Anomaly Detection in Large Social Graphs

Neil Shah

CMU-CS-17-123

October 2017

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

This research was sponsored by the National Science Foundation under grant numbers IIS-1217559, CNS-1314632, IIS-1408924, and DGE-1252522, Yahoo, Lawrence Livermore National Laboratory under grant numbers B604246 and B605884, BBNT Solutions under grant number 13765224, PNC, and the Siemens Foundation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: fraud detection, anomaly detection, graph mining, data mining, machine learning, user behavior, clustering, network science, social science, social network analysis, plain graphs, dynamic graphs, rich graphs, unsupervised learning, scalability, interpretability, fBox, TimeCrunch, FLOCK, EdgeCentric, M3A, DeltaCon-Attr, ConDeNSe

To my parents, who have always supported me and entertained my desire to learn. And to Jess, for understanding and encouraging me, especially when it has been difficult. I love you both.

Abstract

Given the ever-growing prevalence of online social services, leveraging massive datasets has become an increasingly important challenge for businesses and end-users alike. Online services capture a wealth of information about user behavior and platform interactions, such as *who-follows-whom* relationships in social networks and *who-rates-what-and-when* relationships in e-commerce networks. Since many of these services rely on data-driven algorithms to recommend content to their users, authenticity of user behavior is paramount to success. But given anonymity on the internet, how do we know which users and actions are real or not? This thesis focuses on this problem and introduces new techniques to effectively and efficiently discern anomalous and fraudulent behavior in online social graphs. Specifically, we work on three thrusts: plain graphs, dynamic graphs and rich graphs.

Firstly, we focus on *plain graphs*, in which only static connectivity information is known. We detail several proposed algorithms spanning the topics of spectral fraud detection in a single graph, blame attribution between graph snapshots, and structurally diverse graph summarization. Our fBox algorithm in [SBGF14] identifies link fraudsters in social networks with over 93% precision and identifies hundreds of thousands of fake accounts, many of which were yet unsuspended.

Next, we broaden our scope to *dynamic graphs*, in which we leverage connectivity information over a span of time. Many online interactions are timestamped, and thus time and interarrival time between user actions are powerful features which can be used to discern abnormal behavior. We describe multiple relevant works which describe how to identify and summarize anomalous temporal graph structures, model interarrival time patterns in user queries to find anomalous search behavior, and identify "group" anomalies comprising of users acting in lockstep. Our FLOCK approach in [Sha17] is the first to tackle the viewbot problem on livestreaming platforms, and finds astroturfed broadcasts and views with over 90% precision and near-perfect recall.

Lastly, we expand our reach to *rich graphs*, in which connectivity information is supplemented by other attributes, such as time, rating, number of messages sent, etc. Rich graphs are common in practice, as online services routinely track many aspects of user behavior to gain multifaceted insights. Multimodal views of data are useful in identifying various types of anomalies in different subspaces. To this end, we propose works which focus on ranking anomalies in edge-attributed graphs, and characterizing multimodality of online link fraud. Our EdgeCentric approach in [SBH+16] uncovers rating patterns in e-commerce datasets and pinpoints fake reviewers with 87% precision at Flipkart.

The techniques described in this thesis span various disciplines including data mining, machine learning, network and social sciences and information theory and are practically applicable to a number of real-world fraud and general anomaly detection scenarios. They are carefully designed to attain high precision and recall in practice and scale to massive datasets, including social networks, telecommunication networks, e-commerce and collaboration networks with up to *millions* of nodes and *billions* of edges.

Acknowledgements

There are *so* many people who have made it possible for me to stand where I am today. My first thanks goes out to my advisor and dear friend, Christos Faloutsos, without whom I know for a fact that my PhD life could not have been as colorful, enjoyable and productive as it was. Christos has done far more for me than he knows, and certainly more than I can express. Nonetheless, I will try to do him justice. The first time I met Christos, I was blown away by his excitement about his and his students' work. Five minutes into the first meeting, I already knew that I wanted to work with him. I am ever-grateful to Christos for taking me as a student, who despite having a research background, knew next to nothing about data mining and was in all senses of the word, clueless. Over the years, he has been integral in shaping me into the researcher that I have become. From the late nights helping me work on paper deadlines, to the countless times I have been met with a smile and snacks in his office, to his uplifting spirit that has pulled me out of mental slumps more than once, and his kindness and authentic interest in my mental and physical well-being, I will miss working with Christos as his student. I would be so fortunate to work with him again, and look forward to crossing paths hopefully a thousand times in our future careers.

I also want to thank my other thesis committee members, Roni Rosenfeld, Jaime Carbonell and Jiawei Han. Their questions and feedback during my proposal were greatly instructive and helpful in the direction of this work. I would especially like to thank Jiawei, whose comments gave me a number of interesting future directions to take up.

I would be utterly remiss in not acknowledging my undergraduate research advisor, Nagiza Samatova, for my successes both leading up to graduate school and long afterwards. Nagiza was the first academic I had the fortune of working with, as a high-school junior throughout my undergraduate years at NCSU. She was also the first researcher I burdened with my cluelessness, as a fresh and overly cocky high-school student. To this day, I cannot fathom why she took a chance in mentoring and advising me. But thanks to her inexplicable choice, for which I am supremely grateful, I learned the fundamentals of research. Nagiza taught me how to clearly articulate scientific ideas, survey literature, interpret cryptic messages left by overly-articulate academics in their papers, evaluate my approaches, and finally how to write a research paper. There was nothing that could have prepared me for when she sat me down to edit the first attempt at a paper introduction I had written, and emptied her red pen all over my drivel. While this may sound harsh, Nagiza has always had my best intentions in mind, and without her incalculable efforts in instructing, mentoring and generally helping me become a better researcher, I could not have possibly had the opportunities that led me to CMU.

In addition to my primary academic research mentors, I have been very fortunate in being mentored by, working with, and befriending several fantastic folks external to CMU. I would like to thank Brian Gallagher, my first external collaborator upon starting at CMU, for mentoring me and bouncing around ideas on our weekly phone calls. I am grateful to him for offering me my first summer internship at LLNL, where I had an awesome time both in and out of the lab (special thanks to the curly fries at the lab, which were always perfectly seasoned and made it possible to do good research). I am grateful to Yang Song, for his career advice and the opportunity he extended me to do interesting work off my beaten path at MSR. I am also greatly appreciative of my mentors at Twitch, Brad Schumitsch and Drew Harry, who went above and beyond their mandated responsibilities to mentor me, have invaluable discussions about my career path, and show me how much fun working in industry could be. I am also grateful to Leman Akoglu (who has since come back to CMU!), Tim Weninger, and Tina Eliassi-Rad, who have all given me excellent advice, shared in interesting research discussions and been great friends to me throughout graduate school.

I am especially thankful for the colleagues and friends I have gotten to know in the CMU databases group – of particular note are Alex Beutel and Vagelis Papalexakis. These two have always been there for me when I needed mental (or technical) support, and their friendship and mentorship is evident in the kind of avant-garde researcher I am growing to be. In many ways, they have been, and continue to be my role models. I look fondly on our time and words shared at CMU, on many conference trips, and even after their graduation. I was also lucky to have befriended and worked with Danai Koutra, who has constantly amazed me with her productivity and has more than once been a big sister to me. I would also like to thank my academic siblings Hemank Lamba, Bryan Hooi, Kijung Shin, Hyun-Ah Song, Dhivya Eswaran, and Miguel Araujo for our countless interactions, collaborations, discussions and excursions, and for making my time at CMU memorable. These acknowledgements would not be complete without mentioning Marilyn Walgora and Ann Stetser, both of whom are hands-down the best administrative assistants that I know. Without them, literally everything would have been more difficult for me during graduate school. I want to especially thank Marilyn for having Vagelis and I over for Thanksgiving dinner when we weren't with our families, and cooking us a delicious meal which neither of us will forget. An extended thanks also goes out to all the fantastic visitors at CMU I have had the fortune of meeting: Stephan and Nikou Gunnemann, Bruno Ribeiro, Alceu Ferraz-Costa, Yasuko Matsubara, Yasushi Sakurai, Meng Jiang, Rohan Kumar, Shenghua Liu and more.

I am also grateful for the numerous friends I made at CMU outside of research. I especially enjoyed the times I spent with my awesome past officemates: Guru Guruganesh, Sahil Singla and Joy Arulraj. From many long hours of working in silence, wracking our brains over homework and project assignments, failed software installations, table tennis games, career discussions, life ups and downs, lunches and coffee breaks, they made my office life fun every day. In addition to a great office life, I also had a great home life: thanks to Joseph Tassarotti for being an awesome roommate who for 4 years shared in my successes and miseries along the winding road of the PhD. I also want to thank Jay-Yoon Lee, Anthony Platanios, Nika Haghtalab, Zack Coker, Sid Jain, Christian Kroer, David Kurokawa, Vittorio Perera, Ashique Khudabukhsh and many more for their friendship and the experiences I shared with them throughout graduate school.

Despite spending most of my time at CMU, I have made many great friends outside the school as well. I would like to thank Srijan Kumar (my brother in arms against sockpuppets and malicious behavior) and Ivan Brugere (my good friend and companion across multiple internships and conferences). I will fondly remember my time at MSR by the thought-provoking and often hilarious conversations with Julia Kiseleva and Kyle Williams. My experience at Twitch would have not been nearly as lively without frequent discussions and lunches with Soucindar Hoche, Wenjia Zhao and my short-time desk-neighbor June Dershewitz. I also want to mention Da-Cheng Juan, Yike Liu, Priya Govindan, Cyrus Hall, Sal Aguinaga, Joel Pfeiffer, Xiang Ren, Mike O'Brien, Mohit Kumar, Disha Makhija, Amin Mantrach, Jennifer Neville, Tanya Berger-Wolf, Branden Fitelson, B. Aditya Prakash, Jure Leskovec, Polo Chau, Tim Pan, Jimeng Sun, Spiros Papadimitriou and many more with whom I have had riveting conversations and lots of fun over the years. I am fortunate to know and have spent time with you all.

Anyone who knows me can attest to how important the gym is to me. Ryan and Dana Celli of Celli's Fitness Center have set up a great facility to train at, and greatly contributed in making me a stronger person both physically and mentally over the 4 years I spent in Pittsburgh. I have spent countless hours at Celli's blowing off steam. Many thanks to Matt Hartman, Morgan Simon, Julien Comte, Kyle Meeder, Vanessa Ferla, Andy Bartosiewicz, Matt Austerberry and Rio and Kairo (the best Dobermans I know... rest in peace, Rio) for being familiar faces there, and constantly encouraging me over the years in both gym and life ventures.

I cannot forget about the friends who have stuck by me from before CMU. I am grateful to Sriram Lakshminarasimhan and Eric Schendel for mentoring, befriending and generally dealing with me in my high-school and undergraduate years at Nagiza's lab, and for checking up on my well-being throughout graduate school. I would also like to acknowledge Suraj Shah, Parth Gandhi and Rushi Shah, who have been there for nearly two decades as my close friends – they are like brothers to me. I have grown up with their influences, and am who I am as a result. Further thanks to my good friends from NCSU, Shashank Srinivas and Alex Higgins for their friendship in the undergraduate years and afterwards. Their words have cheered me up more times than I can count, and brought light to some of my darker days.

Extra special thanks and all my love to my partner(-in-crime) and closest friend, Jess Feldman. Her love, encouragement, understanding, and belief in my potential has been essential to my success (and conducive to my well-being) these last years. I am also appreciative towards her family for the kindness they have shown me and support they have given. Last but not least, I am most grateful to my parents – I owe them all my successes. I am proud of the way they have raised me, and the person I have become as a result. Their countless sacrifices and unconditional love has ensured my ability to pursue my own dreams. For that, I cannot thank them enough. I love them dearly.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Thesis Overview and Contributions

Mining large datasets has become an especially notable focal point in computer science research in recent years due to the ever-increasing scale and complexity of online systems – an estimated 2.5 exabytes of new data is generated every day[1] from commercial transactions, social networks, system log data, electronic sensors and more. This heavily motivates the development of effective and scalable approaches for extracting patterns from large data sources. In reality, many data sources can be construed as graphs, which represent interactions between entities such as humans or computers. Graphs enable modelling of complex phenomena including interactions between users (who-follows-whom on Twitter), product impressions (who-rates-what on Amazon) and email traffic flow (who-emails-whom in a corporate network). While graph analysis can give us insights on *common* interaction patterns such as how users follow each other on a social network, or how they rate and review products on an e-commerce network, it can also be a powerful tool for identifying *uncommon* anomalous behavior including fake or fraudulent nodes and links used to artificially boost popularity, sockpuppets aiming to spread rumors and misinformation, and other types of disingenuous cyberattackers. These attackers typically hide behind the guise of anonymity in order to skew public perception of entities for monetary or political benefit. Discerning them in a time-effective manner and successfully preventing and mitigating their actions is therefore an important task. However, there are many associated challenges with this task which involve usefully leveraging the right kinds of information: How can we identify anomalous behavior when only structural graph connectivity is known? How can we additionally leverage temporal information for the same? Furthermore, how can we also integrate more complex and multifaceted features to enable a holistic approach to anomaly detection in graphs? This thesis tackles such problems. Our main thesis statement is as follows:

[1]https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

Modeling complex facets of social graph interactions such as interconnectivity, temporality and contextual information can enhance our ability to discern malicious and otherwise abnormal behavior online.

In this thesis, we develop new means for anomaly detection in graphs in three major thrusts: mining *plain graphs*, *dynamic graphs* and *rich graphs*. Each of these tasks involves leveraging different facets of information which reflect various components of user behavior, such as who they choose to connect with on a social network, how often they make connections and take actions, and which (if any) contextually abnormal features they exhibit.

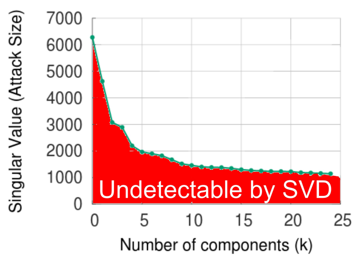
1.1.1 Plain Graphs

In some cases, practitioners and data scientists are equipped with only a *plain graph*, which describes structural connectivity information between objects in a static snapshot, and must use this limited amount of information to find anomalous behavior. This task is common in industrial datasets made available to academics (in which rich features and personally identifiable information are often stripped for privacy or security reasons), as well as in certain types of graphs which inherently represent simple phenomena or are the result of limited observational power. Moreover, plain graph analysis provides the first stepping stone for more sophisticated analysis on more complex graphs. In this thesis, we detail a number of approaches which can intelligently utilize this sparse information to identify both abnormal individual and group connectivity patterns.

