.5 | [43,515] | **Yes** |
| | Email (Europe) | 21 | [1,7636] | 451 | [74,683] | **Yes** |
| | Wiki | 25 | [1, 100029] | 612 | [65, 1066] | **Yes** |
| | Stanford | 21 | [1,38625] | 451 | [66,861] | **Yes** |

| | | | | | | |
|---|---|---|---|---|---|---|
| Web | NotreDame | 22 | [1,10721] | 472.5 | [60,903] | **Yes** |
| graphs | BerkStan | 23 | [1,84230] | 517 | [79,990] | **Yes** |
| | Google | 23 | [1, 6332] | 540 | [72, 1033] | **Yes** |
| Location based | Brightkite | 19 | [1,1134] | 351 | [41,665] | **Yes** |
| OSNs | Gowalla | 21 | [1,14730] | 430 | [44,723] | **Yes** |
| | Oregon (1) | 16 | [1,2389] | 247.5 | [95,352] | **Yes** |
| AS | Oregon(2) | 16 | [1,2432] | 263.5 | [79,476] | **Yes** |
| Graphs | CAIDA | 17 | [1,2628] | 297 | [113,436] | **Yes** |
| | Skitter | 24 | [1, 35455] | 588 | [52, 1128] | **Yes** |
| | Gnutella (02/08/04) | 16 | [1,103] | 247.5 | [30,80] | No |
| | Gnutella (02/08/25) | 17 | [1,66] | 297 | [0,0] | No |
| P2P networks | Gnutella (02/08/24) | 17 | [1,355] | 297 | [0,44] | No |
| | Gnutella (02/08/30) | 18 | [1,55] | 314.5 | [35,70] | No |
| | Gnutella (02/08/31) | 19 | [1, 95] | 351 | [39,76] | No |
| | Amazon (03/03/02) | 21 | [1,420] | 451 | [88,132] | No |
| Amazon | Amazon (2012) | 22 | [1,549] | 472.5 | [0,0] | No |
| Co-purchasing | Amazon (03/03/12) | 22 | [1,2747] | 472.5 | [52,285] | No |
| Networks | Amazon (03/06/01) | 22 | [1, 2752] | 473 | [52, 333] | No |
| | Amazon (03/05/05) | 22 | [1,2760] | 472.5 | [50,333] | No |
| Road | Pennsylvania | 24 | [1,9] | 563.5 | [0,0] | No |
| Networks | Texas | 24 | [1,12] | 563.5 | [0,0] | No |
| | California | 25 | [1, 12] | 612 | [0, 0] | No |

**Suitability of Real Graph Datasets.** We measure the suitability of watermarks in 48 real networks graphs. These graphs represent vastly different types of networks and a wide range of structural topologies with size ranging from $10K$ nodes and $39K$ edges to $5M$ nodes and $48M$ edges. These graphs represent vastly different types of networks and a wide range of structural topologies. They include 3 social graphs generated from Facebook regional networks matching Russia, L.A., and London [153]. They include 12 other graphs from online social networks, including Twitter [87], Youtube [157], Google+ [87], Slovakia Pokec [139], Flickr [104], Livejournal [104], 2 snapshots from Epinions [121], and 4 snapshots from Slashdot [86]. We also add 3 citation graphs from arXiv and U.S. Patents [81], 4 graphs capturing collaborations in arXiv [81] and DBLP [157], 3 communication graphs generated from 2 Email networks [85, 86] and Wiki Talk [83], 4 web graphs [79, 12], 2 location-based online social graphs from Brightkite and Gowalla [30], 5 snapshots of P2P file sharing graph from Gnutella [85], 4 Internet Autonomous System (AS) maps [81], 5 snapshots of Amazon co-purchasing networks [82, 157], and 3 U.S. road graphs [79]. We list statistics of all graphs in Table 4.1, and their corresponding watermark statistics in Table 4.2.

For all graphs, we use $\delta = 0.3$ to ensure a 99.999% watermark uniqueness, and list their watermark size $k$ in Table 4.1. We also show the two above criteria:

node degree and $k$-node subgraph density in Table 4.2. If a graph satisfies both criteria, our results will hold for any watermarks embedded on it.

We can make two observations from Table 4.2. *First*, 35 out of our 48 total graphs are suitable for watermarking. Also note that graphs describing similar networks are consistent in their suitability. For example, all 15 graphs from various online social networks are suitable for watermarks! *Second*, all the 13 graphs unsuitable for watermarks come from only 3 kinds of networks, *i.e.* copurchasing networks, P2P networks, and Road networks. These results in each group are self consistent. These results support our assertion that our proposed watermarking mechanism is applicable to most of today's network graphs with low detection risk. In practice, the owner of a graph can apply the same mechanism to determine if her graph is suitable for our watermark scheme.

Table 4.3: Size and density of subgraph on nodes with degree $>$ $(k+1)/2$ in each graph. Size is the number of subgraph nodes, and density is quantified as average edges each node having inside the subgraph.

| Graph | Subgraph | | Watermark Graph | | Suitability |
|---|---|---|---|---|---|
| | Node # | Avg. Deg. | $k$ | Avg. Deg. | |
| Russia | 4,794 | 22.2 | 39 | 20.0 | **Yes** |
| L.A. | 196,174 | 49.2 | 45 | 23.0 | **Yes** |

| | | | | | |
|---|---|---|---|---|---|
| London | 562,075 | 56.1 | 48 | 24.5 | **Yes** |
| Epinions (1) | 7,083 | 68.7 | 38 | 19.5 | **Yes** |
| Slashdot (08/11/06) | 9,908 | 53.4 | 38 | 19.5 | **Yes** |
| Twitter | 34,014 | 60.5 | 38 | 19.5 | **Yes** |
| Slashdot (09/02/16) | 10,065 | 53.0 | 38 | 19.5 | **Yes** |
| Slashdot (09/02/21) | 10,105 | 53.2 | 38 | 19.5 | **Yes** |
| Slashdot (09/02/22) | 10,605 | 53.4 | 38 | 19.5 | **Yes** |
| GPlus | 68,828 | 347.1 | 39 | 20.0 | **Yes** |
| Epinions (2) | 10,363 | 83.5 | 40 | 20.5 | **Yes** |
| Youtube | 31,720 | 45.1 | 47 | 24.0 | **Yes** |
| Pokec | 564,001 | 53.0 | 48 | 24.5 | **Yes** |
| Flickr | 136,202 | 174.5 | 48 | 24.5 | **Yes** |
| Livejournal | 945,567 | 57.5 | 52 | 26.5 | **Yes** |
| Patents | 2,370 | 15.6 | 34 | 17.5 | **Yes** |
| ArXiv (Theo. Cit.) | 12,054 | 43.4 | 34 | 17.5 | **Yes** |
| ArXiv (Phy. Cit.) | 14,785 | 37.9 | 35 | 18.0 | **Yes** |
| ArXiv (Phy.) | 2,860 | 62.5 | 32 | 16.5 | **Yes** |
| ArXiv (Astro) | 6,536 | 42.9 | 33 | 17.0 | **Yes** |
| DBLP | 15,004 | 17.3 | 43 | 22.0 | **Yes** |
| ArXiv (Condense) | 178,455 | 16.0 | 51 | 26.0 | **Yes** |
| Email (Enron) | 3,481 | 48.2 | 35 | 18.0 | **Yes** |
| Email (Europe) | 1,779 | 44.0 | 42 | 21.5 | **Yes** |
| Wiki Talk | 21,253 | 83.1 | 49 | 25.0 | **Yes** |
| Stanford | 35,600 | 42.1 | 42 | 21.5 | **Yes** |

| | | | | | |
|---|---|---|---|---|---|
| NotreDame | 16,831 | 38.7 | 43 | 22.0 | **Yes** |
| BerkStan | 110,202 | 57.0 | 45 | 23.0 | **Yes** |
| Google | 55,431 | 14.8 | 46 | 23.5 | **Yes** |
| Brightkite | 4,586 | 30.8 | 37 | 19.0 | **Yes** |
| Gowalla | 17,946 | 39.3 | 41 | 21.0 | **Yes** |
| Oregon (1) | 264 | 17.1 | 31 | 16.0 | **Yes** |
| Oregon(2) | 579 | 31.0 | 32 | 16.5 | **Yes** |
| CAIDA | 575 | 16.0 | 34 | 17.5 | **Yes** |
| Skitter | 146,601 | 50.0 | 48 | 24.5 | **Yes** |
| Gnutella (02/08/04) | 796 | 5.2 | 31 | 16.0 | No |
| Gnutella (02/08/25) | 499 | 2.0 | 34 | 17.5 | No |
| Gnutella (02/08/24) | 709 | 2.7 | 34 | 17.5 | No |
| Gnutella (02/08/30) | 1,001 | 3.8 | 35 | 18.0 | No |
| Gnutella (02/08/31) | 1,276 | 3.6 | 37 | 19.0 | No |
| Amazon (03/03/02) | 3,727 | 2.8 | 42 | 21.5 | No |
| Amazon (2012) | 5,318 | 2.5 | 43 | 22.0 | No |
| Amazon (03/03/12) | 25,717 | 6.7 | 43 | 22.0 | No |
| Amazon (03/06/01) | 28,081 | 7.3 | 43 | 22.0 | No |
| Amazon (03/05/05) | 28,044 | 7.5 | 43 | 22.0 | No |
| Pennsylvania | 0 | 0 | 47 | 24.0 | No |
| Texas | 0 | 0 | 47 | 24.0 | No |
| California | 0 | 0 | 49 | 25.0 | No |

To understand key properties determining whether a graph is suitable for watermarking, we measure various graph structural properties, including average node degree, node degree distribution, clustering coefficient, average path length, and assortativity. We also consider the size and density of subgraphs on nodes with degree more than watermark minimum average degree $(k+1)/2$. Our measurement results show that the size and density of subgraphs on nodes with degree $> (k+1)/2$ are the most important properties to determine suitability. Here, the size of these subgraphs is the number of nodes in the subgraph, and the density of the subgraph is measured as the average edges each node has inside the subgraph, *i.e.* average degree inside the subgraph. As shown in Table 4.3, unsuitable graphs do not have subgraphs with density to comparable to watermarks, while subgraphs with the desired density can be found in graphs deemed suitable. These results are consistent with our intuition on quantifying suitability of watermarks.