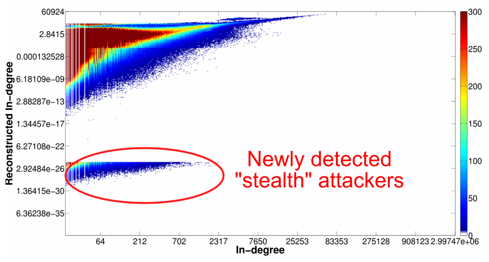
Link fraud, in which botnet operators create and use hordes of sockpuppet accounts to inflate customer popularity (Twitter follows, Facebook page-likes, etc.) is one of the most damaging types of online fraud, due to its ability to skew public perception and hinder recommendation algorithms. Traditional approaches to link fraud detection aim to find large synchronized groups of malicious behavior through matrix factorization algorithms, like Singular Value Decomposition, applied to the social graph. In Chapter 3, we theoretically and empirically demonstrate the limits of such algorithms in detecting stealthy attacks that manifest below the factorization rank (see Figure 1.1a). We further propose the fBox algorithm which finds many previously uncaught link fraudsters on Twitter. Figures 1.1b and 1.1c give a visual depiction of fBox's reconstruction error plot and one example fraudulent account that was found by our approach.

Often times, abnormal and suspicious behavior is difficult to detect within a single graph snapshot. In such cases, a commonly encountered problem is identifying the individual culprits responsible for drastic change between two snapshots: for example, failure points given two snapshots of a computer network, or users with abnormal e-mail activity given two snapshots of an e-mail network. In Chapter 4, we introduce DeltaCon-Attr, a fast and effective approach for attributing blame to nodes and edges. DeltaCon-Attr measures a node's culpability according to the change in *influence* with respect to neighboring nodes and thus identifies culprits that match with human intuition.

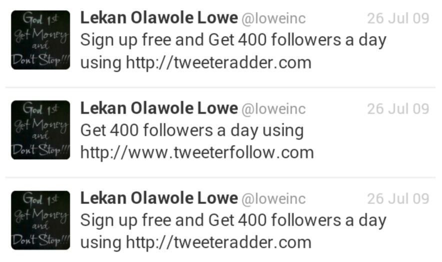
Complementing the previous chapters which focus on individual anomalies, Chapter 5 shifts focus to identifying interesting group-wise structures in a plain graph by means of summarization.



(a) Traditional SVD provably misses fraud attacks below a certain size



(b) fBox's Spectral Reconstruction Map catches suspicious fraudsters that SVD misses on Twitter



Lekan Olawole Lowe @loweinc 26 Jul 09
Sign up free and Get 400 followers a day using http://tweeteradder.com

Lekan Olawole Lowe @loweinc 26 Jul 09
Get 400 followers a day using http://www.tweeterfollow.com

Lekan Olawole Lowe @loweinc 26 Jul 09
Sign up free and Get 400 followers a day using http://tweeteradder.com

(c) Real Twitter profile of a link-fraudster which fBox catches

Figure 1.1: **fBox catches stealthy link fraudsters.** (a) shows the minimum scale of "dense-subgraph" fraud attacks that traditional methods like SVD can catch based on model rank – undetectable attack sizes are shaded in red. Notice that even at a considerable factorization rank $k = 25$, SVD still misses a 960×960 attack. (b) shows how fBox separates stealthy link fraudsters (circled red) from honest users using reconstruction error. (c) shows spammy Tweets from one such previously uncaught fraudster who had been operating for years, unbeknownst to Twitter.

Our proposed ConDeNSe approach harnesses the power of traditional graph clustering and decomposition approaches paired with an information theoretic paradigm, and efficiently produces concise, compressed summaries of large graphs using structures interesting to practitioners (such as cliques, stars, bipartite cores) which often represent unusual node interactions.

Contributions

- **Detecting Stealthy Link-fraud Attacks:** In Chapter 3, we characterize the limits of traditional fraud detection approaches and propose the complementary fBox algorithm. Our algorithm attains *93% precision* on Twitter users and finds *tens of thousands* of previously undetected suspicious accounts.
- **Cross-Graph Blame Attribution:** In Chapter 4, we introduce DeltaCon-Attr for pinpointing nodes and edges most responsible for change between graph snapshots. Our blame attribution approach obeys intuitive principles which competitors violate and demonstrates practical effectiveness on real e-mail network data.
- **Improved Summarization for Large Graphs:** In Chapter 5, we propose the ConDeNSe framework for summarizing static graphs, which creates approximate, concise and non-redundant descriptions of large graphs. ConDeNSe produces only *10%* as many structures as competitors with a *30-50%* lower compression rate.

Impact

- fBox (Chapter 3) was featured in keynotes at WWW 2014, SIAM CSE 2015, ICML 2016, and HotSoS 2016.

- fBox (Chapter 3) was included in the *Multimedia Databases and Data Mining* (15-826) course at Carnegie Mellon University and at a KDD 2015 tutorial on *Graph-Based User Behavior Modeling*.

1.1.2 Dynamic Graphs

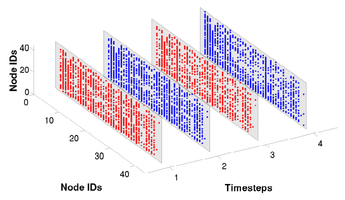
In domains in which graph objects represent users, such as in e-commerce and social networks, interactions are often represented as a *dynamic graph* which changes over time. This stems from the reality that humans themselves behave differently over time. For example, a who-rates-what e-commerce graph typically grows over time, as users purchase and rate more products. Similarly, a social network also changes over time as users follow and unfollow each other depending on changing interests. Temporal information provides a powerful signal for discerning between authentic and inauthentic user behavior (for example, a user who rates products consistently 5 seconds apart is likely not behaving normally, and is likely scripted). Our work provides insights into modeling and incorporating dynamism in graphs to identify abnormal behavior.

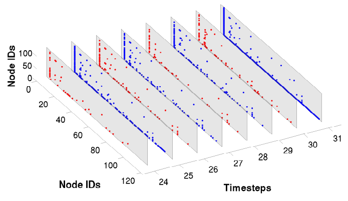
In Chapter 6, we study a similar graph summarization problem as in Chapter 5, but in a dynamic context. Understanding and decomposing the general behavioral makeup of dynamic graphs is central to both graph understanding and more complex tasks such as individual link prediction and group behavior forecasting. We propose TimeCrunch, an information theoretically grounded approach to concisely summarize large dynamic graphs using a lexicon of common, interesting *temporal structures* like bursty cliques, constant stars, etc. TimeCrunch is able to extract coherent temporal structures which match human intuition across a variety of real datasets and can also be used for graph compression. Figure 1.2 shows several such examples across a variety of real datasets.

Given that temporal recurrence patterns are themselves complex and poorly understood, we narrow our focus in the next two chapters to primarily leveraging temporal behavior with limited focus on connectivity.

Search queries are one of the most common online actions taken by users everyday. The conventional assumption for queries is that they are submitted independently and thus follow a constant rate as in a Poisson process. In Chapter 7, we argue that this assumption is false, and show that users' query interarrival times are in fact bimodal, corresponding to in-session and take-off behaviors reflecting whether the user is in the midst of a search session, or just starting one. We propose the Camel-Log distribution to model bimodal user interarrival times and show that it achieves better model fit than competitors. Furthermore, we propose Meta-Click to jointly model Camel-Log parameters and demonstrate how it can be used to detect anomalous querying behavior and search bots.

Lastly, we shift our focus to link fraud in the livestreaming domain. Livestreaming platforms provide channels for streamers to freely broadcast their content to viewers across the world, incentivizing popularity and high viewership. Unfortunately, this incentive drives the viewbotting business, in which bot providers give streamers access to tools to generate fake views to their own channels. In Chapter 8, we provide the first characterization of the viewbotting problem in livestreaming, and propose the multi-level FLOCK algorithm for identifying fake views. FLOCK


(a) 40 users of Yahoo! Messenger forming a *constant near clique* with unusually high 55% density, over 4 weeks in April 2008.


(b) 111 callers in a large phonecall network, forming a *periodic star*, over the last week of December 2007 – note the heavy activity on holidays

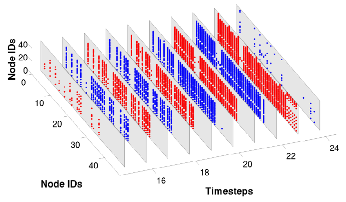

(c) 43 collaborating biotechnology authors forming a *ranged near clique* in the DBLP network, jointly publishing through 2005-2012.

Figure 1.2: **TimeCrunch finds coherent, interpretable temporal structures**. We show the reordered subgraph adjacency matrices, over the timesteps of interest, each outlined in gray; edges are plotted in alternating red and blue, for discernibility.

builds a behavioral model of temporal viewership metrics across many past broadcasts, and accurately identifies viewbotted broadcasts and constituent fake views by penalizing deviance from model fit.

Contributions

- **Interpretable Dynamic Graph Summarization:** In Chapter 6, we tackle the graph summarization problem in dynamic graphs. Our proposed TimeCrunch approach is the first capable of extracting general temporal graph patterns, and demonstrates successful pattern extraction and graph compression on several real datasets.
- **Modeling Interarrival Times in Web Searches:** In Chapter 7, we propose a new user-level model for interarrival times, and a group-level anomaly detection model to find users with irregular behavior. Our M3A approach better models *78%* of AOL search users over the next best competitor and detects abnormally active search bots.
- **Catching Fake Views in Livestreaming Platforms:** In Chapter 8, we introduce FLOCK, the first approach for combating fake views on livestreaming platforms. FLOCK achieves *98% precision* in detecting viewbotted broadcasts and over *90% precision* and *95% recall* in detecting fake views in synthetic attacks.

Impact

- TimeCrunch (Chapter 6) was used for pattern discovery in gene-gene interaction networks in order to model tumor progression [WPK+17].
- TimeCrunch (Chapter 6) was included in the *Mining Large-Scale Graph Data* (EECS 598) course at the University of Michigan and the *Topics in Data Mining* (CS69000-DM1) course at Purdue University. Furthermore, TimeCrunch was featured in the 2016 Carnegie Mellon University CyLab Partners Conference, the 2017 Army Research Lab Network Science bootcamp and an SDM 2017 tutorial on *Summarizing Large-Scale Graph Data: Algorithms, Applications and Open Challenges.*

- The recording of our TimeCrunch (Chapter 6) KDD 2015 presentation has been viewed over 100 times on YouTube.
- M3A (Chapter 7) is used in *production* at *Google* to identify search spammers and abnormal search behaviors.
- FLOCK (Chapter 8) is used in *production* at *Twitch.tv* to combat view astroturfing.

1.1.3 Rich Graphs

Most online services track very rich information about their users to improve user experience and the quality of user recommendations for products and other users. For example, e-commerce networks often maintain detailed information about the types of products their users like, transaction cost, time spent viewing product pages, and product ratings and review text. Similarly, social networks have information on how often users view each others pages, exchange messages, endorse each others' profile statuses and so on. In the presence of *rich graphs* which capture several types of details about interactions, we tackle the commensurate challenges of integrating a multitude of signals to inform our detection algorithms as well as identifying different types of anomalies.

Conventional unsupervised graph-based anomaly detection approaches have often focused on identifying densely connected subgraphs, or individuals with abnormal connectivity behavior given a plain graph. However, leveraging rich information for unsupervised anomaly detection is still an open problem. In Chapter 9, we devise an information theoretically motivated metric for ranking the abnormality of nodes based on their adjacent edge-attributes. Our metric offers intuitive scores, in terms of *bits* required to explain a node's behavior in terms of edge-attributes such as ratings, timestamps, etc. We demonstrate strong performance in detecting fraudulent raters on e-commerce platforms.

When some domain knowledge is available, fraud detection methods often focus on a *single* fraudulent pattern to catch – some approaches look for users forming cliques, others for users forming bipartite cores, and still others focus on users with a specific abnormality in attribute values. In Chapter 10, we use honeypots to empirically study fraudster connectivity and attribute behavior in the rich Twitter ecosystem, and characterize the habits of *several different types* of link fraud regimes. We further propose novel discriminative features based on first-order follower attributes for detecting malicious accounts and show near-ideal classification performance on our ground-truth dataset. Figure 1.3 shows differences across network structure and attribute behavior of genuine and (two discovered modes of) fraudulent users.

Contributions

- **Ranking Anomalies in Edge-Attributed Graphs:** In Chapter 9, we devise a general abnormality ranking function for nodes in graphs which leverages categorical and numerical edge attributes. Our EdgeCentric ranking algorithm is scalable and attains over *90% precision* in detecting fraud over top-ranked users on Flipkart ratings data.
- **Characterizing the Multimodality of Link Fraud:** In Chapter 10, we identify and characterize the *multiple* major regimes of link fraud on Twitter using honeypots, and devise

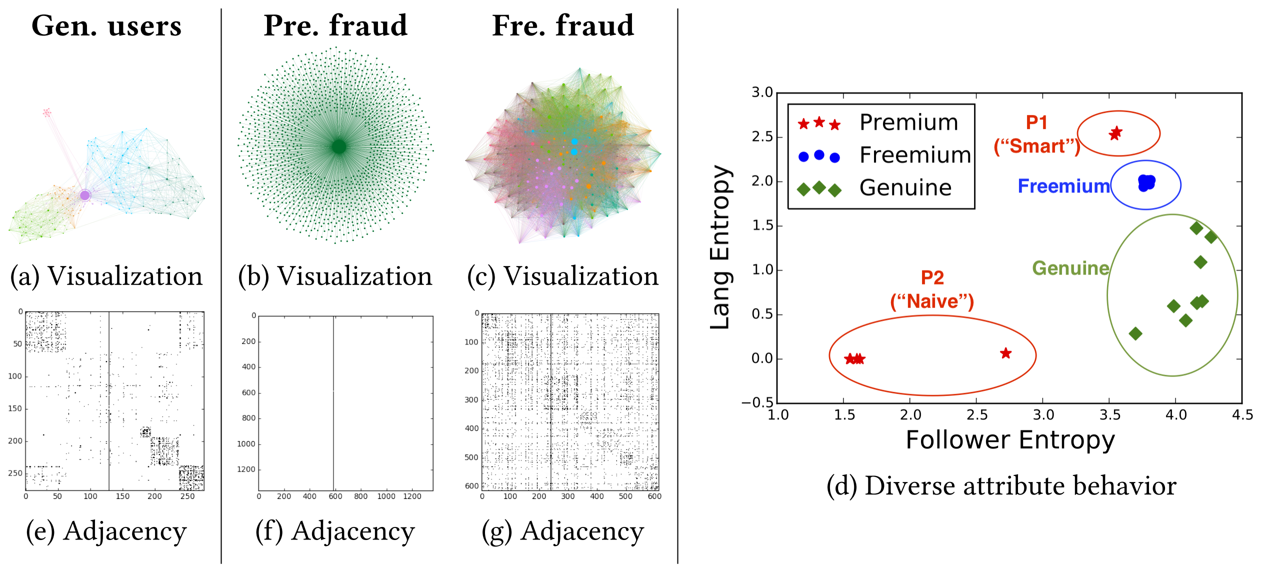


Figure 1.3: **The many faces of fraud:** We discover multiple link fraud behaviors, dubbed freemium (Fre) and premium (Pre), which have different local network and attribute features compared to genuine users. Nodes are colored by modularity class, and sized proportional to in-degree in (a)-(c). The associated, reordered adjacency matrices are shown in (e)-(g) – the vertical line in each spyplot indicates the the central node. Notice the block community structure in genuine followers compared to the star structure for premium and near-clique structure for freemium followers. (d) shows differences in attribute entropy over the various behaviors, showing how fraud patterns skew these distributions away from genuine ones.

discriminative entropy-based features which attain near-ideal classification performance in discerning genuine from fraudulent users.