**Summary.** Since the average watermark subgraph has high node degree and density, a graph suitable for watermarking must include a set of nodes, whose degree and subgraph density are comparable or even higher than watermark subgraphs. We propose two criteria targeting at node degree and subgraph density respectively to quantify whether a graph is suitable for watermarking. We collect a large set of available graph datasets and find 35 out of 48 real graphs are suitable. We expect similar suitability results in other real network graphs.

## 4.6 More Robust Watermarks

Our basic design provides the fundamental building blocks of graph watermarking with little consideration of external attacks. In practice, malicious users can seek to detect or destroy watermarked graphs. Here, we first describe external attacks on watermarks, and then present advanced features that defend against the attacks. Note that these improvement techniques aim to increase the cost of attacks rather than disabling them completely. Finally, we re-evaluate the watermark uniqueness of the advanced design.

### 4.6.1 Attacks on Watermarks

As discussed earlier, our attack model includes attacks trying to destroy watermarks while preserving the topology of the original graph. Based on the number of attackers, attacks on watermarks fall under our two models: single attacker and colluding attackers. With access to only one watermarked graph, a single attacker can modify nodes and/or edges in the graph to destroy watermarks. With multiple watermarked graphs, colluding attackers can perform more sophisticated attacks by cross-comparing these graphs to detect or remove watermarks.

**Single Attacker Model.** The naive edge attack is easiest to launch, and tries to disrupt the watermark by randomly adding or removing edges on the watermarked graph. For the attacker, there is a clear tradeoff between the severity of the attack (number of edges or nodes modified), and the structural change or distortion applied to the graph structure.

At first glance, this attack seems weak and unlikely to be a real threat. The probability of the attacker modifying one edge or node in the embedded watermark graph $W_i$ is extremely low, given the relatively small size of $W_i$ compared to the graph. As shown later, however, this attack can be quite disruptive in practice. By modifying a node $n_i$ or an edge connected to $n_i$, the attack impacts all of $n_i$'s neighboring nodes, since their NSD labels will be modified. These NSD label changes, while small, are enough to make locating nodes in the watermark graph very difficult. This effect is exacerbated in social graphs that exhibit a small world structure, since any change to a supernode's degree will impact a disproportionately large portion of nodes in the graph.

Some versions of this attack would either release a partial subgraph of the watermarked graph, or merge multiple watermarked graphs. In both cases, this destroys the embedded watermarks, but also significantly distorts the graphs and reduces their usability. We do not consider these disruptive attacks in our study, and target them for future work.

**Collusion Attacks.** By obtaining multiple watermarked graphs, an attacker can compare these graphs to eliminate watermarks. Since we anonymize each watermarked graph by randomly reassigning node IDs (see Section 4.4.1), attackers cannot directly match individual nodes across graphs. To compare multiple graphs, we apply the deanonymization methods proposed in [107, 108]. Specifically, we first match 1000 highest degree nodes between two graphs based on their degree and neighborhood connectivity [108], and then start from these nodes to find new mappings with the network structure and the previously mapped nodes [107].

Using deanonymization techniques, attackers can then build a "clean" graph, where an edge exists if it exists in the majority of the watermarked graphs. Since embedded watermark graphs are likely embedded at different locations on each graph, a majority vote approach effectively removes the contributions from watermark subgraphs, leading to a graph that closely approximates the original $G$.

### 4.6.2 Improving Robustness against Attacks

The attacks discussed above can disrupt the watermark extraction process in two ways. First, adding or deleting nodes/edges in $G'$ changes node degrees, and therefore nodes' NSD labels, thereby disrupting the identification of candidate nodes during the second step of the extraction process; second, adding or deleting nodes/edges inside the embedded watermark graph $S^{W_i}$ can change the structure of the watermark graph, making it difficult to identify during the third step of the extraction process. To defend against these attacks, we propose five improvements over the basic extraction design to produce an improved watermark generation algorithm.

**Improvements #1, #2: Addressing changes to node neighborhoods.** Extracting a watermark involves searching through nodes in $G'$ by their NSD labels. By adding or deleting nodes/edges, attackers can effectively change NSD labels across the graph. To address this, we propose two changes to the basic extraction design.

*First*, we bucketize node degrees (with bucket size $B$) to reduce the sensitivity of a node's NSD label to its neighbors' node degrees. For example, with $B = 5$, a node with degree 9 will stay in the same bucket even if one of its edges has been removed (reducing its node degree to 8).

*Second*, when selecting a watermark node's candidate node list, we replace the exact NSD label matching with the approximate NSD label matching. A match is found if the overlap between two bucketized NSD labels exceeds a threshold $\theta$. For example, with $\theta = 50\%$, a node with bucketized NSD label "1-2-3-4" would match a node with label "1-2-3" since the overlap is $75\% > \theta$.

These changes clearly allow us to identify more candidates for each watermark node, thus improving robustness against small local modifications. On the other hand, more candidates lead to more computation during the subgraph matching step (step 3 in Section 4.4.2). Such expansion, however, does not affect watermark uniqueness and detectability, since they are unrelated to the size of candidate pools.

**Improvement #3, #4: Addressing changes to subgraph structure.** Random changes made to $G'$ may directly impact a node or edge in the embedded watermark. To address this, we propose two techniques.

*First*, we add redundancy to watermarks by embedding the same watermark graph $W_i$ into $m$ disjoint subgraphs $S_1, S_2, ...S_m$ from the original graph $G$. This greatly increases the probability of the owner locating at least one unmodified copy of $W_i$ during extraction, even in the presence of attacks that make significant changes to nodes and edges in $G'$. Note that since we embed watermarks on disjoint subgraphs, this does not affect watermark uniqueness $1 - P(\mathcal{E})$. While embedding $m$ watermarks will impact false positive, which is $1 - (1 - P(\mathcal{E}))^m$.

*Second*, it is still possible that all the watermark graphs are "destroyed" by the attacker and there are no matches in the extraction process. If this happens, we replace the exact subgraph matching in the step 3 of the extraction process with the approximate subgraph matching. That is, a subgraph matches the watermark graph if the amount of edge difference between the two is less than a threshold $L$. By relaxing the search criteria used in step 3 of the extraction process, this technique allows us to identify "partially" damaged watermarks, thus again improving robustness against attacks. However, it can also increase false positives in watermark extraction, reducing watermark uniqueness. We show in Section 4.6.3 that the impact on watermark uniqueness can be tightly bounded by controlling $L$.

**Improvement #5: Addressing Collusion Attacks.** Recall that for powerful attackers able to match graphs at an individual node level, they can leverage majority votes across multiple watermarked graphs to remove watermarks. To defend against this, our insight is to embed watermarks that have some portion of spatial overlap in the graph, such that those components will survive majority votes over graphs.

We propose a *hierarchical* watermark embedding process to defend against collusion attacks. To build watermarked graphs for $M$ users, we uniform-randomly divide the $M$ users into 2 groups ($a_1$ and $a_2$) and associate each group with a public-private key pair $< K^{a_1}_{pub}, K^{a_1}_{priv} >$ or $< K^{a_2}_{pub}, K^{a_2}_{priv} >$, which is generated and held by the data owner. We repeat this to randomly divide $M$ users into another 2 groups ($b_1$ and $b_2$) associated with group key pairs $< K^{b_1}_{pub}, K^{b_1}_{priv} >$ and $< K^{b_2}_{pub}, K^{b_2}_{priv} >$ separately. After this step, each user is assigned to two groups [4]. For example, a user $i$ is assigned to groups $a_1$ and $b_2$.

[4]More details about the group assignment are in [163].

For user $i$, we then follow step 2-4 in Section 4.4.1 to embed the two *group watermarks* and its *individual watermark*. Specifically, by receiving user $i$'s signature $K^i_{priv}(T)$, we first generate three seeds: $\Omega_i$ by combining $K^i_{priv}(T)$ and $K^G$, $\Omega_{a_1}$ by combining $K^{a_1}_{priv}$ and $K^G$, and $\Omega_{b_2}$ by combining $K^{b_2}_{priv}$ and $K^G$, where $K^G$ is graph key for graph $G$. With the two group seeds $\Omega_{a_1}$ and $\Omega_{b_2}$, we generate and embed two non-overlap group watermarks. Then we use user $i$'s individual seed $\Omega_i$ to embed an individual watermark without overlapping with either of the embedded group watermarks. Note that because the group and individual watermarks are generated with different seeds, this hierarchical embedding process does not affect watermark uniqueness.