Impact

- EdgeCentric (Chapter 9) is used in *production* at *Flipkart* to identify ratings fraud.
- EdgeCentric (Chapter 9) and our work on link fraud multimodality (Chapter 10) were both presented at the 2017 Army Research Lab Network Science panel.

1.2 Thesis Organization

We now describe the organization for the rest of the document. Chapter 2 provides basic background information for commonly used ideas and techniques in this thesis, which may be helpful for the uninformed or forgetful reader. The next 3 parts (I, II and III) correspond to works in the *plain*, *dynamic* and *rich* graph themes respectively. Each chapter begins with a short summary of the motivation and direction of the contained content.

Part I contains works for identifying suspicious link behavior (Chapter 3), cross-graph blame attribution (Chapter 4), and reducing large graphs to small supergraphs (Chapter 5).

Part II encompasses works focusing on interpretable dynamic graph summarization (Chapter 6), modeling interarrival times in web searches (Chapter 7), and astroturfing in livestreaming platforms (Chapter 8).

Part III is composed of two works which tackle ranking anomalies in edge-attributed graphs (Chapter 9) and characterizing multifacetedness of link fraud behaviors (Chapter 10).

Finally, Part IV concludes (Chapter 11) and discusses avenues for future work (Chapter 12).

Chapter 2

Background

Below, we provide brief refreshers/precursors for some key concepts discussed in this thesis. Further necessary details for understanding these concepts contextually will be presented as needed throughout the document.

2.1 Graphs

Graphs are the core data structure used to describe social interactions in this dissertation. They are also called networks – we use these terms interchangeably. Graphs are commonly used to model connectivity between entities, and typically consist of *nodes* (also known as *vertices*) connected to each other with *edges*. Nodes represent entities, like humans and computers. Edges represent actions taken between nodes – for example, edges on the Twitter social network between users indicate that one user follows another. Graphs are typically denoted G, with the set of nodes $\mathcal{V}$ and edges $\mathcal{E}$. Below, we discuss several relevant definitions that are used often.

Bipartite Graph: The number of *parts* of a graph refers to the number of independent sets into which the nodes can be split. That is, every edge in the graph must touch different parts, and no edges exist between nodes in the same part. A *bipartite* graph is thus a graph in which there are two such parts $\mathcal{V}_1$ and $\mathcal{V}_2$. Formally, $\mathcal{E} \subseteq \{(u,v)|u \in \mathcal{V}_1, v \in \mathcal{V}_2\}$. Some examples of real-world bipartite graphs include the *user-likes-page* graph on Facebook, and the *user-watches-video* graph on Youtube.

Undirected Graph: In many cases, edges merely represent a mutually established relationship between two nodes. In such cases, we say that a graph is *undirected*, by which we mean that the edges lack directionality. Formally, $\mathcal{E} \subseteq \{(u,v)|u \in \mathcal{V}, v \in \mathcal{V}\}$, and $(u,v) \in \mathcal{E} \leftrightarrow (v,u) \in \mathcal{E}$. Common examples of real-world undirected graphs include mutual friendship networks such as Facebook, and bipartite (and more generally k-partite) networks in which directionality is unimportant.

Directed Graph: In some cases, edges represent a *directed* relationship between two nodes, in which the existence of an edge (u,v) does not necessitate the existence of (v,u). Formally,

$(u, v) \in \mathcal{E} \not\Leftrightarrow (v, u) \in \mathcal{E}$. Some well-known examples of these graphs are friendship networks which do not mandate friend "approval," such as Instagram and Twitter's *who-follows-whom* graph, or a mobile provider's *who-calls-whom* graph.

Multigraph: Above, we have limited our discussions to *simple* graphs, in which an edge between two nodes either exists, or does not. In some cases, it is appropriate to define a *multigraph* in which edge multiplicity is not boolean, but rather some value in $\mathbb{N}^0$. Formally, we denote a multigraph with $G(\mathcal{V}, \mathcal{E}, m)$ where $m : \mathcal{E} \to \{(u, v)|u \in \mathcal{V}, v \in \mathcal{V}\}$ (m maps each edge to the pair of nodes it connects). Multigraphs are commonly useful for representing counts of interactions, such as the *user-messages-user* graph on Facebook, where each edge between two users represents a different message.

Hypergraph: When dealing with rich, multimodal data, we are often concerned with complex interactions that are not restricted to two nodes. A *hypergraph* generalizes the concept of a graph to allow for edges between more than two nodes. Formally, we can denote a hypergraph in the same fashion as a graph, but with the edge set $\mathcal{E}$ containing arbitrarily-sized subsets reflecting many-node interactions. In this work, we are primarily interested in "limited" hypergraphs with only k-node edges that reflect connectivity between k parts, or independent sets of nodes $\mathcal{V}_1 \ldots \mathcal{V}_k$. Formally, $\mathcal{E} \subseteq \{(u_1 \ldots u_k)|u_1 \in \mathcal{V}_1 \ldots u_k \in \mathcal{V}_k\}$. This structure is useful when representing multi-aspect (especially temporal) data such as *user-watches-video-at-time*, or *user-messages-user-at-time*.

Subgraph: Sometimes, we are interested in the edges between a particular subset of nodes $\mathcal{V}_1 \subseteq \mathcal{V}$ rather than those of the whole graph. We denote the induced *subgraph* as $G_1(\mathcal{V}_1, \mathcal{E}_1)$ where $\mathcal{E}_1$ is the set of relevant edges $\{(u, v) \in \mathcal{E}|u \in \mathcal{V}_1, v \in \mathcal{V}_\infty\}$. Subgraph analysis is used when studying small portions of a network – one of the most famous types of subgraphs is called the *egonet*, which is defined on the nodes adjacent to a central node, or *ego*.

Matrix Representation: For mathematical convenience, graphs are commonly described using their *adjacency matrices*. Formally, a simple graph G can be described using a $|\mathcal{V}| \times |\mathcal{V}|$ adjacency matrix $\mathbf{A}$, where $(u, v) \in \mathcal{E} \leftrightarrow \mathbf{A}_{u,v} = 1$ and $(u, v) \notin \mathcal{E} \leftrightarrow \mathbf{A}_{u,v} = 0$. That is, the corresponding matrix entry is only nonzero if u and v are incident. If G is a multigraph, the value of $\mathbf{A}_{u,v}$ instead reflects the count of edges between u and v. If G is a k-partite hypergraph, as described above, $\mathbf{A}$ reflects a k-dimensional *tensor* (multi-dimensional matrix), and $\mathbf{A}$ can be subscripted with k indices to reflect k-node interactions. For example, the value $\mathbf{A}_{u,v,w}$ in a *user-watches-video-at-time* hypergraph might reflect the number of times user u watched video v at time w.

2.2 Clustering

Clustering is a common data mining task with rich prior literature – [AR13] provides an excellent and thorough overview. The high-level goal of clustering algorithms is to group or partition a set of n items $\{i_1 \ldots i_n\}$ into some number of k clusters. In some scenarios, each item is meant to be associated only to a single cluster – this is known as *hard clustering*. For example, in unsupervised anomaly detection, we might be interested in classifying a sample as either part of a "good" cluster or a "bad" cluster, without the possibility for a sample to be mostly good but somewhat bad. Alternatively, sometimes *soft clustering* (also known as *fuzzy clustering*) is more appropriate, in which each item can be associated with one or more clusters. For example, if we tried to cluster individuals based on their left-leaning and right-leaning political beliefs, we may find it more appropriate to allow each individual to partially belong to the left-leaning cluster and partially to the right-leaning cluster, due to differences in political alignment (though some individuals may still fully belong to the left-leaning and right-leaning clusters with no involvement in the other). Below, we briefly discuss some preliminaries regarding graph and feature clustering approaches which are discussed in this thesis.

Graph Clustering: In some applications, we are interested in clustering *nodes* in a graph. For example, if we know that a social graph is composed of 2 underlying communities or friend groups, graph clustering can reveal the approximate group memberships by grouping nodes according to an appropriate criterion. In fact, many algorithms exist for exactly this goal of *community detection* [MV13], and use a variety of similarity metrics as criteria for grouping. Some examples of commonly used criteria for grouping nodes include modularity [PSS+10], graph cut [KK00], compression cost [CPMF04], and random walk-based similarity [PARS14]. METIS [KK00], Girvan-Newman [NG04], Louvain [BGLL08] and spectral clustering [AKY99] are some of the most commonly used algorithms in this space. It is useful to mention that some of these are hard clustering approaches which involve unique assignment of each node to a single partition, whereas others are soft clustering approaches which allow a node to be involved in numerous clusters. We give further background on these as necessary throughout the thesis.

Feature Clustering: Many times, we are interested in grouping items with arbitrary *features*, or *attributes* without any graph-based context. These problems are often motivated by the need to find similar items to a given item, or to infer underlying structure of item types. For example, if we were interested in recommending similar Twitter users to each other, we might describe each user u with a d-dimensional vector which serves as a compressed representation of his/her interests, and cluster users together who have a high similarity (or equivalently low distance) to one another. The most widely used algorithms for general feature clustering are the traditional k-means algorithm [Llo82], and it's adaptations including minibatch k-means [Scu10] and k-means++ [AV07]. These algorithms are simple and interpretable, well-studied and reasonably scalable in practice. They traditionally aim to minimize the squared-error loss

$$\sum_{i=1}^{k} \sum_{u \in \mathcal{C}_i} |u - w_i|^2$$

where $\mathcal{C}_i$ denotes the set of points in the i^{th} of k clusters, and w_i denotes the centroid of the i^{th} cluster. Typically, the objective is minimized using an alternating optimization approach which involves fixing centroids and updating cluster assignments, and vice versa (note that this is a hard clustering). It can be shown this procedure converges to a local minimum. One of the biggest challenges in the use of k-means type approaches is their requirement of the user-specified parameter k. Often, practitioners are not aware what the "right" value of k should be, and resort to heuristics such as the rule-of-thumb, the elbow method, Silhouette coefficient and more [KM13]. Unfortunately, these methods typically require manual inspection and testing over many k-means executions with varying k, which is often not viable in practice on large datasets. Alternative approaches such as X-means [PMO00] and G-means [HEO03] use more principled heuristics (information theoretic and statistical test-based, respectively) and iterative splitting procedures to learn the value of k automatically. It has been shown that such approaches tend to arrive at qualitatively good clusterings and well-approximate the "right" number of clusters in appropriate scenarios. A similar analogue exists in the soft clustering space, one of the most popular approaches involves modeling data as the mixture of k Gaussian distributions (known as Gaussian Mixture Modeling [XZC01]) and learning parameters for the Gaussian means and covariances using expectation-maximization [CG10] – in such cases, [Ras00, BJ+06] provide non-parametric alternatives for also learning k.

2.3 Singular Value Decomposition

Occasionally, it is of interest to represent a high-dimensional matrix with an "approximate" lower dimensional one. This is a common task in feature extraction [PWWB09] and high-dimensional clustering [SFŚ+04], and as we shall see later in the thesis, for anomaly detection. One of the most common approaches for this task is known as the *singular value decomposition* (SVD), which in its fullest form factorizes a general $m \times n$ matrix $\mathbf{A}$ into three matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

The full-rank decomposition implies that that $\mathbf{U}$ is of size $m \times k$, $\mathbf{\Sigma}$ of size $k \times k$, and $\mathbf{V}$ of size $n \times k$, where k is equal to the rank of $\mathbf{A}$. $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, and known as the left and right singular vectors of $\mathbf{A}$, respectively. $\mathbf{\Sigma}$ is a diagonal matrix which contains the non-negative real singular values of $\mathbf{A}$, which are also the square-roots of the eigenvalues of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$.

When $k < rank(\mathbf{A})$, we have a *truncated SVD*, where

$$\mathbf{A}_k = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

and $\mathbf{A}_k$ is a k-rank approximation (generally known as a *low-rank approximation*) of $\mathbf{A}$. The SVD has many useful theoretical properties, but the most famous one is a consequence of the Eckart-Young-Mirsky theorem [EY36], which states that $\| \mathbf{A} - \mathbf{A}_k \|_F^2 \leq \| \mathbf{A} - \mathbf{B} \|_F^2$ for any k-rank matrix $\mathbf{B}$. This states that the k-rank approximation $\mathbf{A}_k$ obtained via SVD on $\mathbf{A}$ is the *best* low-rank approximation in terms of difference in Frobenius norm to the original matrix $\mathbf{A}$. This optimality, combined with efficient methods to compute the SVD, make it a top-contender for the dimensionality reduction task.

The SVD can also be interpreted as representing the original matrix $\mathbf{A}$ as a weighted sum of rank-1 matrices:

$$\mathbf{A} = \sum_{i=1}^{rank(\mathbf{A})} \sigma_i u_i {v_i}^T$$

where u_i is the i^{th} column of $\mathbf{U}$, v_i is the i^{th} column of $\mathbf{V}$, and σ_i is the i^{th} singular value ("weight"). The resemblance to the original $\mathbf{A}$ increases with increasing number of terms (*latent* *factors*) of the sum, and rank of the intermediate result. The above equality is exact when the decomposition is full-rank, and approximate when it is truncated.

In practice, these latent factors have shown to be useful in extracting communities, or dense blocks from the input matrix [PSS+10, JCB+14a]. Intuitively, items that project very strongly to a latent factor are often connected. SVD can help us interpret meaning in large, and complex matrices. For example, if we consider a matrix of *users-purchase-items*, we might see that the first latent factor corresponds to sports enthusiasts who purchase sporting goods, the second corresponds to gamers who purchase video games and consoles, and so on. In this way, SVD tries to "explain" as much of the matrix as possible given a budget on matrix rank.

2.4 Minimum Description Length

There are several ways in which one can determine whether one model is a better fit to some data than another. Model selection principles generally aim to trade-off between the fit and complexity of the model. *Model fit* refers to the extent to which the model is able to accurately describe a certain set of data, whereas *model complexity* refers to the cost of expressing the model itself (i.e. in terms of the number of parameters). Various commonly used selection principles include the Akaike Information Criterion, Bayesian Information Criterion [Kuh04], and the *Minimum Description Length* (MDL) principle [Ris78].

In this thesis, we use MDL with some frequency. MDL is driven by the insight is that "any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally" [Grü05]. In fact, it is a practical version of *Kolmogorov complexity*, which refers to the shortest possible description of a string in some fixed language (and is also unfortunately uncomputable) [Nan10]. MDL provides an information theoretic perspective on model selection, in which both a model and the model's description of some data are seen as a *sequence of bits*. Informally, the model which gives the shortest possible description is the best model. Formally, the principle states that given a family of models $\mathcal{M}$ (typically, a parametric family), the best model $M \in \mathcal{M}$ for data D is given by

$$\arg\min_{M \in \mathcal{M}} L(M) + L(D|M)$$

where $L(M)$ refers to the cost in bits of encoding the model (complexity), and $L(D|M)$ is the cost in bits of encoding the data given the model (fit). Together, the two terms establish a basis for losslessly reconstructing D – MDL uses a lossless compression paradigm to enforce fairness in comparing models.

Part I

Mining Plain Graphs

Based on content published in [SBGF14].

Chapter 3

fBox: Identifying Suspicious Link Behavior

Exploiting subspace project error to identify abnormally connected link fraudsters.

How can we discern whether the links or connections of a user in an online social network are honestly or dishonestly incentivized? Online link fraud hurts the authenticity of social platforms and paints a deceptive picture about the true popularity of users and products. Targeting dense subgraphs in the social graph is a common approach to identifying link fraud, but techniques like spectral decomposition which are commonly employed for this task are unfortunately biased to detecting only large attacks given the limited expressivity of low-rank representations. In this chapter, we take an adversarial approach to show the detection limits of such approaches and propose fBox, a complementary algorithm which targets a separate class of smaller scale “stealth attacks.” Our method is highly scalable and shows high efficacy in pinpointing many tens of thousands of suspicious accounts on the Twitter platform.

3.1 Introduction

In an online network, how can we distinguish honest users from deceptive ones? Since many online services rely on machine learning algorithms to recommend relevant content to their users, it is crucial to their performance that user feedback be legitimate and indicative of true interests. “Fake” links via the use of sockpuppet/bot accounts can enable arbitrary (frequently spammy or malicious) users and products of varying nature seem credible and popular, thus degrading the online experience of users. Unsurprisingly, numerous sites such as `buy1000followers.co`, `boostlikes.com` and `buyamazonreviews.com` exist to provide services such as fake

	Detects stealth attacks	Camouflage resistant	Offers visualization
SpokEn	✘	✘	✔
Spectral subspace plotting	✘	✔	✔
CopyCatch	✘	✔	✘
OddBall	✘	✘	✘
fBox	✔	✔	✔

Table 3.1: Qualitative comparison between fBox and other link fraud detection methods.

Twitter followers, Facebook page-likes and Amazon product reviews for typically just a few dollars per one-thousand fake links.

Here we focus exactly on the link-fraud problem. We take an adversarial approach to illustrate when and how current methods fail to detect fraudsters and design a new complementary algorithm, fBox, to spot attackers who evade these state-of-the-art techniques. Figure 3.1 showcases several suspicious accounts spotted by fBox– we elaborate on three of them, marked using the triangle, square and star glyphs. All three are identified as outliers in the fBox Spectral Reconstruction Map (SRM) shown in Figure 3.1b. The corresponding Twitter profiles are shown in Figure 3.1c, and further manual inspection shows that all three accounts exhibit suspicious behavior:

- triangle: it has only 2 tweets but over 1000 followers
- square: it is part of a 50-clique with suspicious names
- star: it posts tweets advertising a link fraud service

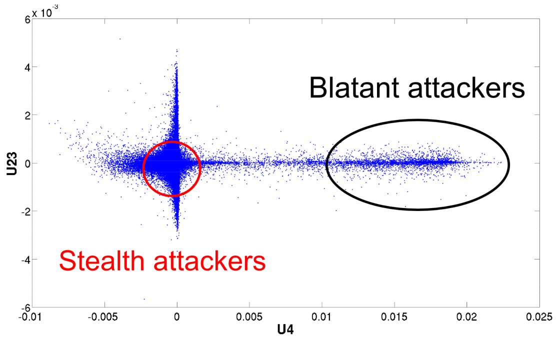
Our main contributions are the following:

1. **Theoretical analysis:** We prove limitations of the detection range of spectral-based methods.
2. **fBox algorithm:** We introduce fBox, a *scalable* method that *boxes-in* attackers, since it spots small-scale, stealth attacks which evade spectral methods.
3. **Effectiveness on real data:** We apply fBox to a real, 41.7 *million* node, 1.5 *billion* edge Twitter who-follows-whom social graph from 2010 and identify many still-active accounts with suspicious follower/followee links, spammy Tweets and otherwise strange behavior.

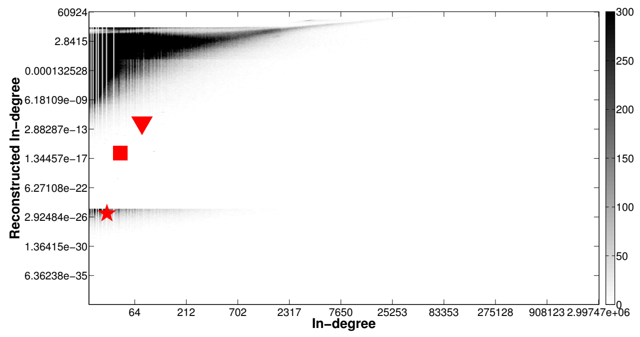
Reproducibility: Our code is available at `http://www.cs.cmu.edu/~neilshah/code/`. The Twitter dataset is also publicly available as cited in [KLPM10].

3.2 Background and Related Work

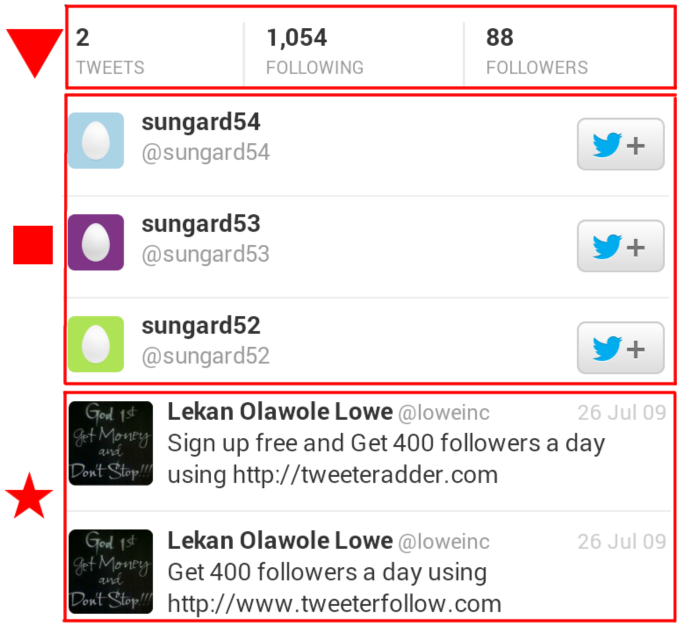
We begin by reviewing in detail several of the current state-of-the-art methods in web fraud and spam detection. Table 3.1 shows a qualitative comparison between various link fraud detection methods.



(a) Spectral subspace plot



(b) Proposed ISRM



2 TWEETS | 1,054 FOLLOWING | 88 FOLLOWERS

sungard54 @sungard54

sungard53 @sungard53

sungard52 @sungard52

Lekan Olawole Lowe @loweinc 26 Jul 09
Sign up free and Get 400 followers a day using http://tweeteradder.com

Lekan Olawole Lowe @loweinc 26 Jul 09
Get 400 followers a day using http://www.tweeterfollow.com

(c) Suspicious accounts

Figure 3.1: **fBox catches stealth attacks which are missed by spectral methods.** (a) shows a spectral subspace plots on the Twitter social graph which identifies blatant attacks (circled in black) but ignores stealth attackers (circled in red, near the origin). (b) portrays how the proposed fBox ISRM (In-link Spectral Reconstruction Map) can describe these users by their *reconstruction degree* and identifies several with improbably poor reconstruction. (c) shows their suspicious profiles with matching glyphs (see text for details).

3.2.1 Spectral methods

We classify techniques that analyze the latent factors produced in graph-based eigenanalysis or matrix/tensor decomposition as spectral methods. These algorithms seek to find patterns in the graph decompositions to extract coherent groups of users or objects. Prakash et al's work on the EigenSpokes pattern [PSS+10] and Jiang et al's work on spectral subspaces of social networks [JCB+14b] are two such approaches that we will primarily focus on and which have been employed on real datasets to detect suspicious link behavior. [YWB11] uses a similar analysis of spectral patterns, but focuses on random link attacks (RLAs), which have different properties than link fraud and therefore produce different patterns.

These works utilize the Singular Value Decomposition (SVD) of the input graph's adjacency matrix in order to group similar users and objects based on their projections. Recall that the SVD of a $u \times o$ matrix $\mathbf{A}$ is defined as $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V^T}$, where $\mathbf{U}$ and $\mathbf{V}$ are $u \times u$ and $o \times o$ matrices respectively containing the left and right singular vectors, and $\mathbf{\Sigma}$ is a $u \times o$ diagonal matrix containing the singular values of $\mathbf{A}$. Both papers note the presence of unusual patterns (axis-aligned spokes, tilting rays, pearl-like clusters, etc.) when plotting the singular vectors $\mathbf{U_i}$ and $\mathbf{U_j}$ for some $i, j \leq k$, where k is the SVD decomposition rank, indicative of suspicious lockstep behavior between similar users. The authors use these patterns to chip out communities of similar users from input graphs.

Beyond directly searching for suspicious behavior, spectral methods have been used for a variety of applications. [MWP+14] builds off the above work to use tensor decomposition for network intrusion detection. [BMFS14] proposes a robust collaborative filtering model that clusters latent parameters to limit the impact of fraudulent ratings from potential adversaries. [NJW01] and [HYJT08] propose using eigenvectors of graph decompositions for graph partitioning and community detection.

Although spectral methods have shown promise in finding large communities and blatantly suspicious behavior in online networks, they are universally vulnerable given knowledge of the decomposition rank k used in a given implementation. All techniques operating on large graphs use such a parameter in practical implementations given that matrix decompositions are very computationally expensive [KMPF14]. Previous spectral methods have generally chosen small values of $k < 100$ for purposes of computability. As we will show in Section 3.3, knowledge of k or the associated singular value threshold (inferrable from sample datasets online) enables an intelligent adversary to engineer attacks to fall below the detection threshold.

3.2.2 Graph-traversal based methods

A wide variety of algorithms have been proposed to directly traverse the graph to find or stop suspicious behavior. [SMR08] offers a random walk algorithm for detecting RLAs. [GVK+12] proposes a PageRank-like approach for penalizing promiscuous users on Twitter, but is unfortunately only shown to be effective in detecting already caught spammers rather than detecting new ones. [PCWF07] uses belief propagation to find near-bipartite cores of attackers on eBay.

However, most similar in application is Beutel et al's CopyCatch algorithm to find suspicious lockstep behavior in Facebook Page Likes [BXG+13]. CopyCatch is a clustering method that seeks to find densely connected groups in noisy data through restricted graph traversal, motivated with the intuition of fraud taking the form of naïvely created bipartite cores in the input graph. The algorithm uses local search in the graph to find dense temporally-coherent near-bipartite cores (TNBCs) given attack size, time window and link density parameters.

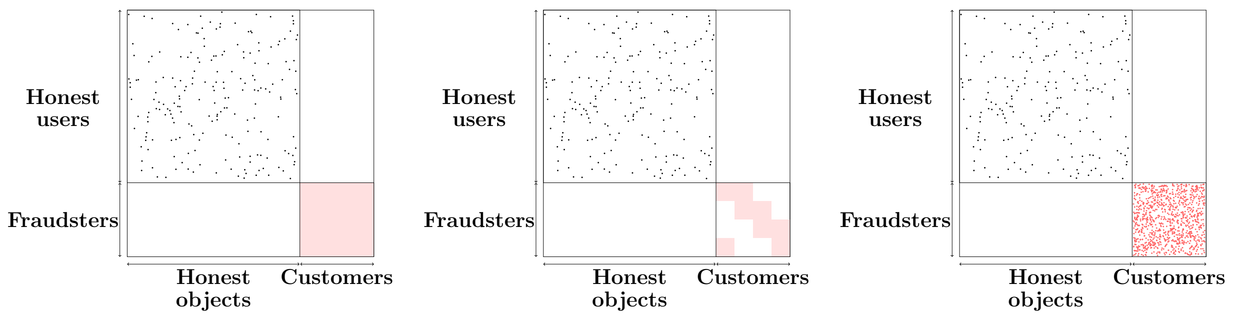
Clustering methods like CopyCatch are able to avoid detection problems caused by camouflage (connections created by attackers to legitimate pages or people for the purposes of appearing like honest users) given that they ignore such links if the attacker is party to any TNBC. However, identifying the appropriate "minimal attack" parameters is nontrivial. Non-conservative parameter estimates will result in many uncaught attackers whereas excessively conservative estimates will result in numerous false positives. From an adversarial point-of-view, we argue that the cost of incurring false positives and troubling honest users is likely not worth the added benefit of catching an increased number of attackers after some point. Therefore, an alternative approach to catch stealth attacks falling below chosen thresholds is necessary.

3.2.3 Feature-based methods

Spam and fraud detection has classically been framed as a feature-based classification problem, e.g. based on the words in spam email or URLs in tweets. However, [TGSP11] focuses on malicious Tweets and finds that blacklisting approaches are too slow to stem the spread of Twitter spam. OddBall [AMF10] proposes features based on egonets to find anomalous users on weighted graphs. [DDS+04] and [LM05] take a game theoretic approach to learning simple classifiers over generic features to detect spam. While related in the adversarial perspective, these approaches focus on general feature-based classification as used for spam email, rather than graph analysis as is needed for link fraud detection.

3.3 An Adversarial Analysis – Our Perspective

In this section, we examine the exploitability of state-of-the-art methods from an adversarial point-of-view and present lemmas and theorems detailing the limitations of these methods. Particularly, we demonstrate through theoretical analysis that existing methods are highly vulnerable to evasion by intelligent attackers. Table 3.2 contains a comprehensive list of symbols and corresponding definitions used in our paper.



(a) Naïve attack (b) Staircase attack (c) Random attack

Figure 3.2: **Figures 3.2(a)-(c) show the different types of adversarial attacks we characterize.** Note the distinctions between how the fraudulent links would be distributed in the relevant attack subgraphs.

Given knowledge of the detection threshold used by a certain service, how can an attacker engineer smart attacks on that service to avoid detection by fraud detection methods?