Under this design, a collusion attack can successfully destroy all the watermarks only if the attacker can perfectly match each individual node, and the majority of the graphs come from different groups. Otherwise, the majority vote on raw edges will preserve the *group watermark*. We can compute the upper bound of the attack success rate by Equation 4.6, *i.e.* the probability that the majority of the graphs obtained by the attacker come from different groups:

$$\lambda(M_a, J) = \left(1 - J \sum_{i=\lceil \frac{M_a+1}{2} \rceil}^{M_a} \binom{M_a}{i} \cdot (\frac{1}{J})^i \cdot (\frac{J-1}{J})^{M_a-i}\right)^2 \tag{4.6}$$

where $M_a$ is the number of watermarked graphs obtained by the attacker and $J$ is the number of groups in each group partition. The above design chose $J = 2$

because it minimizes $\lambda(M_a, J), \forall M_a$. Furthermore, when $M_a$ is odd, $\lambda(M_a, 2) = 0$; and when $M_a$ is even, $\lambda(M_a, 2)$ is at most 0.25 when $M_a = 2$. Note that in equation (4.6) the operation $(.)^2$ is due to the fact that we group the users twice into two different group classes: $a_1, a_2$ and $b_1, b_2$. If we only perform the group partition once (*e.g.* dividing the users into $a_1, a_2$), then $\lambda(2, 2) = 0.5$. In practice we can further reduce $\lambda$ by performing multiple rounds of group division (2 in the above design) and adding more group watermarks.

Note that group watermarks contain much less information than single user watermarks. In fact, the more robust a group watermark, the larger granularity (and less precision) it will provide. Our proposed solution is to extend the system by using additional "dimensions," *e.g.* go beyond the two dimensions of $a$ and $b$ mentioned above. Combining results from multiple dimensions will quickly narrow down the set of potential users responsible for the leak. However, since a colluding attack requires the involvement of multiple leakers, even identifying a single leaker is insufficient. Developing a scheme to reliably detect multiple (ideally all) colluding users is a topic for future work.

### 4.6.3 Impact on Watermark Uniqueness

To improve the robustness of our watermark system, we relax the subgraph matching criteria from exact matching to approximate matching with at most $L$

edge difference. Such relaxation does not affect watermark detectability because it does not change the embedding process. However, it may affect watermark uniqueness, which we will analyze next.

Consider two watermarked graphs $G^{W_i}$ and $G^{W_j}$ that were independently generated for user $i$ and $j$ following the three steps defined in Theorem 1. Let $S^{W_i}$ and $S^{W_j}$ represent the embedded watermark graph in $G^{W_i}$ and $G^{W_j}$, respectively. To examine the watermark uniqueness, we seek to compute the probability that a subgraph in $G^{W_j}$ differs from $S^{W_i}$ by at most $L$ edges.

Our analysis follows a similar structure of Theorem 1's proof. Let $\mathcal{E}_Y$ denote the event where a subgraph of $G^{W_j}$ built on $k$ nodes $Y = \{y_1, y_2, ..., y_k\}$ only differs from $S^{W_i}$ by $\leq L$ edges. Our goal is to calculate the probability of the event $\mathcal{E} = \cup_Y \mathcal{E}_Y$, which is the union on all combinations of $k$ nodes.

We first compute the probability of individual $\mathcal{E}_Y$. Recall that the edges between $\binom{k}{2} - (k-1)$ node pairs in $S^{W_i}$ are generated randomly with probability $\frac{1}{2}$ and are independent of $G^{W_j}$, while the rest $k-1$ edges ($< x_l, x_{l+1} >, l = 1...k-1$) are fixed. Thus we can show that the probability that a subgraph $G^{W_j}[Y]$ differs from $S^{W_i}$ by $h$ edges is upper bounded by $\frac{1}{2}^{e-k+1} \cdot \binom{e}{h}$ where $e = \binom{k}{2}$. Therefore, we can derive the probability of $\mathcal{E}_Y$ as $P(\mathcal{E}_Y) \leq \frac{1}{2}^{e-k+1} \cdot \sum_{h=0}^{L} \binom{e}{h}$. And consequently, we have Equation 4.7

$$P(\mathcal{E}) \leq n^k \cdot \frac{1}{2}^{e-k+1} \cdot \sum_{h=0}^{L} \binom{e}{h} \tag{4.7}$$

where $e = \binom{k}{2}$, $k = (2+\delta)log_2 n$, and $n$ is the node count of $G^{W_j}$.

Next, given the probability of uniqueness $1 - P(\mathcal{E})$, we compute the upper bound on $L$ to ensure $1 - P(\mathcal{E}) \geq 0.99999$ for all the graphs in Table 4.1 except Road graphs, Co-purchasing graphs, and P2P network graphs. Again we set $\delta = 0.3$. The result is listed in Table 4.4, where the maximum limit of $L$ varies between 0 and 12. In general, the larger the graph, the higher the upper bound on $L$.

## 4.7 Experimental Evaluation

We use real network graphs to evaluate the performance of the graph watermarking system in three key metrics: *false positives*, *graph distortion* and *watermark robustness*. Having analytically quantified watermark uniqueness in Section 4.5 and Section 4.6, we focus on examining graph distortion and watermark robustness while ensuring $\leq 0.001\%$ false positive rate. We also study the computational efficiency of the proposed watermark embedding and extraction schemes.

| Graph | $L$ Bound |
|---|---|
| Oregon (1) | 0 |
| Oregon (2) | 1 |
| CAIDA | 1 |
| Email (Enron) | 1 |
| arXiv (Theo. Cit.) | 1 |
| arXiv (Phy. Cit.) | 1 |
| arXiv (Phy.) | 1 |
| arXiv (Astro) | 1 |
| Patent | 2 |
| Slashdot (08/11/06) | 3 |
| Twitter | 3 |
| Slashdot (09/02/16) | 3 |
| Slashdot (09/02/21) | 3 |
| Slashdot (09/02/22) | 3 |
| Brightkite | 3 |
| Russia | 4 |
| Epinions (1) | 4 |
| Google+ | 4 |
| Epinions (2) | 5 |
| Standford | 5 |
| Email (Europe) | 5 |
| Gowalla | 5 |
| BerkStand | 6 |
| DBLP | 7 |
| NorteDame | 7 |
| L.A. | 8 |
| London | 8 |
| Flickr | 8 |
| Wiki | 8 |
| Google | 8 |
| Skitter | 8 |
| Youtube | 9 |
| Pokec | 9 |
| arXiv (Condense) | 11 |
| Livejournal | 12 |

**Table 4.4:** Upper bound of $L$ for the 35 network graphs.

**Experiment Setup.** Given the large number of graph computations per data point, we focus our experiments on two larger network graphs in Table 4.1, *i.e.* the LA regional Facebook graph and the Flickr graph. The two graphs have very different sizes and graph structures. To guarantee $\leq 0.001\%$ false positives, we use $\delta = 0.3$, $k = 45$ for the LA graph, and $\delta = 0.3$, $k = 48$ for the Flickr graph. For our basic design, we generate 1 watermark per graph. For our advanced design, we set $L$ to 8, the degree bucket size to 10, and the NSD similarity threshold to $\theta = 0.75$. For each user, we embed 5 watermarks in its graph, 3 individual watermarks and 2 group watermarks. We chose these settings because they work well in practice. We leave the optimization of these parameters to future work.

Next, we present experimental results on graph distortion, robustness against attacks, and computational efficiency.

### 4.7.1 Graph Distortion from Watermarks

We consider three metrics to measure graph distortion.

- **Modifications to the raw graph.**We count the number of nodes/ edges modified by embedding watermarks. More modifications to the graph introduce higher distortion.
- **dK-2 Deviation.** dK-2 series, *i.e.*, joint degree distributions, are an important graph structural metric [128]. We quantify graph distortion using the nor-

malized Euclidean distance between the dK-2 series of the original and of the watermarked graphs [5]. Larger dK-2 deviation implies higher distortion to the graph structure.

- **Graph metrics with and without watermarks.** We measure the widely used graph metrics before and after the watermarking, including degree distribution, assortativity (AS) [128], clustering coefficient (CC) [128], average path length, and diameter. Large deviation in any of the metrics indicates large distortion.

We have examined the distortion introduced by both the basic and advanced designs. We only show the results of the advanced design because it adds more watermarks and thus leads to higher distortion. For LA and Flickr graphs, we generate 10 different watermarked graphs (using 10 different seeds) and present the average result across these graphs. Because computing shortest paths on the large graphs is highly computational intensive, we compute the average path length and diameter among 1000 random nodes [153].

Table 4.5 shows the percentage of modified nodes/edges by watermarking. Even after embedding 5 watermarks, the modification for both graphs is less than 0.04%, implying little distortion on the watermarked graphs. This is further confirmed by the average dK-2 distances.

[5]The Euclidean distance between dK-2 series is normalized by the number of tuples in the dK-2 series.

| Graph | Nodes (%) | Edges (%) | dK-2 Deviation |
|---|---|---|---|
| Watermarked LA | 0.037% | 0.033% | 0.0008 |
| Watermarked Flickr | 0.014% | 0.019% | 0.0001 |

**Table 4.5:** Percentage of modified nodes/edges after embedding 5 watermarks into a graph and dK-2 Deviation.

We also compare the original and watermarked graphs using 5 graph metrics: AS, CC, degree distribution, average path length, and diameter. Similarly, the metrics remain the same before and after watermarking, and we found no difference between the statistical distributions of each metric in the graphs.

Together, this indicates that embedding watermarks produces negligible impact on graph structure. Thus we believe watermarked graphs can replace the originals in graph applications and produce (near-)identical results.

### 4.7.2 Robustness against Attacks

Next, we study the robustness of the watermarking system under the attacks. For each of the two attacks discussed in Section 4.6.1, we vary the attack strength, repeat each experiment 10 times, and examine the following two metrics:

- **Robustness.** In the single attacker model, the robustness is the ratio of graphs from which we can successfully extract at least one of the 3 individual watermarks. In the collusion attack, in addition to this ratio, we also measure the ratio of graphs where we can extract at least one of the 5 watermarks (3 individual + 2 group watermarks).

- **Cost of the attack.** The normalized distortion on the attacked graphs. It represents the dK-2 deviation between the attacked graphs and the original graph, normalized by that between the "clean" watermarked graphs and the original graph. If the normalized distortion is $> 1$, the attack introduces more distortion than embedding watermarks.

**Results on the Single Attacker Model.** For the single attacker model, we quantify the attack strength by the number of modified edges. The robustness and the cost of the attack are measured as a function of the number of modified edges.

We first evaluate the robustness of the basic watermark. Figure 4.3 and Figure 4.4 show that randomly modifying a small number of edges disrupted the extraction process. For example, in LA, our basic design cannot recover the watermark with 100% probability when we only modify 20 edges. In each case, at least one of the nodes in the watermarks had a modified NSD label (one of its neighbors' degree changed), and it could not be located in the extraction process.

We show the distortion on the attacked graphs separately in Figure 4.5 and Figure 4.6. As expected, the small number of modifications causes small distortions in graph structures. Still in LA, when the robustness is 0, the distortion is

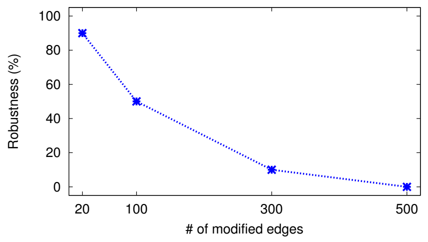


**Figure 4.3:** Robustness of basic design against single attacker model, LA.

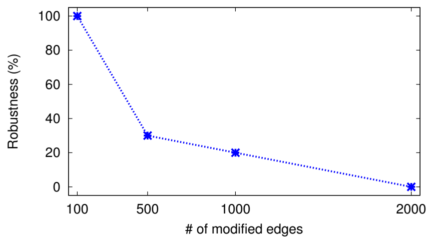


**Figure 4.4:** Robustness of the basic design against single attacker model, Flickr.

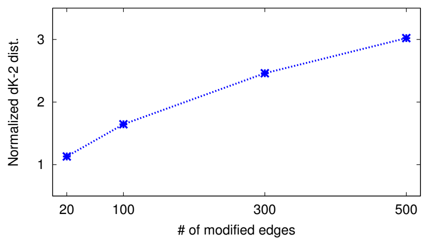


**Figure 4.5:** Distortion caused by single attacker model in the basic design, LA.

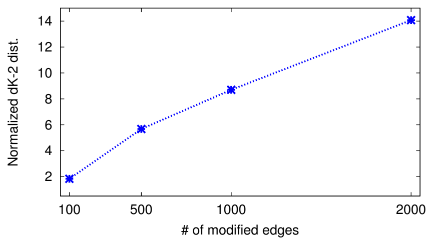


**Figure 4.6:** Distortion caused by single attacker model in the basic design, Flickr.

around 3x more than that the watermarked graphs. Both results show that the basic watermark scheme is easily disrupted by small, single user attacks.

Figure 4.7 and Figure 4.8 show that robustness of the improved scheme decreases with attack strength, since more edges are modified to “destroy” watermarks. Like in Flickr, the system can handle attack strength up to 933K modified edges, which is $>$ 400x stronger than the maximum attack strength in the basic design.

On the other hand, Figure 4.9 and Figure 4.10 show that the cost of these attacks is large. For Flickr, with more than 1.4M modified edges, an attack leads

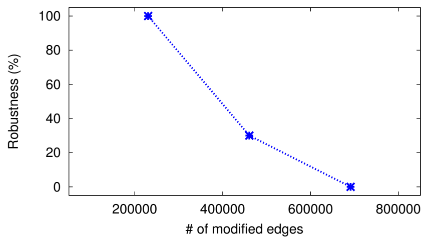


**Figure 4.7:** Robustness of the improved design against single attacker model, LA.

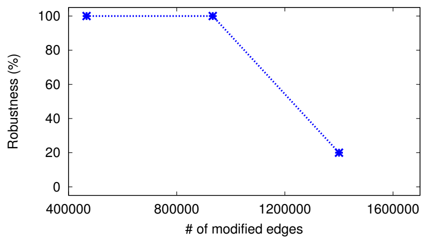


**Figure 4.8:** Robustness of the improved design against single attacker model, Flickr.

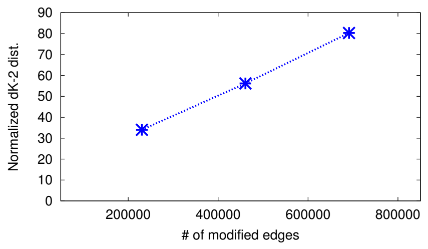


**Figure 4.9:** Distortion caused by single attacker model in the improved design, LA.

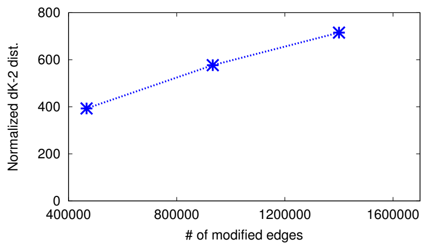


**Figure 4.10:** Distortion caused by single attacker model in the improved design, Flickr.

to 800x more distortions over that caused by embedding 5 watermarks. Our improved watermark is highly robust against single user attacks.

**Results on Collusion Attacks.** To implement the collusion attack described in Section 4.6.1, we first generate 10 watermarked graphs and randomly pick $M_a$ graphs from them as the graphs acquired by the attacker. We vary the number of graphs obtained by the attacker $M_a$ between 2 to 5. For each $M_a$ value we repeat the experiments 10 times and report the average value. Since basic watermarks are easily disrupted by the collusion attack, we focus on the robustness of the improved mechanisms.

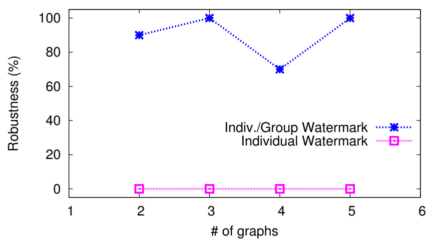


**Figure 4.11:** Robustness of the improved design against collusion attacker model, LA.

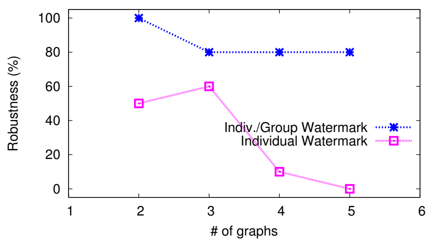


**Figure 4.12:** Robustness of the improved design against collusion attacker model, Flickr.

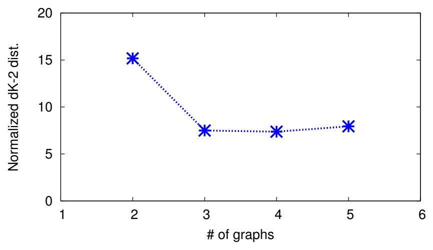


**Figure 4.13:** Distortion caused by collusion attacker model in the improved design, LA.

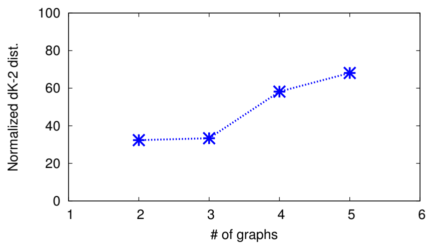


**Figure 4.14:** Distortion caused by collusion attacker model in the improved design, Flickr.

Figure 4.11 and Figure 4.12 show the robustness of the watermarked LA and Flickr graphs against the collusion attack. Figure 4.11 shows that in LA, by applying majority votes on raw edges, the collusion attack can effectively remove all 3 individual watermarks. However, the attack is ineffective in removing both group watermarks: we can extract at least one group watermark in more than 60% of the attacked graphs. Here the robustness values deviate slightly from that projected by Equation (4.6) because we limit the number of statistical sampling to 10 runs. Unlike LA, Figure 4.12 plots that the collusion attack cannot remove all the individual watermarks in Flickr when using 2 or 3 watermarked graphs. This is because the deanonymization method causes a large portion of nodes mismatched in Flickr ( 30% nodes). Finally, Figure 4.13 and Figure 4.14 show that the collusion attacks also introduce larger distortions in graph structure.

These results show that even a powerful collusion attack is ineffective in removing all embedded watermarks. Moreover, the potential inaccuracy of the deanonymization method makes the attack even weaker in removing individual watermarks. Of course, the attackers will eventually succeed in disrupting watermarks if they are willing to modify and sacrifice the utility of the graph. While we provide a robust defense against attackers with low level of tolerance for graph distortion, we hope follow-on work will develop more robust defenses against higher distortion attacks.