Formally, we pose the following adversarial problem:

Symbol	**Definition**
u and o	Number of user and object nodes described by the input graph
$\mathcal{U}$ and $\mathcal{P}$	Sets of indexed rows and columns corresponding to user and object nodes in the input graph
$\mathbf{A}$	$u \times o$ input graph adjacency matrix where $\mathbf{A}_{x,y} = 1$ if a link exists between user node x and object node y
f and c	Number of attacker and customer nodes described by the attack graph
s	Number of fraudulent actions each customer node has paid commission for in the attack graph
$\mathcal{F}$ and $\mathcal{C}$	Sets of indexed rows and columns corresponding to attacker nodes and customer nodes in the attack graph
$\mathbf{S}$	$f \times c$ attack graph adjacency matrix where $\mathbf{S}_{x,y} = 1$ if a link exists between attacker node x and customer node y
k	Decomposition rank parameter used by spectral methods
λ_k and σ_k	kth largest eigenvalue and singular value of a given matrix (largest values for $k = 1$)
m, n and p	Bipartite core size and edge probability parameters used by clustering methods

Table 3.2: Frequently used symbols and definitions

Problem 3.1: Stealth Attack Engineering

Given an input graph adjacency matrix **A**, with rows and columns corresponding to users and objects, **engineer** a stealth attack which falls *just below* the minimum sized attack detectable by spectral methods.

As previously described, most detection methods focus on finding fairly blatant bipartite cores or cliques in the input graph. Therefore, if an adversary knows the minimum size attack that detection methods will catch, he can carefully engineer attacks to fall just below that threshold. For clustering approaches like CopyCatch, this threshold is clearly set based on input parameters, and the attacker can simply use fewer accounts than specified to avoid detection. In this setting, the practitioner will try to set n, m and p as strictly as possible and an adversary will attempt to add as many edges as possible without creating a detectable temporally coherent bipartite core.

However, for spectral methods like SpokEn, the possible attack size for an adversary is unclear. We argue that from an adversarial perspective, these spectral methods have a detection threshold based on the input graph's singular values. For a rank k SVD used in these methods, this threshold is governed by the kth largest singular value, σ_k. In practice, an adversary could estimate σ_k from the results of various experimental attacks conducted at distinct scales, or by conducting analysis on publicly available social network data. Once an adversary has such an estimate, we show that it is easy to conduct attacks on the graph adjacency matrix **A** that will necessarily lie below this threshold and avoid detection.

To analyze what type of attacks can evade detection by spectral methods, let us consider that there are c customers who have each commissioned an attacker with f nodes in his botnet for s fraudulent actions (page likes, followers, etc.), where $s \leq f$. This type of attack can be considered as an injected submatrix **S** of size $f \times c$, where rows correspond to attacker nodes (controlled by a single fraudulent operator) in the set of users ($\mathcal{F} \subset \mathcal{U}$) and the columns represent customers in the set of objects ($\mathcal{C} \subset \mathcal{P}$). In this formulation, the desired in-degree of all nodes in **S** is s.

As described earlier, an attack will only be detected by a spectral algorithm if it appears in the top k singular values/vectors. Therefore, our goal as an adversary becomes to understand the spectral properties of our attacks and ensure that they do not project in the rank k decomposition. We can consider the spectral properties of **S** in isolation from the rest of the graph, as it is well known that the spectrum of a disconnected graph is the union of the spectra of its connected components. From this, we deduce that it is sufficient to consider only the representation of **S** and ignore the remainder of **A** when trying to minimize the leading singular value that the attack contributes to the singular spectrum of **A**. *Therefore, an attack* **S** *with leading singular value* σ' *will go undetected by spectral methods if* $\sigma' < \sigma_k$*, where* σ_k *is the* k*th largest singular value computed for the adjacency matrix* **A**.

Having reduced the problem of adversarial injection to distributing some amount of fraudulent activity over the $f \times c$ matrix **S**, we next consider several distinct patterns of attack which characterize types of fraudulent behavior discovered in the analysis of prior work. Specifically, we explore three fraud distribution techniques: naïve, staircase and random graph injections. Figure 3.2 gives a pictorial representation of each of these types of attacks. We evaluate the suitability of each attack for an adversary on the basis of the leading singular value that the pattern generates.

3.3.1 Naïve Injection

This is the most notable attack pattern considered in prior work. The naïve injection distributes the sc total fraudulent actions into an $s \times c$ submatrix of **S**. Thus, only s of the f attacker nodes perform any fraudulent actions, and all fraudulent actions are distributed between these s nodes. In graph terms, this is equivalent to introducing a $s \times c$ complete bipartite core. The leading singular value characterization of such an attack is as follows:

Theorem 3.1: Naïve Attack Singular Value

The leading singular value of an $s \times c$ bipartite core injection is $\sigma_1 = \sqrt{cs}$.

Proof. Since **S** is a full block, where $\mathbf{S_{i,j}} = 1$ for all $i \leq s, j \leq c$, $\mathbf{SS^T}$ must be an $s \times s$ matrix where $\mathbf{SS^T}_{\mathbf{i,j}} = c$ for all $i, j \leq s$. By the Perron-Frobenius Theorem for non-negative matrices, the leading eigenvalue of $\mathbf{SS^T}$ λ_1 is bounded by

$$\min \sum_j \mathbf{SS^T}_{\mathbf{i,j}} \leq \lambda_1 \leq \max \sum_j \mathbf{SS^T}_{\mathbf{i,j}} \text{ for } i \leq s$$

Given that the row sums are equal to cs, $\lambda_1 = cs$ for $\mathbf{SS^T}$. Since the singular values of **S** are equal to the square roots of the eigenvalues of $\mathbf{SS^T}$ by definition, it follows naturally that the leading singular value is $\sigma_1 = \sqrt{cs}$ for **S**. ■

Such an attack corresponds to attackers naïvely linking the same set of s nodes to each of the c customers, producing a full block in **A**. Figure 3.2a shows a visual representation of such an attack.

3.3.2 Staircase Injection

The staircase injection (discovered in [JCB$^+$14b]) evenly distributes cs fraudulent actions over f attacker nodes. However, unlike in the naïve method, where each node that performs any fraudulent actions does so for each of the c customers, the staircase method forces different subsets of nodes to associate with different subsets of customers. A characterization of the leading singular value of such an attack is as follows:

Theorem 3.2: Staircase Attack Singular Value

The leading singular value of an s, c, f staircase injection is $\sigma_1 = s\sqrt{c/f}$.

Proof. The staircase injection is approximately equivalent (row-sum-wise) to a random graph-injection of $f \times c$ with edge probability $p = s/f$. The reduction is as follows: Consider that the in degree of each customer is s by construction of the pattern. Next, note that by definition of the staircase pattern, the starting row index of a sequence of existing links (denoted by 1s in cells of the matrix) in a column given the column index c_i (0-indexed) is $c_i s$ mod f. Then, it is apparent that the periodicity of the pattern of starting indices is $t = lcm(s, f)/s$ and the out degree of each fraudster per $f \times t$ block can be calculated as st/f. Given that $t|c$, it follows that starting indices are uniformly distributed, ensuring that the out degree is uniform and equal to $(st/f)(c/t) = cs/f$. Note that if $f = c$ (for square **S**), the out degree, like the in degree is also equal to s. Also note that the uniformity of in degrees and out degrees means that each node will have s out of a possible f in-degree and sc/f out of a possible c out-degree. Since $s/f = sc/(fc)$, it follows that the staircase injection can be construed as a random graph injection of $f \times c$ with edge probability $p = s/f$. Such a random graph injection has singular value $p\sqrt{fc} = s\sqrt{c/f}$ (proof given in Section 3.3.3). ■

This distribution results in the **S** matrix looking like a staircase of links. Figure 3.2b shows a visual representation of such an attack.

We restrict our analysis here to staircase injections in which all users have equal out degrees o and equal in degrees i, though o need not equal i. When out degrees and in degrees are not equal, users and objects do not have uniform connectivity properties which complicates calculations. In particular, we assume that the periodicity of the staircase pattern, given by $t = lcm(s, f)/s$ is such that $t|c$ to ensure this criteria. However, for large values of c/t, $\sigma_1 \approx s\sqrt{c/f}$ given LLN.

3.3.3 Random Graph Injection

The random graph injection bears close resemblance to the near-bipartite core with density p attack noted in [BXG$^+$13]. The random graph injection distributes $\approx sc$ fraudulent actions over the f attacker nodes approximately evenly. Figure 3.2c shows a visual representation of such an attack. This approach assigns each node a fixed probability $p = sc/cf = s/f$ of performing a fraudulent operation associated with one of the c customers. Given LLN, the average number of fraudulent operations per customer will be close to the expected value of s, and as a result the total number of fraudulent actions will be close to sc. A characterization of the leading singular value of such an attack is as follows:

Theorem 3.3

The leading singular value of an s, c, f directed random bipartite graph is $\sigma_1 \sim s\sqrt{c/f}$.

Proof. Given that probability of an edge between an attacker node and a customer is $p = s/f$, it is apparent that

$$E(\mathbf{SS^T}_{\mathbf{i,j}}) = p^2c \text{ for } i, j \leq f$$

since the value of each cell in the $f \times f$ matrix $\mathbf{SS^T}$ will be a result of the inner product of the corresponding row and column vectors of length c with probability p of a nonzero entry at any $i \leq c$. Since each row in $\mathbf{SS^T}$ has f entries,

$$E(\sum_j \mathbf{SS^T}_{\mathbf{i,j}}) = p^2cf \text{ for } i \leq f$$

By the Perron-Frobenius theorem for non-negative matrices, the leading eigenvalue λ_1 of $\mathbf{SS^T}$ will be bounded by

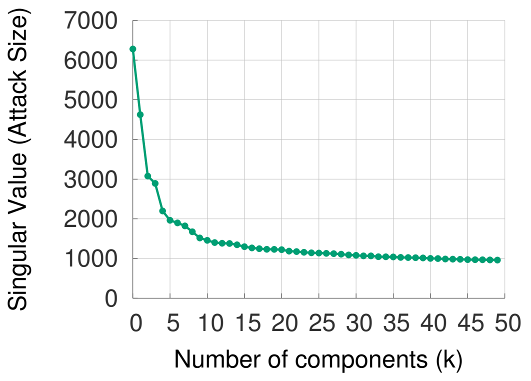
$$\min \sum_j \mathbf{SS^T}_{\mathbf{i,j}} \leq \lambda_1 \leq \max \sum_j \mathbf{SS^T}_{\mathbf{i,j}} \text{ for } i \leq f$$

Given that the row sums are all approximately equal to $p^2cf = cs^2/f$ (exactly equal to cs^2/f if edges in **S** are perfectly uniformly distributed), the leading eigenvalue is $\lambda_1 \approx cs^2/f$ for $\mathbf{SS^T}$. Since the singular values of **S** are equal to the square roots of the eigenvalues of $\mathbf{SS^T}$, it follows naturally that the leading singular value is $\sigma_1 = s\sqrt{c/f}$ for **S**. ■

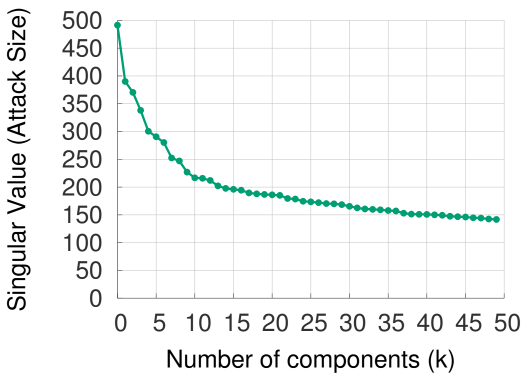
The random graph injection is similar to the Erdös-Rényi model defined by $G(n, p)$ [ER59], except we consider a directed graph scenario with cf possible edges. However, as Erdös and Rényi studied the asymptotic behavior of random graphs, their results are applicable here as well.

3.3.4 Implications and Empirical Analysis

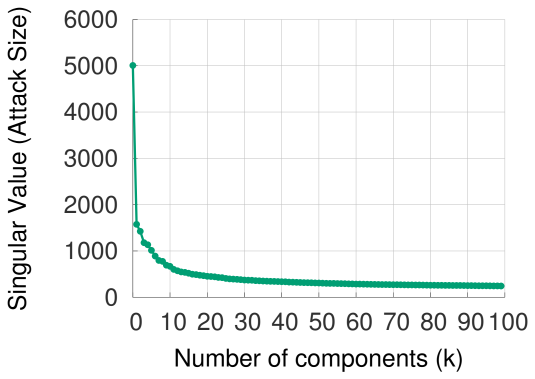
Thus far, we have discussed three different types of potential attack patterns for a fixed number of fraudulent actions and theoretically derived expressions concerning the leading singular value that they contribute to the singular spectrum of **A**. Two of the attack patterns, the staircase and random graph injections, produce leading singular values σ_1 of exactly and approximately $s\sqrt{c/f}$ respectively. Conversely, naïve injection results in a leading singular value of $\sigma_1 = \sqrt{cs}$. Given these results, it is apparent that naïve injection is the least suitable for an adversarial use, since it will necessarily produce a larger singular value than the other two methods given that $s \leq f$. This result is intuitive: the leading eigenvalue of a matrix is a measure of effective connectivity, and packing fraudulent actions into a full block matrix results in higher connectivity



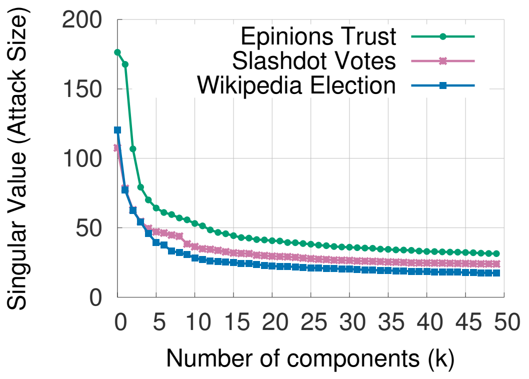
(a) Twitter Followers



(b) Amazon Reviews



(c) Netflix Ratings



(d) Smaller Graphs

Figure 3.3: **Skewed singular value distribution in real networks — spectral (k-rank SVD) approaches suffer from stealth attacks.** (a), (b), (c) and (d) show distributions for corresponding networks which allow stealth attacks capable of significantly impacting local network structure to go undetected.

than spreading the actions out over a large, sparse matrix. Our results beget two important conclusions:

1. Fraud detection tools must consider modes of attack other than naïve injection — more intelligent and less detectable means of attack exist and are being used.
2. Given knowledge of the effective singular value threshold σ_k used by spectral detection methods, or m, n, p parameter choice for clustering based methods, attackers can easily engineer attacks of scale up to *just below* the threshold without consequence.

To demonstrate that this leaves a significant opening for attackers, we analyze the distribution of singular values for a variety of real world graphs and show just how easy it is to construct attacks which slip below the radar. In particular, we compute the SVD for six different real world graphs: Twitter's who-follows-whom social graph, Amazon's bipartite graph of user reviews for products, Netflix's graph of user reviews for movies, Epinions's network of who-trusts-whom, Slashdot's friends/foe social graph, and Wikipedia's bipartite graph of votes for administrators.