### 4.7.3 Computational Efficiency

We measure the efficiency of embedding and extracting watermarks, including the time to select candidates (step 2) and to identify watermarks (step 3). We accelerate the extraction process by parallelizing the key steps across servers. Specifically, in candidate selection, any available server is assigned an unchecked watermark node to find its candidates. In watermark identification, each available server will be assigned to search one watermark from one candidate of watermark

| Graph | Embedding (s) | Basic Extraction | | Improved Extraction | |
|---|---|---|---|---|---|
| | | Single(s) | Parallel (s) | Single (s) | Parallel (s) |
| LA | 40 | 270 | 39 | 310 | 42 |
| Flickr | 80 | 767 | 195 | 776 | 197 |
| Livejournal | 695 | 2568 | 310 | 2605 | 317 |

**Table 4.6:** The efficiency of the watermarking system, including watermark embedding time on one server, the extraction time on one server and the parallel extraction time across 10 servers.

node $x_1$. When a watermark is found or no more candidates are unchecked, the extraction process stops (for that user).

We perform measurements to quantify impact of parallelizing extraction over a cluster. All system parameters are the same as previous tests, except that we embed 1 watermark into a graph. We compare the improved watermark extraction method to the basic extraction method. In addition to Flickr and LA graphs, we also measure the efficiency on Livejournal graph [104], a larger graph with 5.2M nodes, 49M edges. We parallelize watermark extraction across 10 servers, each with 192GB RAM, and report the average times from 10 different watermarked graphs.

Table 4.6 shows that our system is efficient in embedding and extracting watermarks. Embedding one watermark into a graph is very fast, *e.g.* average embedding time for the largest graph, Livejournal, is around 12 minutes. Even using one server to extract watermarks, the computation time is small, *e.g.* 13 minutes in Flickr using both the basic and improved schemes. Time to identify the watermark graph on the candidate subgraphs (step 3) is much less than the time required to find candidates (step 2), which corresponds to 99% of total computation time. Since finding candidates takes $O(kn)$ computational complexity and $k = (2 + \delta)\log_2 n$, the complexity to extract a watermark from a real-world graph is $O(n \log_2 n)$. Here $k$ is node number in the watermark graph and $n$ is nodes in the total graph.

*Second*, we find that distributed extraction produces good speedup, 8 over 10 servers for Livejournal and 7 for LA (for both extraction methods). The speedup for Flickr is only around 4 using both methods, because one of the watermarked graphs takes much longer time than others in finding candidates, 4x longer. Without this outlier, the average parallel extraction time on Flickr is around 2.5 minutes for both methods, 5x faster than using single server.

*Finally*, there is no significant difference between computation time for the two extraction methods.

## 4.8 Summary

In this chapter, we take a first step towards the design and implementation of a robust graph watermarking system. Graph watermarks have the potential to significantly impact the way graphs are shared and tracked. Our work identifies the critical requirements of such a system, and provides an initial design that targets the critical properties of uniqueness, robustness to attacks, and minimal distortion to the graph structure. We also identify key attacks against graph watermarks, and evaluate them against an improved design with additional features for improved robustness under attack. Finally, we show the watermarking system is efficient in both watermark embedding and watermark extraction.

Our evaluation shows that our initial watermarking system modifies very few nodes and edges in a graph, *i.e.* less than 0.04% nodes and edges in a graph with 603K nodes and 7.6M edges. Results also demonstrate extremely low distortion, *i.e.* the watermarked graphs are highly consistent with the original graph in all graph metrics we considered. Empirical tests on several real, large graphs show that our robustness features dramatically improved our resilience against both single and multi-user collusion attacks. Finally, we show that the embedding process and the extraction process are efficient, and the extraction process is easily parallelized over a computing cluster.

While our proposed scheme achieves many of our initial goals, there is significant room for improvement and ongoing work. One focus is developing stronger redundancy schemes to protect against attackers with a greater tolerance for graph distortion, *i.e.* willing to make a greater number of node/edge changes. Another is to develop alternate schemes that can recover more information about multiple attackers in the colluding attack model. We will discuss more in Section 5.3 in Chapter 5.

# Chapter 5

# Conclusion

In this dissertation, we have taken the new opportunities for graph analysis by having access to large traces of graphs, and we have also faced the challenges from scalability, dynamics, and data privacy. Our methodology is data-driven, where in Chapter 2 we address problems that have been first considered, in Chapter 3 we revisit fundamental graph problem, and in Chapter 4 we provide new alternatives for existing problems. We believe that our methodology is general and can be applied to both research in graph analysis, and also various other fields.

In this chapter, we summarize our work on data-driven graph analysis. We also share what we have learnt and lessons through the research process, in the hope that they may provide useful guidance for data-driven analysis or graph analysis. Finally, we discuss future work to conclude the chapter.

## 5.1 Conclusion

Thanks to the proliferation of networks, having access to large traces of graphs is becoming possible. Compare to graphs studied in prior research works, those graphs may be several orders of magnitude larger, have much higher volumes of dynamics, and represent much more private information. They are bringing great opportunities as well as new challenges. In this dissertation, we look at graph problems from a data-driven perspective, where we explore dynamic graph analyzing and modeling in terms of absolute time, we revisit fundamental graph problem and try to improve current solutions, and we provide new alternatives to securely share graph datasets. In all the problems we have looked at, we provide novel solutions and validate their effectiveness with various large-scale graph datasets.

We first seek to analyze and model graph dynamics. Starting from the exploration of self-similarity properties, which is critical of determining how to model network dynamics, our work takes a concrete step towards studying the detailed dynamics of social networks. We focus on “ time-stamped” traces of network growth, i.e., network includes detailed timings of when nodes arrive and edges are created. By performing empirical studies of network dynamics, we find that the edge creation process in social graphs is consistent with self-similarity scaling, once we account for periodic user activity that makes edge creation process non-stationary. We leverage these findings to build a complete model of social network dynamics that combines temporal and spatial components. Specifically, the temporal behavior of our model reflects the coexistence of long-term non-stationary periodic structure, e.g. diurnal or weekly patterns, and properties consistent with self-similarity at shorter time scales. The spatial side accounts for a dynamic graph model that simulates edge creation process driven primarily by existing users, and captures graph densification, shrinking network diameter, and decreasing local clustering. We validate our model against network dynamics in Renren and Facebook datasets, and show that it succeeds in producing desired properties in both temporal patterns and graph structural features.

We then revisit the fundamental graph problem, *i.e.*, link prediction, from an empirical perspective. We use different real traces of large dynamic networks, and take a concrete step towards objectively quantifying the predictive power of today's link prediction algorithms. We implement 18 algorithms, including both metric-based and classification-based approaches. We find that the best metric-based predictors (vary across different networks) perform on par with the most accurate classifier (SVM in all cases), and we derive potential guidelines for choosing metrics based on network structure. We also take a deeper look at current link prediction algorithms for the source of low accuracy, in terms of both structural and temporal aspects. Furthermore, we provide "temporal filters" that can greatly improve prediction accuracy (across different methods and networks) by leveraging knowledge of prior network dynamics, even for predictors that have already integrated temporal information.

Finally, we study graph privacy problems, *i.e.*, how to securely share graph datasets. We take a first step towards the design and implementation of a robust graph watermarking system. Graph watermarks have the potential to significantly impact the way graphs are shared and tracked. Our work identifies the critical requirements of such a system, and provides an initial design that targets the critical properties of uniqueness, robustness to attacks, and minimal distortion to the graph structure. We also identify key attacks against graph watermarks, and evaluate them against an improved design with additional features for improved robustness under attack. Finally, we show the watermarking system is efficient in both watermark embedding and watermark extraction.

## 5.2 Lessons

With the efforts to study graph problems from data-driven perspective, we have accumulated rich experience and learned various lessons. In the following we summarize these lessons.

**Data-Driven Studies is Critical.** Data-driven analysis is a very useful tool in graph analysis, where it helps researchers understand what the real data look like, clear confusions, and provide insights on how we should build models/systems/algorithms. It is especially powerful to correct invalid assumptions with synthetic datasets that are often much smaller and less dynamic.

One prominent example is our revisit to link prediction problem (Chapter 3). As the basis for social recommendations in a wide range of social networks and applications, the link prediction problem is believed by many people to be well solved due to the success of these social sites, and the large number of literature. However, there has been little opportunity to study these link prediction proposals from an empirical perspective. With the access to large traces of social graph growth, we reassess this problem and find current link prediction performance remains poor in absolute terms. We then take a deeper look for the source of low accuracy, and propose an effective filtering mechanism that can greatly improve prediction accuracy based on our insights from data.

Another example is analyzing and modeling social network dynamics (Chapter 2). We start from understanding large-scale social network traces, from the perspective of self-similarity. Using our findings from both temporal patterns and graph structural features, we build a complete model which can more accurately capture important properties in social network dynamics, compared to traditional graph models that based on assumption that are likely invalid for large and dynamic graphs.

The findings and insights from empirical measurements and analysis can often act as guidance when we try to design solution to meet challenges from real world problems. They are powerful tools that are not only applicable in graph analysis, but also useful in other research fields.

**Tradeoffs Based on Priority of Goals.** Designing algorithms and systems for large-scale real graphs often face challenges from tradeoffs among different goals, *e.g.*, accuracy, scalability and robustness. The efforts to achieve all goals usually fail because they often naturally conflict with each other. Then researchers need to prioritize various goals, which is challenging and decisions should be made based on specific application scenario.

For instance, in the design of graph watermark system (Chapter 4), we want the system to have less distortion on graph structure, and also the watermarked graph be robust against potential attacks. There inherently conflicts with each other: strong robustness guarantee indicates more distortion of the graph, which leads to low utility of graph structure, and vice versa. In a graph sharing scenario, we think system robustness is preferred to guarantee graph privacy. So we add a *hierarchical* watermark embedding process on the basic system design to fight agains potential collusion attacks.

Also when we evaluate existing approaches for link prediction, we often face the challenges from scalability versus prediction accuracy. A detailed example is Katz, a widely used metric-based algorithm. Since Katz is proved to be not scalable, we adopt two approximation methods, *i.e.*, Katz$_{lr}$ [6] and Katz$_{sc}$ [137]. Katz$_{lr}$ has less approximation in mechanism, and almost consistently outperforms Katz$_{sc}$. However, it is difficult to scale on larger datasets such as Renren and YouTube. As a result, from the empirical view we think Katz$_{sc}$ is better for larger and denser graphs while Katz$_{lr}$ remains a good choice for smaller and sparser graphs.

Emphasizing on one goal is usually at the cost of some other goals, and to provide a reasonable priority list of goals requires a thorough understanding of both the problem and the scenario. Often this is not an easy task, but it is a must for elegant algorithm/system design.

**Apply Lessons/Wisdom from Other Fields.** Lessons and wisdom from other research fields are often generic to inspire ideas in one's focused area. To make them more helpful, one needs to deeply understand his/her own problem, the specific background, and identify new challenges, if any.

We have learned this lesson from detecting the existence of self-similarity in dynamic graph analysis (Chapter 2). Before we start modeling, we are reminded lessons from traffic modeling, where the discovery of self-similarity in traffic process defines hard limits on how traffic dynamics can be models using traditional means, *e.g.*, Poisson models. Therefore, we start from the measurement of self-similarity property using traditional tools by traffic modeling, *i.e.*, Variance analysis and R/S analysis. However, we find both tools are unable to produce reliable results because of underlying deterministic trends in the process, caused by human behaviors. We then apply a more advanced tool, *i.e.*, wavelet analysis, to fight against such underlying trends. This example shows how lessons from traffic modeling inspired us, and how we need to face specific challenges from social graphs.

In addition, we introduce the idea "watermark" into graph privacy since we see watermarking technique is widely used to protect intellectual property, *e.g.*, in digital watermarking, relational datasets, and there are lots of stories on how they have successfully limited data piracy. Unlike digital or relational datasets watermarking, graph watermarking is quite challenging. We cannot have any labels or identifiers, and can work only on graph structures since any leaked graph can have all metadata stripped. This requires us to deeply understand graph structural properties.

## 5.3 Future Work

In this dissertation, we introduce our efforts for research in graphs from an empirical lens. Looking forward, there are many interesting problems left unexplored in broader graph contexts. In this section, we discuss three potential directions: studying more categories of graphs, securely sharing graphs, and online user behavior analysis.

### 5.3.1 Studying More Categories of Graphs

As fundamental abstraction for networks, there are many different categories of graphs. Many graph models and systems focus on ongoing networks and graph structure only. While today's complex networks continue to develop, there is increasing need for research in more general topics for graphs. Here we target two basic graph types: declining networks, and graphs with meta information.

**Networks in decline.** Unlike ongoing networks, which are growing and the main focus in today's research, networks in decline can be characterized as networks experiencing a sustained reduction in demand [40]. For example, according to [1], in just three years (from 2009 to 2012), the former social network giant MySpace has gone from around 20 million daily visitors to around 2 million. This is not the only example. Social websites like Friendster and Orkut have also experienced great loss. Even Twitter is reported [4] to struggle with growing its user base and face a fairly flat growth in the following several years. In those situation, edges and nodes may disappear and reappear, and the whole networks may have different properties in terms of both graph structures and temporal patterns. This brings us new opportunities and challenges to revisit many graph problems. For example, the measurement, analysis and modeling of dynamic traces in declining networks, prediction for both appearing and disappearing links, etc.

**Graphs with meta information.** Graph meta information may include node-related information like user profiles, and also edge-related information like the type and strength of social links. Those are rich information which are often combined with network structural data in deployed systems/algorithms in real world application. For example, Twitter's user recommendation service WTF ("Who to Follow") [53] combines user profiles and behavioral data, shared interests, common connections and other information sources to largely boost prediction accuracy empirically. Also, there are research focuses on graphs with additional information besides graph topology, such as edge weights [96], node connections on other social networks [106], and link direction [158], they all prove to have impact on graph problems. We plan to consider these factors in future work, *e.g.*, evaluate link prediction in graphs with meta data.

### 5.3.2 Securely Sharing Graphs

Graph sharing is critical to both the research community, where they can have access to more real large graph datasets, and also the industry, where companies can get cooperation chances for better facilities and be free from scandals/lawsuits caused by graph leakage. A variety of solutions have been proposed, ranging from anonymization tools that defend against specific attacks, like *k-anonymization* [91, 168], or more attack-agnostic defense, like graph randomization [56], graph generalization [55], differential privacy [43] and cryptography approaches [28]. However, most of the studies focus on static graphs, which fail to meet challenges from high volume of dynamics from today's network graphs. Here we identify one future direction as securely sharing dynamic graphs. Also, in our dissertation we provide initial work for applying watermarks to graph data, which sets the stage for follow-up work.

**Further exploring graph watermarks.** One direction is to improve robustness against a range of other attacks. For example, in the single attacker model, there might be more disruptive attack versions. The attacker may either release a partial subgraph of the watermarked graph, or merge multiple watermarked graphs. In both cases, this destroys the embedded watermarks and how to defend against them is an interesting topic. Another example is for the collusion attack. Using our proposed hierarchical watermarking technique, we can combine results from multiple dimensions and quickly narrow down the set of potential users responsible for the leak. However, since a colluding attack requires the involvement of multiple leakers, even identifying a single leaker is insufficient. Developing a scheme to reliably detect multiple (ideally all) colluding users is a topic for future work.

Another potential problem is the optimization of parameters in our watermark system. In the advanced design, we need to set degree bucket size, the NSD similarity threshold, edge difference threshold $L$, and number of embedded individual and group watermarks. We hope to give both experimental evaluation for different parameter settings, and also theoretical proof for the accuracy boundary.

**Sharing dynamic graphs.** In dynamic network traces, there might be new privacy breaches, *e.g.*, information of the networks' own formation, and dynamic processes on top of networks like diffusion. We want to ask the following questions:

- *What are the new challenges in dynamic graph sharing ?*
- *How do we quantify graph utility in terms of dynamic graphs ?*
- *Can current approaches be adapted for sharing dynamic graphs ?*
- *Can we propose new approaches by leveraging network dynamics ?*

### 5.3.3 User Studies in Graph Analysis

Many network services are driven by user behaviors, *e.g.*, Facebook has 1.79 Billion users in 2016 [3], and Google search has more than 2.2 Billion users [132] who consume over 3.5 Billion searches per day [2]. To get a deeper understanding of network graphs, especially social network graphs, analyzing online user behavior is indispensable.

Most of current graph solutions are either based on assumptions, or measurement results from networks, and lack insights for user inner motivations and latent behaviors. Even those do focus on user behavior analysis, they often study from an aggregate behavioral level in terms of graph structural changes [42, 124], still deficient in direct understanding of motivations underlying behaviors. For example, the link prediction problem itself is closely related to user motivations to build connections, whereas most of current algorithms are based on graph structural intuitions, or classification methods. Doing user studies like surveys [159] or interviews [26] will help us better understand this problem and ourselves.

# Bibliography

[1] The social drop-off. Pingdom, June 2012. `http://royal.pingdom.com/2012/06/01/the-social-drop-off/`.

[2] Google search statistics. Internet Live Stats, 2014. `http://www.internetlivestats.com/google-search-statistics/`.

[3] Number of monthly active facebook users worldwide as of 3rd quarter 2016 (in millions). Statista, 2016. `https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/`.

[4] Twitter's share of us social network users is dropping. eMarketer, August 2016. `https://www.emarketer.com/Article/Twitters-Share-of-US-Social-Network-Users-Dropping/1014343/`.

[5] Patrice Abry and Darryl Veitch. Wavelet analysis of long-range-dependent traffic. *IEEE TOIT*, 44(1):2–15, 1998.

[6] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *Proc. of ICDMW*, 2009.

[7] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[8] Rakesh Agrawal and Jerry Kiernan. Watermarking relational databases. In *Proc. of VLDB*, 2002.

[9] L. Akoglu and C. Faloutsos. RTG: a recursive realistic graph generator using random typing. In *ECML PKDD*, Sept. 2009.

[10] L. Akoglu, M. McGlohon, and C. Faloutsos. RTM: Laws and a recursive generator for weighted time-evolving graphs. In *Proc. of ICDM*, 2008.

[11] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *Proc. of SDM Workshop on Link Analysis, Counter-terrorism and Security*, 2006.

[12] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[13] Michael Arrington. How "dirty" mp3 files are a back door into cloud drm. TechCrunch, April 2010.

[14] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. 2007.

[15] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proc. of WSDM*, 2011.