Graph	Nodes	Edges
Twitter [KLPM10]	41.7 million	1.5 billion
Amazon [ML13]	6m users & 2m products	29 million
Netflix [Net06]	480k users & 17k videos	99 million
Epinions [LHK10]	131,828	841,372
Slashdot [LHK10]	82,144	549,202
Wikipedia [LHK10]	8274	114,040

Table 3.3: Graphs used for empirical analysis

For each graph, we turn it into a binary bipartite graph and compute the SVD for a fixed rank. The properties of the datasets can be seen in Table 3.3 and the results can be seen in Figure 3.3.

In Figure 3.3a we observe the top $k = 50$ singular values for the Twitter graph. We see that the largest singular value is over 6000, but as k increases the singular values begin to settle around 1000, with $\sigma_{50} = 960.1$. Theorem 3.1 implies that an attacker controlling 960 accounts could use them to follow 960 other accounts and avoid projecting onto any of the top 50 singular vectors. Note that Theorem 3.1 also implies that an attacker could add 92 thousand followers to 10 lucky accounts and also go undetected. These are very large numbers of followers that could significantly shift the perception of popularity or legitimacy of accounts. Common spectral approaches would fail to detect such attacks.

A similar analysis can be made for the other graphs. Figure 3.3b shows that $\sigma_{50} = 141.6$ in the Amazon review graph. Therefore, attackers could add 140 reviews for 140 products without projecting onto the top 50 singular vectors. Considering the average product has 12.5 reviews and a product in the 99^{th} percentile has 187 reviews, 140 reviews is sufficiently large to sway perception of a product on Amazon.

As seen in Figure 3.3c, we find that $\sigma_{50} = 309.7$ and $\sigma_{100} = 243.4$ for the Netflix ratings graph. Therefore, attackers could naïvely add an injection of 240 ratings to 240 videos from 240 accounts and avoid detection in the top 100 singular vectors.

For the Epinions network, we see in Figure 3.3d that $\sigma_{50} = 31.4$. Although this value is much smaller than that for other graphs, the Epinions network is small and sparse, with the average user having an in-degree of 6.4. Based on this singular value, an attacker adding 30 edges (statements of trust or distrust) to 30 users would significantly influence the external view of those users.

In Slashdot's friend vs. foe graph, $\sigma_{50} = 23.9$, as seen in Figure 3.3d. This means that attackers could add 23 ratings for 23 users while avoiding spectral detection. Considering that the average in-degree for accounts in this network is 6.7, adding 23 edges would significantly impact the perception of a user.

Lastly, we examine the graph of 2794 administrative elections on Wikipedia. As shown in Figure 3.3d, $\sigma_{50} = 17.5$. This implies that 17 users could for 17 elections all vote together and avoid

	No Camouflage	**Camouflage**
Blatant Attacks	SpokEn; CopyCatch	Spectral subspace plotting; CopyCatch
Stealth Attacks	(proposed) **fBox**	(proposed) **fBox**

Table 3.4: Types of attacks and suitable detection techniques

detection. In fact, 31% of elections were settled by 17 votes or less. An attacker could also modify the shape of the attack such that 5 users would each receive 57 votes, enough to win 72%. Given an attack of this scale, a small group of accounts could cooperate to unfairly rig election outcomes.

From these examples across a variety of networks, we see that using spectral approaches for catching fraud leaves a wide opening for attackers to manipulate online graphs.

3.4 Proposed Framework for Fraud Behaviors

As demonstrated in Section 3.3, current detection methods are effective in catching blatant attacks, but drop in efficacy as the attack size decreases. Though the scale of attacks detected is defined differently for various datasets given distinct decomposition rank k, such a detectability cross-over point necessarily exists given the well-defined nature of the singular value produced by common types of attacks. In this section, we give a broader overview of possible attack modes and the capabilities of current methods in dealing with them. Table 3.4 illustrates how current techniques fit into our classification of suitable defenses against four different attack types and how the proposed fBox algorithm can fill in the remaining holes to provide a more holistic framework for fraud detection.

The four types of attacks we broach in this work are classified based on two dichotomies — the *scale* of attack and the presence of *camouflage*. The scale of attack concerns whether an attack of some size defined in terms of the aforementioned s, c and f parameters in the context of a given dataset (and decomposition rank k for spectral methods), is detectable or not. The attack could be staged using any of the fraud distribution patterns discussed in Section 3.3. In the context of clustering methods, scale is more formally defined by the minimal attack size parameters used. Camouflage refers to uncommissioned actions conducted by attackers in order to appear more like honest users in the hopes of avoiding detection. For example, camouflage on Twitter is most commonly seen as attackers following some honest users for free in addition to paid customers. Attacks with camouflage are more difficult to detect than those without, given the increased likelihood of a practitioner to overlook suspicious actions.

3.4.1 Blatant Attack/No Camouflage

Of the four types, blatant attacks without camouflage are the easiest to spot. Blatant attacks whose singular values are above the threshold σ_k and thus appear in the rank-k decomposition of spectral methods produce spoke-like patterns and can be identified using SpokEn. It is worth noting that SpokEn is a method to chip out large communities from graphs, and not necessarily attackers. Verification of the blatant lockstep behavior as fraudulent is required in this case.

3.4.2 Blatant Attack/Camouflage

Naturally, blatant attacks with camouflage are more difficult to spot than without. Though the singular values of the attacks are above the threshold σ_k and the associated singular vectors appear in the rank-k decomposition of spectral methods, Jiang et al. showed that rather than axis-aligned spokes, the spectral subspace plots showed tilting rays. CopyCatch is also effective in detecting blatant attacks with camouflage (provided that the parameter choices are sufficiently large to limit the rate of false positives), given that camouflage is ignored in the case that an m, n, p near-bipartite core is found for a subset of $\mathcal{U}$ and $\mathcal{P}$ for a fixed snapshot of the input graph.

3.4.3 Stealth Attack/No Camouflage

As concluded in Section 3.3, current detection schemes are highly vulnerable to stealth attacks engineered to fall below parameter thresholds of σ_k for spectral methods or m, n, p for clustering methods. To the best of our knowledge, no previous technique has been able to successfully and effectively identify users involved in these types of attacks. Though stealth attacks may be individually of lesser consequence to detect than larger cases of fraud, they have the insidious property of being able to achieve the same number of fraudulent actions in a more controlled and less detectable manner at the cost of simply creating more fraud-related accounts. In response to this threat, we propose the fBox algorithm for identifying such attacks in Section 3.5 and demonstrate its effectiveness in Section 3.6.

3.4.4 Stealth Attack/Camouflage

Given that identifying small scale attacks has thus far been an open problem in the context of fraud detection, the problem of identifying these with camouflage has also gone unaddressed. The difficulty in dealing with camouflage is particularly apparent when considering user accounts with few outgoing or incoming links, as is typically the case with smaller attacks. From the perspective of a practitioner, it may appear that a truly fraudulent account is mostly honest but with a few suspicious or uncharacteristic links (insufficient to mark as fraudulent) or infrequently/unsavvily used due to the small number of total links. We demonstrate in Section 3.6 that fBox is robust to such smart attacks with moderate amounts of camouflage on real social network data.

3.5 Proposed Algorithm

Thus far, we have seen how existing state-of-the-art techniques have firm effective detection thresholds and are entirely ineffective in detecting stealth attacks that fall below this threshold. Given this problem, it is natural to consider the following question — how can we identify the many numerous small scale attacks that are prone to slipping below the radar of existing techniques? In this section, we formalize our problem definition and propose fBox as a suitable method for addressing this problem.

Algorithm 3.1: fBox algorithm pseudocode

```
Input: Input graph adjacency matrix $\mathbf{A}$,
       Decomposition rank $k$,
       Threshold $\tau$
userCulprits = $\{\}$
objectCulprits = $\{\}$
outDegrees = $rowSum(\mathbf{A})$
inDegrees = $colSum(\mathbf{A})$
$[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = svd(\mathbf{A}, k)$
for each row $i$ in $\mathbf{U\Sigma}$ do
    recOutDegs = $\|(\mathbf{U\Sigma})_\mathbf{i}\|_2^2$
end for
for each row $j$ in $\mathbf{V\Sigma}$ do
    recInDegs = $\|(\mathbf{V\Sigma})_\mathbf{j}\|_2^2$
end for
for each unique $od$ in outDegrees do
    nodeSet = $find$(outDegrees $==$ $od$)
    recOutDegSet = recOutDegs(nodeSet)
    recThreshold = $percentile$(recOutDegSet, $\tau$)
    for each node $n$ in nodeSet do
        if recOutDegs($n$) $\leq$ recThreshold then
            userCulprits = userCulprits + $n$
        end if
    end for
end for
for each unique $id$ in inDegrees do
    nodeSet = $find$(inDegrees $==$ $id$)
    recInDegSet = recInDegs(nodeSet)
    recThreshold = $percentile$(recInDegSet, $\tau$)
    for each node $n$ in nodeSet do
        if recInDegs($n$) $\leq$ recThreshold then
            objectCulprits = objectCulprits + $n$
        end if
    end for
end for
return userCulprits,
       objectCulprits
```

3.5.1 Problem Formulation

We identify the major problem to be addressed as follows:

Problem 3.2: Stealth Attack Detection

Given an input graph adjacency matrix $\mathbf{A}$, with rows and columns corresponding to users and objects (could be pages, articles, etc. or even other users), **identify** stealth attacks which are undetectable given a desired decomposition rank-k for $\mathbf{A}$ (undetectable in that their singular values fall below the threshold σ_k).

Note that Problem 3.2 is an exact foil to Problem 3.1. In this paper, we primarily focus on smart attacks which fall below a practitioner-defined spectral threshold, given that a number of previous works mentioned have tackled the problem of discovering blatant attacks. Given that this body of work is effective in detecting such attacks, we envision that the best means of boxing in attackers is a *complementary* approach to existing methods, as our analysis in Section 3.4 is indicative of the lack of suitability of a one-size-fits-all technique for catching all attackers.

3.5.2 Description

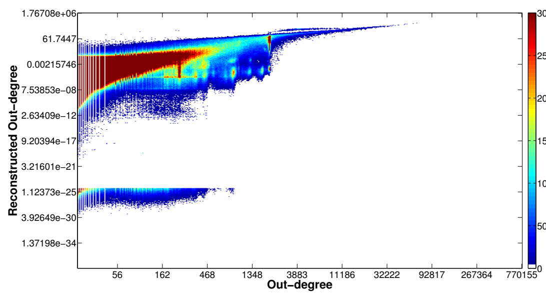
As per the problem formulation, we seek to develop a solely graph-based method, which will be able to complement existing fraud detection techniques by discerning previously undetectable attacks. In Section 3.3, we demonstrated that smaller attacks are particularly characterized by comparatively low singular values (below σ_k), and thus do not appear in the singular vectors given by a rank k decomposition. Assuming an isolated attack which has been engineered to fall below the detection threshold, the users/objects comprising the attack will have absolutely *no* projection onto any of the top-k left and right singular vectors respectively. In the presence of camouflage, projection of the culprit nodes may increase slightly given some nonzero values in the corresponding indices in one or more of the vectors. In either case, we note that nodes involved in these attacks have the unique property of having zero or almost-zero projections in the projected space. Given this observation, two questions naturally arise: (a) how can we effectively capture the extent of projection of a user or object? and (b) is there a pattern to how users or objects project into low-rank subspaces?

In fact, we can address the first question by taking advantage of the norm-preserving property of SVD given below, which states that the row vectors of a full rank decomposition and associated projection will retain the same l_2 norm or vector length as in the original space. That is, for $k = rank(\mathbf{A})$,

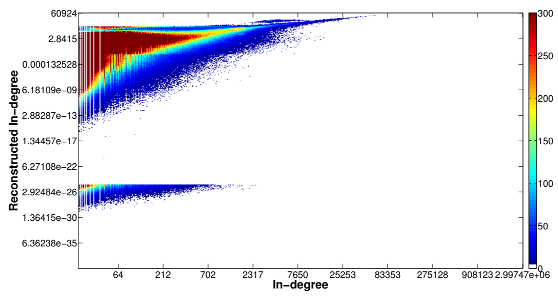
$$\|\mathbf{A_i}\|_2 = \|(\mathbf{U\Sigma})_\mathbf{i}\|_2 \text{ for } i \leq u$$

In the same fashion, one can apply the norm-preserving property to decomposition of $\mathbf{A^T}$ to show

$$\|\mathbf{A^T}_\mathbf{j}\|_2 = \|(\mathbf{V\Sigma})_\mathbf{j}\|_2 \text{ for } j \leq o$$



(a) OSRM for Twitter fans



(b) ISRM for Twitter idols

Figure 3.4: **SRMs show correlation between the reconstruction degree and suspiciousness of nodes.** (a) and (b) show the SRMs produced from analysis on the Twitter social graph.

Since the l_2 norms are preserved in a full rank decomposition, it is obvious that the sum of squares of components are also preserved. Note that for the 0-1 adjacency matrix $\mathbf{A}$ we consider here, the sum of squares of components of the ith row vector corresponds to the out-degree of user i and the sum of squares of components of the jth column vector corresponds to the in-degree of object j — given these considerations, we define the degree of a node in a given subspace as the squared l_2 norm of its link vector in that subspace. Thus, for a full rank decomposition, the out-degree given by $\|\mathbf{A_i}\|_2^2$ and *reconstructed* out-degree given by $\|(\mathbf{U\Sigma})_\mathbf{i}\|_2^2$ of user i are equal. The same can be said for the in-degree and *reconstructed* in-degree of object j. For rank k decompositions where $k < rank(\mathbf{A})$ (guaranteed in practical use of spectral methods), we can show that the true degrees upper bound the reconstructed degrees as follows:

Theorem 3.4: l_2-norm Bound for Reconstruction Degree

The reconstruction degree of any node (row) i in a k-rank projection of $\mathbf{A}$, $\mathbf{A}_{proj}$ is upper bounded by the true degree of the same node (row) i in $\mathbf{A}$.

Proof. It is sufficient to show that the respective l_2 norms of row i in $\mathbf{A}_{proj}$ is upper bounded by the l_2 norm of the same row in $\mathbf{A}$, or that $\mathbf{A}_{proj_i} \leq \mathbf{A}_i$. First, observe that when the projection rank $k = rank(\mathbf{A})$, the reconstruction degree and true degree for row i are equal – that is, $\|\mathbf{A}_{proj_i}\|_2 = \|\mathbf{A}_i\|_2$. Since U and V are unitary, we can rewrite the SVD formulation $\mathbf{A}_{proj} = \mathbf{U}\Sigma\mathbf{V}^T$ as $\mathbf{A}_{proj}\mathbf{V} = \mathbf{AV} = \mathbf{U}\Sigma$ and observe that RHS has the same l_2 norm as LHS given the norm-preserving property of unitary matrices. Next, note that a $(k+1)$-rank decomposition has the same first k columns as a k-rank decomposition by the uniqueness (up to sign) of SVD, and thus the same l_2 norm for the ith row over the k-length row vector formed over the first k entries in the row. Then, any nonzero element in the $(k+1)^{th}$ column and i^{th} row will increase the l_2 norm of row i as it will contribute a positive term under the square-root. If the element in the $(k+1)^{th}$ column and i^{th} row is 0, then the l_2 norm of row i remains the same. In either case, the l_2 norm of row i is non-decreasing with increasing rank, and thus cannot exceed the l_2 norm of a full-rank decomposition. ■

Thus, we can capture the extent of projection of a user by the tuple of his true out-degree and reconstructed out-degree, and we can capture the extent of projection of an object by the tuple of its true in-degree and reconstructed in-degree.