[16] Ziv Bar-Yossef and Li-Tal Mashiach. Local approximation of pagerank and reverse pagerank. In *Proc. of CIKM*, pages 279–288, 2008.

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

[18] A.L. Barabâsi, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(3-4):590–614, 2002.

[19] W. Bender, D. Gruhl, N. Morimoto, and Aiguo Lu. Techniques for data hiding. *IBM systems journal*, pages 313–336, 1996.

[20] J. Beran. *Statistics for long-memory processes.* Chapman & Hall/CRC, 1994.

[21] Jan Beran, Robert Sherman, Murad S Taqqu, and Walter Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on communications*, 43(2/3/4):1566–1579, 1995.

[22] Bin Bi and Junghoo Cho. Modeling a retweet network via an adaptive bayesian approach. In *Proc. of WWW*, 2016.

[23] Catherine A Bliss, Morgan R Frank, Christopher M Danforth, and Peter Sheridan Dodds. An evolutionary algorithm approach to link prediction in dynamic social networks. *Journal of Computational Science*, 5(5):750–764, 2014.

[24] A. Blum, T.H.H. Chan, and M.R. Rwebangira. A random-surfer web-graph model. In *Proc. of WAEE and WAAC*, volume 123, 2006.

[25] Anthony Bonato, Noor Hadi, Paul Horn, Paweł Prałat, and Changping Wang. A dynamic model for on-line social networks. *Algorithms and Models for the Web-Graph*, 5427:127–142, 2009.

[26] Julia Ayumi Bopp, Elisa D Mekler, and Klaus Opwis. Negative emotion, positive experience?: Emotionally moving moments in digital games. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 2996–3006. ACM, 2016.

[27] Björn Bringmann, Michele Berlingerio, Francesco Bonchi, and Aristides Gionis. Learning and predicting the evolution of social networks. *IEEE Intelligent Systems*, 25(4):26–35, 2010.

[28] Barbara Carminati, Elena Ferrari, and Andrea Perego. Rule-based access control for social networks. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1734–1744. Springer, 2006.

[29] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proc. of KDD*, 2009.

[30] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *KDD*, 2011.

[31] Yoon-Sik Cho, Greg Ver Steeg, Emilio Ferrara, and Aram Galstyan. Latent space model for multi-modal social data. In *Proc. of WWW*, 2016.

[32] Aaron Clauset, Cristopher Moore, and M.E.J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[33] Christian S. Collberg, Stephen G. Kobourov, Edward Carter, and Clark D. Thomborson. Graph-based approaches to software watermarking. In *WG*, 2003.

[34] Stephen A Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[35] D. Cox. Long-range dependence: A review. *Statistics: An Appraisal*, 1984.

[36] Mark E Crovella and Azer Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM TON*, 1997.

[37] Paulo Ricardo da Silva Soares and R Bastos Cavalcante Prudencio. Time series based link prediction. In *Proc. of WCCI*, 2012.

[38] J. Davidsen, H. Ebel, and S. Bornholdt. Emergence of a small world from local interactions: Modeling acquaintance networks. *Physical Review Letters*, 88(12):128701, 2002.

[39] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. On facebook, most ties are weak. *Communications of the ACM*, 57(11):78–84, 2014.

[40] Christopher Decker. Regulating networks in decline. *Journal of Regulatory Economics*, 49(3):344–370, 2016.

[41] Yuhui Deng, Xiaohua Meng, and Jipeng Zhou. Self-similarity: Behind workload reshaping and prediction. *Future Generation Computer Systems*, 28(2):350–357, 2012.

[42] Derek Doran. On the discovery of social roles in large scale social systems. *Social Network Analysis and Mining*, 5(1):49, 2015.

[43] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[44] Zoltán Eisler, Imre Bartos, and János Kertész. Fluctuation scaling in complex systems: Taylor's law and beyond 1. *Advances in Physics*, 57(1):89–142, 2008.

[45] Michael Fire, Lena Tenenboim, Ofrit Lesser, et al. Link prediction in social networks using computationally efficient topological features. In *SocialCom*, 2011.

[46] Mark W Garrett and Walter Willinger. Analysis, modeling and generation of self-similar vbr video traffic. In *Proc. of SIGCOMM*, 1994.

[47] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959.

[48] Michaela Goetz, Jure Leskovec, Mary McGlohon, and Christos Faloutsos. Modeling blog dynamics. In *ICWSM*, 2009.

[49] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[50] Steven D Gribble, Gurmeet Singh Manku, Drew Roselli, Eric A Brewer, Timothy J Gibson, and Ethan L Miller. Self-similarity in file systems. In *Proc. of SIGMETRICS*, 1998.

[51] Roger Guimera, Leon Danon, Albert Diaz-Guilera, Francesc Giralt, and Alex Arenas. Self-similar community structure in a network of human interactions. *Physical review E*, 68(6):065103, 2003.

[52] Roger Guimerà and Marta Sales-Pardo. Missing and spurious interactions and the reconstruction of complex networks. *Proc. of the National Academy of Sciences*, 106(52):22073–22078, 2009.

[53] Pankaj Gupta, Ashish Goel, Jimmy Lin, et al. Wtf: The who to follow service at twitter. In *Proc. of WWW*, 2013.

[54] Sami Hanhijärvi, Gemma C. Garriga, and Kai Puolamäki. Randomization techniques for graphs. In *SDM*, 2009.

[55] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. 2008.

[56] Michael Hay, Gerome Miklau, David Jensen, Philipp Weis, and Siddharth Srivastava. Anonymizing social networks. Technical Report 07-19, UMass, 2007.

[57] Haibo He and Edwardo Garcia. Learning from imbalanced data. *TKDE*, 21(9):1263–1284, 2009.

[58] Kashmir Hill. Facebook recommended that this psychiatrist's patients friend each other. Fusion.net, August 2016. `http://fusion.net/story/339018/facebook-psychiatrist-privacy-problems/`.

[59] Keven Ho. 41 up-to-date facebook facts and stats. Wishpond, 2014. `http://blog.wishpond.com/post/115675435109/40-up-to-date-facebook-facts-and-stats/`.

[60] P. Holme and B.J. Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[61] Jonathan RM Hosking. Fractional differencing. *Biometrika*, 68(1):165–176, 1981.

[62] Jing Jiang, Christo Wilson, Xiao Wang, Peng Huang, Wenpeng Sha, Yafei Dai, and Ben Y. Zhao. Understanding latent interactions in online social networks. In *Proc. of IMC*, 2010.

[63] Muhammad Kamran et al. A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints. *IEEE TKDE*, 2012.

[64] Thomas Karagiannis, Michalis Faloutsos, and Rudolf H Riedi. Long-range dependence: now you see it, now you don't! In *IEEE Globecom*, 2002.

[65] David Karger and Matthias Ruhl. Find nearest neighbors in growth-restricted metrics. In *Proc. of STOC*, 2002.

[66] Hisashi Kashima and Naoki Abe. A parameterized probabilistic model of network evolution for supervised link prediction. In *ICDM*, 2006.

[67] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[68] Henry Kautz, Bart Selman, and Mehul Shah. Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997.

[69] Jon M Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[70] Valdis E Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[71] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proc. of FOCS*, pages 57–65, 2000.

[72] Jérôme Kunegis. Konect: the koblenz network collection. In *Proc. of WWW Companion*, 2013.

[73] Tommy Landry. How social media has changed us: The good and the bad. Return on Now, September 2014. `http://returnonnow.com/2014/09/how-social-media-has-changed-us-the-good-and-the-bad/`.

[74] Sin-Joo Lee and Sung-Hwan Jung. A survey of watermarking techniques applied to multimedia. In *ISIE*, 2001.

[75] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking*, 2(1):1–15, 1994.

[76] Will E Leland and Daniel V Wilson. High time-resolution measurement and analysis of LAN traffic: Implications for lan interconnection. In *Proc. of INFOCOM*, 1991.

[77] Vincent Leroy, B Barla Cambazoglu, and Francesco Bonchi. Cold start link prediction. In *Proc. of KDD*, 2010.

[78] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *Proc. of KDD*, 2008.

[79] J. Leskovec et al. Statistical properties of community structure in large social and information networks. In *WWW*, 2008.

[80] J. Leskovec and C. Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proc. of ICML*, 2007.

[81] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of KDD*, 2005.

[82] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *TWEB*, 2007.

[83] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *WWW*, 2010.

[84] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of KDD*, 2005.

[85] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 2007.

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

[87] Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012.

[88] Yingjiu Li, Vipin Swarup, and Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE TDSC*, 2(1):34–45, 2005.

[89] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[90] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proc. of KDD*, 2010.

[91] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[92] Qingyun Liu, Shiliang Tang, Xinyi Zhang, Xiaohan Zhao, Ben Y Zhao, and Haitao Zheng. Network growth and link prediction through an empirical lens. In *Proc. of IMC*, 2016.

[93] Qingyun Liu, Xiaohan Zhao, Walter Willinger, Xiao Wang, Ben Y Zhao, and Haitao Zheng. Self-similarity in social network dynamics. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 2(1):5, 2016.

[94] Weiping Liu and Linyuan Lü. Link prediction based on local random walk. *Europhysics Letters*, 89(5):58007, 2010.

[95] Zhen Liu, Qian-Ming Zhang, Linyuan Lü, and Tao Zhou. Link prediction in complex networks: A local naïve bayes model. *Europhysics Letters*, 96(4):48007, 2011.