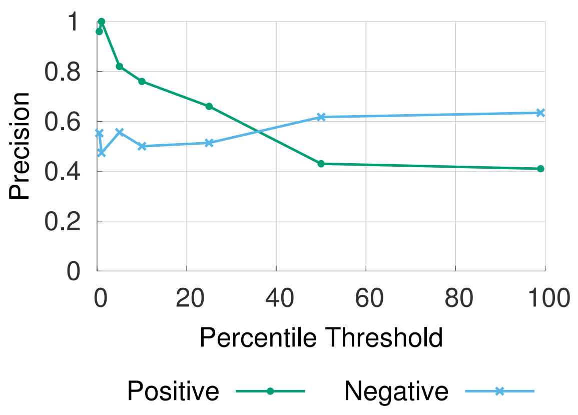
We conjecture that due to the different graph connectivity patterns of dishonest and honest users as well as dishonest and honest objects, their projections in terms of reconstructed degrees should also vary. Intuitively, dishonest users who either form isolated components or link to dishonest objects will project poorly and have characteristically low reconstruction degrees, whereas honest users who are well-connected to real products and brands should project more strongly and have characteristically higher reconstruction degrees. In fact, we find that in real data, users and objects have certain ranges in which they commonly reconstruct in the projected space. Figure 3.4 shows the OSRM (*Out-link Spectral Reconstruction Map*) and ISRM (*In-link Spectral Reconstruction Map*) for a large, multi-million node and multi-billion edge social graph from Twitter, where we model follower (fan) and followee (idol) behavior. The data is represented in heatmap form to indicate the distribution of reconstructed degrees for each true degree. The SRMs indicate that for each true degree, there is a tailed distribution with most nodes reconstructing in a common range and few nodes reconstructing as we move away from this range in either direction. Most notably, there are a large number of nodes with degrees up to the hundreds with an almost-zero reconstruction, depicted by a well separated point cloud at the bottom of both SRMs. For higher true degree values in the thousands, nodes are more sparse and rarely project as poorly as for lower true degrees, but many points at these degree values reconstruct several degrees of magnitude lower than the rest. These observations serve to substantiate our conjecture that poorly reconstructing nodes are suspicious, but what about the well reconstructing nodes? Interestingly, we find that nodes which reconstruct on the high range of the spectrum for a given degree have many links to popular (and commonly Twitter-verified) accounts. We do not classify such behavior as suspicious in the OSRM context, as it is common for Twitter users to follow popular actors, musicians, brands, etc. We do not classify such behavior as suspicious in the ISRM context either, as popular figures tend to more commonly be followed by other popular figures. At the bottom of the reconstruction spectrum, however, we most commonly find accounts which demonstrate a number of notably suspicious behaviors in the context of their followers/followees and the content of their Tweets — more details are given in Section 3.6.

Based on our intuitive conjecture and empirical verification, we focus our fBox algorithm on identifying nodes with characteristically poor reconstructed degree in comparison to other nodes of the same true degree as suspicious. Specifically, we mark the bottom τ percent of nodes per fixed degree for both users and objects as culprit nodes. We outline the high-level steps of fBox in Algorithm 3.1.

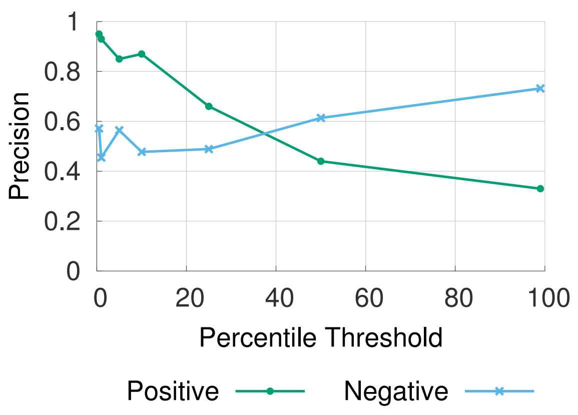
3.6 Experiments

3.6.1 Datasets

For our experiments we primarily use two datasets: the who-follows-whom Twitter graph and the who-rates-what Amazon graph. The Twitter graph was scraped by Kwak et al. in 2010 and contains 41.7 million users with 1.5 billion edges [KLPM10]. We showed the distribution



(a) Fans



(b) Idols

Figure 3.5: **(a) and (b) show fBox's strong predictive value and low false-discovery rate in identifying suspicious accounts.**

of singular values in Figure 3.3a. The Amazon ratings graph was scraped in March 2013 by McAuley and Leskovec [ML13] and contains 29 million reviews from 6 million users about 2 million products. The distribution of singular values can be seen in Figure 3.3b. Our analysis is conducted both directly and via synthetic attacks.

3.6.2 fBox on real Twitter accounts

To show our effectiveness in catching smart link fraud attacks on real data, we conducted a classification experiment on data from the Twitter graph. Specifically, we collected the culprit results for suspicious fans and idols with degree at least 20 (to avoid catching unused accounts) for seven different values of the detection threshold τ, at 0.5, 1, 5, 10, 25, 50 and 99 percentile. For each combination of τ value and user type (fan or idol), we randomly sampled 50 accounts from the "culprit-set" of accounts classified as suspicious by fBox and another 50 accounts from the remainder of the graph in a 1:1 fashion, for a total of 1400 accounts. We randomly organized and labeled these accounts as suspicious or honest (ignoring foreign and protected accounts) based on several criteria — particularly, we identified suspicious behavior as accounts with some combination of the following characteristics:

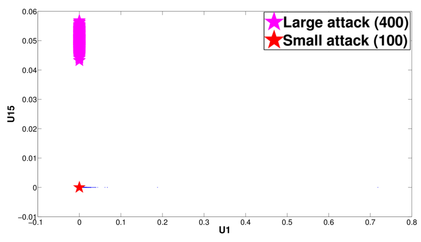
- Suspension by Twitter since data collection
- Spammy or malicious tweets (links to adware/malware)
- Suspicious username, or followers/followees have suspicious usernames (with common prefixes/suffixes)
- Very few tweets (<5) but numerous (>20) followees who are themselves suspicious
- Sparse profile but numerous (>20) followees who are themselves suspicious

Figure 3.5 shows how the performance of fBox varies with the threshold τ for Twitter fans and idols. As evidenced by the results, fBox is able to correctly discern suspicious accounts with 0.93+ precision for $\tau \leq 1$ for both fans and idols. And as expected, increasing τ results in lower precision. As with many informational retrieval and spam detection problems, there are an unbounded number of false negatives, making recall effectively impossible to calculate. Rather, we use the negative precision and observe that it increases as we increase τ. Ultimately, because fBox is meant to be a complementary method to catch new cases of fraud, we do not believe that missing some of the attackers already caught by other methods is a major concern. With these considerations, we recommend conservative threshold values for practitioner use. On Twitter data, we found roughly 150 thousand accounts classified as suspicious between the SRMs for $\tau = 1$.

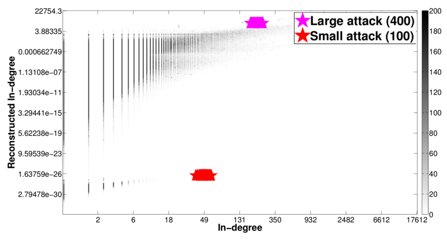
3.6.3 Complementarity of fBox

As mentioned before, fBox is complementary to spectral techniques and is effective in catching smart attacks that adversaries could engineer to avoid detection by these techniques. We demonstrate this claim using both synthetically formulated attacks on the Amazon network as well as comparing the performance of both fBox and SpokEn on the Twitter network. In the first experiment, we inject random attacks of scale 100 (100×100) and 400 (400×400), each with density $p = 0.5$ into the Amazon graph and compare the effectiveness of spectral subspace plots and SRMs in spotting these attacks. Figure 3.6a shows the spectral subspace plot for the 1st and 15th components of the SVD, corresponding to one naturally existing community and the blatant attack, respectively. The plot clearly shows nodes involved in the blatant attack as a spoke pattern, but groups the nodes involved in the small attack along with many honest nodes that reconstruct poorly in these components at the origin point. However, in Figure 3.6b, we see that the smaller injection is identified as clearly suspicious with distinct separation from other legitimate behavior.

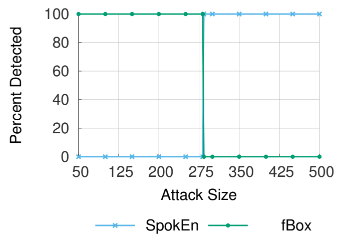
We additionally tested both fBox and SpokEn on a number of injections sizes, each random attacks with $p = 0.5$. Figure 3.6c shows the fraction of the attacking fans caught by each algorithm. As seen in the figure, the two methods are clearly complementary, with fBox catching all attacks that SpokEn misses. This verifies the analysis in Section 3.3 and substantiates fBox's suitability for catching stealth attacks that produce leading singular value $\sigma' < \sigma_k$.



(a) Spectral subspace plot



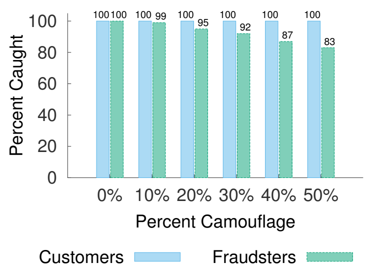
(b) ISRM



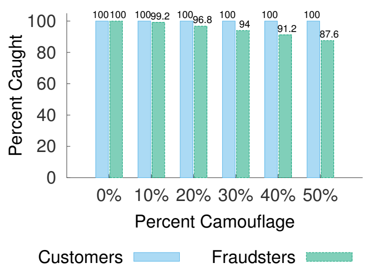
(c) Complementarity

Figure 3.6: **fBox and SpokEn are complementary, with fBox detecting smaller stealth attacks missed by SpokEn.** (a) shows how spectral subspace plots identify blatant attacks but ignore smaller ones. (b) shows the ISRM plot for the same injections, clearly identifying the suspiciousness of the smart attack. (c) depicts the complementary nature of fBox and spectral methods in detecting attacks at various scales.

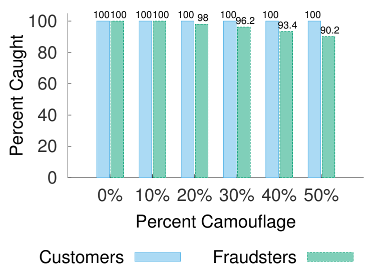
In our second experiment, we compared the performance of both fBox and SpokEn on a sample of 65743 accounts selected from the Twitter graph. For each of these accounts, we queried the Twitter API to collect information regarding whether the account was suspended or had posted Tweets promoting adware/malware (checked via Google SafeBrowsing), and if so we marked the account as fraudulent. This ground truth marking allows us to unbiasedly measure the complementarity of fBox and SpokEn in catching users that are surely malicious. Of these users, 4025 were marked as fraudulent via Twitter (3959) and Google SafeBrowsing (66). For rank $k = 50$, SpokEn produced 8211 suspicious accounts whereas fBox (with $\tau = 1$) produced 149899. The user sets identified by both methods were found to be completely distinct, suggesting that the methods are indeed complementary. Furthermore, fBox identified 1133 suspicious accounts from the sampled dataset, of which only 347 were caught via Twitter and Google SafeBrowsing, suggesting that roughly 70% of fBox-classified suspicious accounts are missed by Twitter.



(a) 100×100 attack



(b) 250×250 attack



(c) 500×500 attack

Figure 3.7: **(a), (b) and (c) show fBox's robustness to moderate amounts of camouflage for attack sizes of 100, 250 and 500.**

3.6.4 fBox in the face of camouflage

One key point in dealing with intelligent attackers is ensuring that fBox is robust in detecting attacks with moderate amounts of camouflage. To measure our performance in such a setting, we ran fBox on a variety of attack sizes in our target range and for each attack varied the amount of camouflage added. In our model, we include camouflage by following honest accounts at random. For a random attack of size $n \times n$ and edge probability p, we vary the percent of idols of fraudulent fans that are camouflage: for 0% camouflage each fan follows the pn customers only and for 50% camouflage each attacker node follows pn customers and pn random honest idols — in general, the percent of camouflage r for g camouflage links is defined as $\frac{100g}{g+pn}$. We ran this test for attacks of size 100, 250, and 500 (all below the $\sigma_{25} = 1143.4$ detection threshold) with $p = 0.5$ on the Twitter graph.

Figure 3.7 demonstrates fBox's robustness — for all configurations of attack size and camouflage, we catch *all* customer idols and over 80% of fraudulent fans. As attack size increases, increased camouflage is less impactful (intuitively, larger attacks are more flagrant), with fBox catching over 90% of the fraudulent fans even with 50% camouflage.

Analysis on *fame*, where customers buying links also have honest links was not conducted. Customer fame is the analog of attacker camouflage. However, we expect similar results in detection of accounts in the presence of fame given the symmetry of SVD and fBox's disjoint user/object reconstruction. However, the presence of fame is less realistic in many applications — for example, in the Twitter context, it is difficult for a spammy account to get honest fans whereas fraudulent fans can follow real idols at will.

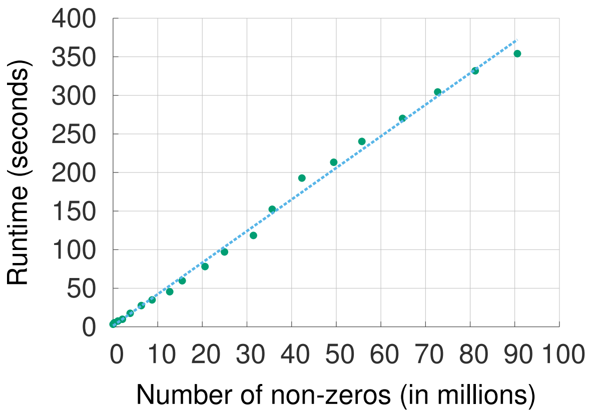


Figure 3.8: **fBox scales linearly on the size of input data.**

3.6.5 Scalability of fBox

The running time of fBox is dominated by the (linear) large matrix-vector multiplication per iteration of the Lanczos algorithm to compute SVD for large, sparse matrices. Figure 3.8 depicts the linear runtime of fBox for $k = 25$ while varying number of non-zeros.