[96] Linyuan Lü and Tao Zhou. Link prediction in weighted networks: The role of weak ties. *Europhysics Letters*, 89(1):18001, 2010.

[97] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[98] Benoit M Macq and Jean-Jacques Quisquater. Cryptology for digital tv broadcasting. *Proceedings of the IEEE*, pages 944–957, 1995.

[99] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. *ACM SIGCOMM Computer Communication Review*, 36(4):135–146, 2006.

[100] B. B. Mandelbrot and J. W. van Ness. Fractional Brownian motions, fractional noises and applications. *SIAM Review*, 10:422–437, 1968.

[101] M. McGlohon, L. Akoglu, and C. Faloutsos. Weighted graphs and disconnected components: patterns and a generator. In *Proc. of KDD*, 2008.

[102] Alfred J. Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, Inc., 1996.

[103] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Proc. of ECML PKDD*, 2011.

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

[105] Alan Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems.* PhD thesis, Rice University, Department of Computer Science, May 2009.

[106] Arvind Narayanan, Elaine Shi, and Benjamin IP Rubinstein. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *Proc. of IJCNN*, 2011.

[107] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proc. of IEEE S&P*, May 2009.

[108] Arvind Narayanan and Vitaly Shmatikov. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *Proc. of IJCNN*, 2011.

[109] Saket Navlakha, Christos Faloutsos, and Ziv Bar-Joseph. Massexodus: modeling evolving networks in harsh environments. *Data Mining and Knowledge Discovery*, 29(5):1211–1232, 2015.

[110] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.

[111] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proc. of ICML*, 2011.

[112] Ryutarou Ohbuchi, Hiroo Ueda, and Endoh Shuh. Robust watermarking of vector digital maps. In *ICME*, 2002.

[113] Ryutarou Ohbuchi, Hiroo Ueda, and Endoh Shuh. Watermarking 2d vector maps in the mesh-spectral domain. In *Shape Modeling International*, 2003.

[114] Kihong Park and Walter Willinger. *Self-similar network traffic and performance evaluation.* Wiley Online Library, 2000.

[115] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking (ToN)*, 3(3):226–244, 1995.

[116] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[117] Gang Qu and Miodrag Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *ICCAD*, 1998.

[118] Christopher Ratcliff. 23 up-to-date stats and facts about instagram you need to know. Search Engine Watch, April 2016. `https://searchenginewatch.com/2016/04/20/23-stats-and-facts-about-instagram/`.

[119] Matthew J Rattigan, Marc Maier, and David Jensen. Using structure indices for efficient approximation of network properties. In *Proc, of KDD*, 2006.

[120] Rudy Raymond and Hisashi Kashima. Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs. In *Proc. of ECML PKDD*, 2010.

[121] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. In *The Semantic Web-ISWC 2003*, pages 351–368. Springer, 2003.

[122] Alma Riska and Erik Riedel. Long-range dependence at the disk drive level. In *Proc. of QEST*, 2006.

[123] Matthew J. B. Robshaw. MD2, MD4, MD5, SHA and other hash functions. Technical Report TR-101, RSA Laboratories, 1995. v. 4.0.

[124] Daniel M Romero, Brian Uzzi, and Jon Kleinberg. Social networks under stress. In *Proceedings of the 25th International Conference on World Wide*

*Web*, pages 9–20. International World Wide Web Conferences Steering Committee, 2016.

[125] J.J.K. O Ruanaidh, W.J. Dowling, and F.M. Boland. Phase watermarking of digital images. In *ICIP*, 1996.

[126] Diego Rybski, Sergey V Buldyrev, Shlomo Havlin, Fredrik Liljeros, and Hernán A Makse. Scaling laws of human interaction activity. *PNAS*, 106(31):12640–12645, 2009.

[127] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B.Y. Zhao. Measurement-calibrated graph models for social network experiments. 2010.

[128] Alessandra Sala, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y. Zhao. Measurement-calibrated graph models for social network experiments. 2010.

[129] Alessandra Sala, Xiaohan Zhao, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. Sharing graphs using differentially private graph models. In *IMC*, 2011.

[130] Purnamrita Sarkar, Deepayan Chakrabarti, and Michael Jordan. Nonparametric link prediction in dynamic networks. *Proc. of ICML*, 2012.

[131] David Savage, Xiuzhen Zhang, Xinghuo Yu, Pauline Chou, and Qingmai Wang. Anomaly detection in online social networks. *Social Networks*, 39:62–70, 2014.

[132] Marc Schenker. New study suggests google plus' active usage may be much lower. Digital Trends, April 2015. `http://www.digitaltrends.com/social-media/study-suggests-far-fewer-active-google-plus-users-than-google-claims/`.

[133] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. Understanding online social network usage from a network perspective. In *Proc. of IMC*, 2009.

[134] Herb Scribner. 10 ways the internet has changed your life. Deseret News, August 2014. `http://www.deseretnews.com/article/865608397/10-ways-the-Internet-has-changed-your-life.html/`.

[135] Umang Sharan and Jennifer Neville. Exploiting time-varying relationships in statistical relational models. In *Proc. of WebKDD Workshop on Web mining and social network analysis*, 2007.

[136] Chaoming Song, Shlomo Havlin, and Hernan A Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005.

[137] H.H. Song, T.W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proc. of IMC*, 2009.

[138] Walton Steve. Information authentication for a slippery new age. *Dr. Dobbs Journal*, pages 18–26, 1995.

[139] Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference AND International Workshop Present Day Trends of Innovations*, 2012.

[140] M. S. Taqqu and J. Levy. Using renewal processes to generate long-range dependence and high variability. *Dependence in Probability and Statistics*, pages 73– 89.

[141] Murad S Taqqu, Vadim Teverovsky, and Walter Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(4):785–798, 1995.

[142] Ben Taskar, Ming-Fai Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *Proc. of NIPS*, 2003.

[143] R. Toivonen, J.P. Onnela, J. Saramaki, J. Hyvonen, and K. Kaski. A model for social networks. *Physica A: Statistical and Theoretical Physics*, 371(2):851–860, 2006.

[144] Tomasz Tylenda, Ralitsa Angelova, and Srikanta Bedathur. Towards time-aware link prediction in evolving social networks. In *Proc. of SNA-KDD*, 2009.

[145] A. Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E*, 67(5):056104, 2003.

[146] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. In *Information Hiding*, 2001.

[147] B. Viswanath, A. Mislove, M. Cha, and K.P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.

[148] C. Wang, V. Satuluri, and S. Parthasarathy. Local probabilistic models for link prediction. In *Proc. of ICDM*, 2007.

[149] Mengzhi Wang, Tara Madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *Proc. of ICDE*, 2002.

[150] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[151] Walter Willinger, Vern Paxson, and Murad S Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. *A practical guide to heavy tails: statistical techniques and applications*, 23:27–53, 1998.

[152] Walter Willinger, Murad S Taqqu, Robert Sherman, and Daniel V Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on Networking (ToN)*, 5(1):71–86, 1997.

[153] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao. User interactions in social networks and their implications. In *Proc. of EuroSys*, 2009.

[154] Greg Wolfe, Jennifer L. Wong, and Miodrag Potkonjak. Watermarking graph partitioning solutions. In *DAC*, 2001.

[155] Hongke Xia and Xiang Hu. Fbm: A flexible random walk based generative model for social network. *Open Cybernetics & Systemics Journal*, 9(1):280–287, 2015.

[156] Xiang-Gen Xia, C. G. Boncelet, and G. R. Arce. A multiresolution watermark for digital images. In *ICIP*, 1997.

[157] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, page 3. ACM, 2012.

[158] Dawei Yin, Liangjie Hong, and Brian D Davison. Structural link analysis and prediction in microblogs. In *Proc. of CIKM*, 2011.

[159] Ming Yin, Mary L Gray, Siddharth Suri, and Jennifer Wortman Vaughan. The communication network within the crowd. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1293–1303. International World Wide Web Conferences Steering Committee, 2016.

[160] Xiaowei Ying and Xintao Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, 2008.

[161] X. Zhao, A. Sala, C. Wilson, X. Wang, S. Gaito, H. Zheng, and B.Y. Zhao. Multi-scale dynamics in a massive online social network. In *Proc. of IMC*, 2012.

[162] Xiaohan Zhao, Qingyun Liu, Haitao Zheng, and Ben Y Zhao. Towards graph watermarks. In *Proc. of COSN*, 2015.

[163] Xiaohan Zhao, Qingyun Liu, Lin Zhou, Haitao Zheng, and Ben Y. Zhao. Graph watermarks. *Arxiv preprint arXiv:1506.00022*, 2015.

[164] Bin Zhou et al. Preserving privacy in social networks against neighborhood attacks. In *Proc. of ICDE*, 2008.

[165] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *European Physical Journal B*, 71(4):623–630, 2009.

[166] William Zhu, Clark Thomborson, and Fei-Yue Wang. A survey of software watermarking. In *ISI*. 2005.

[167] Matteo Zignani, Sabrina Gaito, Gian Paolo Rossi, Xiaohan Zhao, Haitao Zheng, and Ben Y. Zhao. Link and triadic closure delay: Temporal metrics for social network dynamics. In *Proc. of ICWSM*, 2014.

[168] Lei Zou, Lei Chen, and M. Tamer Özsu. K-automorphism: A general framework for privacy preserving network publication. In *VLDB*, 2009.