3.7 Conclusions

In this work, we approached the problem of distinguishing dishonest attackers and their customers from honest users in the context of online social network or web-service graphs using a graph-based approach (using the adjacency matrix representing user/object relationships). Our main contributions are:

1. **Theoretical analysis:** We examine several state-of-the-art fraud detection methods from an adversarial point-of-view and provide theoretical results (Theorems 3.1-3.3) pertaining to the susceptibility of these methods to various types of attacks.
2. **fBox algorithm:** We detail fBox, a method motivated by addressing the blind-spots discovered in theoretical analysis, for detecting a class of *stealth* attacks which previous methods are effectively unable to detect.
3. **Effectiveness on real data:** We apply fBox to a large Twitter who-follows-whom dataset from 2010 and discover many tens of thousands of suspicious users with over 93% precision whose accounts remain active to date.

Our experiments show that our method is scalable, effective in detecting a complementary range of attacks to existing methods and robust to a reasonable degree of camouflage for small and moderately sized stealth attacks.

Based on content published in [KSV+15].

Chapter 4

DeltaCon-Attr: Cross-Graph Blame Attribution

Identifying culprits of connectivity change between graph snapshots using influence-based similarity.

How are two graphs different? Which nodes and edges are responsible for the changes between them? Detection and ranking of changes in graphs is a problem which arises in numerous settings related to pinpointing anomalous behavior. Notable examples include tracking physical network interconnectivity and analyzing behavior of users in communication networks. In this work, we propose DeltaCon-Attr, a complement to the DeltaCon graph similarity algorithm, for the change detection and ranking task. DeltaCon-Attr is principle and agrees with human intuition, scalable and demonstrates practical effectiveness.

4.1 Introduction

Graphs arise naturally in numerous situations; social, traffic, collaboration and computer networks, images, protein-protein interaction networks, brain connectivity graphs and web graphs are only a few examples. A problem that comes up often in all those settings is the following: how much do two graphs or networks differ in terms of connectivity, and which are the main node and edge culprits for the difference?

In this work, we tackle this problem on graphs with known node-correspondence (fixed node-set across graphs). The main focus of this chapter is on the latter problem (blame attribution), in which the aim is to identify the culprit nodes and edges which are most responsible for the largest changes in graph structure. The intuition behind our approach is that nodes and edges which are most culpable for differences between graphs are the ones which most heavily influence changes in *connectivity*. To this end, we leverage the DeltaCon algorithm for computing graph similarity as it is scalable, principled and considerate of connectivity differences [KVF+13].

Change detection and blame attribution is a particularly important problem in anomaly detection scenarios such as detecting changes in the connectivity of a computer network, the behavior of users in communication networks, or even identifying spammers and bad actors in social media. Tracking changes in networks over time, spotting anomalies and detecting events is a research direction that has attracted much interest in previous literature [CBWG11, Nob03, WPT11, WTPP14]. However, blame attribution is still an open problem as the list of requirements constantly increases: the exponential growth in graphs, both in number and size, calls for methods that are not only accurate, but also scalable to graphs with billions of nodes.

In this chapter, we introduce the following contributions:

1. **Algorithm**: We propose DeltaCon-Attr, a scalable and principled approach for node and edge-based blame attribution between graphs.
2. **Experiments**: We detail a number of experiments on synthetic and real datasets and show that DeltaCon-Attr gives results that agree with human intuition of culpability and is faster while maintaining comparable accuracy to the prior state-of-the-art.

The rest of this chapter is organized as follows: Section 4.2 includes background on the DeltaCon approach, Section 4.3 discusses the proposed DeltaCon-Attr approach for blame attribution, Section 4.4 details experiments on synthetic and real datasets, Section 4.5 surveys prior work and Section 4.6 concludes.

4.2 Background: DeltaCon

In this section, we detail our previously proposed DeltaCon [KVF+13] approach for graph similarity, which DeltaCon-Attr leverages. While the graph similarity problem is not the main focus in this chapter, the intuition, motivating properties and algorithmic details are highly relevant for future discourse and are thus included here for the reader's benefit. Most detailed proofs have been omitted in this chapter for relevance reasons, but are available in the full-text of [KSV+15].

4.2.1 DeltaCon: Intuition

How can we find the similarity in connectivity between two graphs or, more formally, how can we solve the following problem?

Symbol	Description
G	graph
$\mathcal{V}, n$	set of nodes, number of nodes
$\mathcal{E}, m$	set of edges, number of edges
$sim(G_1, G_2)$	similarity between graphs G_1 and G_2
$d(G_1, G_2)$	distance between graphs G_1 and G_2
$\mathbf{I}$	$n \times n$ identity matrix
$\mathbf{A}$	$n \times n$ adjacency matrix with elements a_{ij}
$\mathbf{D}$	$n \times n$ diagonal degree matrix, $d_{ii} = \sum_j a_{ij}$
$\mathbf{L}$	$= \mathbf{D} - \mathbf{A}$ laplacian matrix
$\mathbf{S}$	$n \times n$ matrix of final scores with elements s_{ij}
$\mathbf{S'}$	$n \times g$ reduced matrix of final scores
$\vec{e}_i$	$n \times 1$ unit vector with 1 in the i^{th} element
$\vec{s}_{0k}$	$n \times 1$ vector of seed scores for group k
$\vec{s}_i$	$n \times 1$ vector of final affinity scores to node i
g	number of groups (node partitions)
ϵ	$= 1/(1 + \max_i (d_{ii}))$ positive constant (< 1) encoding the influence between neighbors
DC$_0$, **DC**	DeltaCon, DeltaCon$_0$

Table 4.1: Symbols and Definitions. Bold capital letters: matrices, lowercase letters with arrows: vectors, plain font: scalars.

Problem 4.1: DeltaConnectivity

Given (a) two graphs, $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$ with the same node set[1], $\mathcal{V}$, but different edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$, and (b) the node correspondence, **find** a similarity score, $sim(G_1, G_2) \in [0, 1]$, between the input graphs. Similarity score of value 0 means totally different graphs, while 1 means identical graphs.

The obvious way to solve this problem is by measuring the overlap of their edges.

Why does this often not work in practice? It turns out that this measure weights every edge equally in terms of importance. But, clearly, from the aspect of information flow, a missing edge from a clique does not play as important role in the graph connectivity as a missing bridge. So, could we instead measure the differences in the 1-step away neighborhoods, 2-step away neighborhoods etc.? If yes, with what weight? It turns out that our method does exactly this in a principled way.

[1]If the graphs have different, but overlapping, node sets, $\mathcal{V}_1$ and $\mathcal{V}_2$, we assume that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, and the extra nodes are treated as singletons.

4.2.1.1 Fundamental Concept

The first conceptual step of our proposed method is to compute the pairwise node affinities in the first graph, and compare them with the ones in the second graph. For notational compactness, we store them in a $n \times n$ similarity matrix[2] $\mathbf{S}$. The s_{ij} entry of the matrix indicates the influence node i has on node j. For example, in a who-knows-whom network, if node i is, say, republican and if we assume homophily (i.e., neighbors are similar), how likely is it that node j is also republican? Intuitively, node i has more influence/affinity to node j if there are many, short, heavily weighted paths from node i to j.

The second conceptual step is to measure the differences in the corresponding node affinity scores of the two graphs and report the result as their similarity score.

4.2.1.2 How to Measure Node Affinity?

Pagerank [BP98], personalized Random Walks with Restarts (RWR) [Hav03], lazy RWR [AF02], and the "electrical network analogy" technique [DS84] are only a few of the methods that compute node affinities. We could have used Personalized RWR: $[\mathbf{I} - (1-c)\mathbf{A}\mathbf{D}^{-1}]\vec{s_i} = c\ \vec{e_i}$,where c is the probability of restarting the random walk from the initial node, $\vec{e_i}$ the starting (seed) indicator vector (all zeros except 1 at position i), and $\vec{s_i}$ the unknown Personalized Pagerank column vector. Specifically, s_{ij} is the affinity of node j with respect to node i. For reasons that we explain next, we chose to use a more recent and principled method, the so-called Fast Belief Propagation (FaBP) [KKK+11], and specifically a simplified form of it given by:

$$[\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]\vec{s_i} = \vec{e_i} \tag{4.1}$$

where $\vec{s_i} = [s_{i1}, ...s_{in}]^T$ is the column vector of final similarity/influence scores starting from the i^{th} node, ϵ is a small constant capturing the influence between neighboring nodes, $\mathbf{I}$ is the identity matrix, $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal matrix with the degree of node i as the d_{ii} entry.

An equivalent, more compact notation, is to use a matrix form, and to stack all the $\vec{s_i}$ vectors ($i = 1, \ldots, n$) into the $n \times n$ matrix $\mathbf{S}$. We can easily prove that

$$\boxed{\mathbf{S} = [s_{ij}] = [\mathbf{I} + \epsilon^2\mathbf{D} - \epsilon\mathbf{A}]^{-1}}. \tag{4.2}$$

4.2.1.3 Why use Belief Propagation?

The reasons we choose BP and its fast approximation with Equation 4.2 are: (a) it is based on sound theoretical background (maximum likelihood estimation on marginals), (b) it is fast (linear on the number of edges), and (c) it agrees with intuition, taking into account not only direct neighbors, but also 2-, 3-, and k-step-away neighbors, with decreasing weight. We elaborate on the last reason, next:

[2]In practice, we don't measure all the affinities (see Section 4.2.2.2 for an efficient approximation).

Intuition 4.0: Attenuating Neighboring Influence

By temporarily ignoring the term $\epsilon^2\mathbf{D}$ in (4.2), we can expand the matrix inversion and approximate the $n \times n$ matrix of pairwise affinities, $\mathbf{S}$, as

$$\mathbf{S} \approx [\mathbf{I} - \epsilon\mathbf{A}]^{-1} \approx \mathbf{I} + \epsilon\mathbf{A} + \epsilon^2\mathbf{A}^2 + \ldots.$$

As we said, our method captures the differences in the 1-step, 2-step, 3-step etc. neighborhoods in a weighted way; differences in long paths have a smaller effect on the computation of the similarity measure than differences in short paths. Recall that $\epsilon < 1$, and that $\mathbf{A}^k$ has information about the k-step paths. Notice that this is just the intuition behind our method; we do not use this simplified formula to find matrix $\mathbf{S}$.

4.2.1.4 Which properties should a similarity measure obey?

Let $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$ be two graphs, and $sim(G_1, G_2) \in [0, 1]$ denote their similarity score. Then, we want the measure to obey the following axioms:

- A1. *Identity property*: $sim(G_1, G_1) = 1$
- A2. *Symmetric property*: $sim(G_1, G_2) = sim(G_2, G_1)$
- A3. *Zero property*: $sim(G_1, G_2) \to 0$ for $n \to \infty$, where G_1 is the complete graph (K_n), and G_2 is the empty graph (i.e., the edge sets are complementary).

Moreover, the measure must be:

(a) intuitive It should satisfy the following desired properties:

P1. [*Edge Importance*] For unweighted graphs, changes that create disconnected components should be penalized more than changes that maintain the connectivity properties of the graphs.

P2. [*Edge-"Submodularity"*] For unweighted graphs, a specific change is more important in a graph with few edges than in a much denser, but equally sized graph.

P3. [*Weight Awareness*] In weighted graphs, the bigger the weight of the removed edge is, the greater the impact on the similarity measure should be.

We also introduce an additional, *informal*, property:

IP. [*Focus Awareness*] "Random" changes in graphs are less important than "targeted" changes of the same extent.

(b) scalable The huge size of the generated graphs, as well as their abundance require a similarity measure that is computed fast and handles graphs with billions of nodes.

In [KSV$^+$15], we formalize the properties and discuss their satisfiability by our proposed similarity measure theoretically. They are excluded here, as they are not relevant to the main focus of this chapter.

4.2.2 DeltaCon: Details

Now that we have described the high level ideas behind the DeltaCon method, we move on to the details.

4.2.2.1 Algorithm Description

Let the graphs we compare be $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$. If the graphs have different node sets, say $\mathcal{V}_1$ and $\mathcal{V}_2$, we assume that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, where some nodes are disconnected. As mentioned before, the main idea behind our proposed similarity algorithm is to compare the node affinities in the given graphs. The steps of our similarity method are:

Step 1 By Equation 4.2, we compute for each graph the $n \times n$ matrix of pairwise node affinity scores ($\mathbf{S}_1$ and $\mathbf{S}_2$ for graphs G_1 and G_2 respectively).

Step 2 Among the various distance and similarity measures (e.g., Euclidean distance (ED), cosine similarity, correlation) found in the literature, we use the root Euclidean distance (RootED, a.k.a. Matusita distance[3])

$$d = \text{RootED}(\mathbf{S}_1, \mathbf{S}_2) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (\sqrt{s_{1,ij}} - \sqrt{s_{2,ij}})^2}. \tag{4.3}$$

We use the RootED distance for the following reasons:

1. it is very similar to the Euclidean distance (ED), the only difference being the square root of the pairwise similarities (s_{ij}),
2. it usually gives better results, because it "boosts" the node affinities[4] and, therefore, detects even small changes in the graphs (other distance measures, including ED, suffer from high similarity scores no matter how much the graphs differ), and
3. satisfies the desired properties $P1$-$P3$, as well as the informal property IP.

Step 3 For interpretability, we convert the distance (d) to a similarity measure (sim) via the formula $sim = \frac{1}{1+d}$. The result is bounded to the interval [0,1], as opposed to being unbounded $[0,\infty)$.

Notice that the distance-to-similarity transformation does *not* change the ranking of results in a nearest-neighbor query.

The straightforward algorithm, DeltaCon$_0$ (Algorithm 4.1), is to compute all the n^2 affinity scores of matrix $\mathbf{S}$ by simply using Equation 4.2. We can do the inversion using the Power Method or any other efficient method.

[3]Using p^{th} root instead of square root gives consistent results (for small values of p). Without loss of generality, we use the Matusita distance, which corresponds to $p = 2$.

[4]The node affinities are in $[0, 1]$, so the square root makes them bigger.

Algorithm 4.1: DeltaCon$_0$

```
INPUT: edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, i.e., $\mathbf{A}_1$ and $\mathbf{A}_2$
// $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, if $\mathcal{V}_1$ and $\mathcal{V}_2$ are the graphs' node sets
$\mathbf{S_1} = [\mathbf{I} + \epsilon^2\mathbf{D}_1 - \epsilon\mathbf{A}_1]^{-1}$          // $s_{1,ij}$: affinity/influence of
$\mathbf{S_2} = [\mathbf{I} + \epsilon^2\mathbf{D}_2 - \epsilon\mathbf{A}_2]^{-1}$          //node $i$ to node $j$ in $G_1$
$d(G_1, G_2)$ =RootED $(\mathbf{S}_1, \mathbf{S}_2)$
return $sim(G_1, G_2) = \frac{1}{1+d(G_1,G_2)}$
```

4.2.2.2 Speeding up: DeltaCon

Unfortunately, DeltaCon$_0$ is quadratic (n^2 affinity scores s_{ij} are computed by using the power method for the inversion of a sparse matrix) and thus not scalable. Thus, we look for means to speed up the algorithm.

We present a faster, linear algorithm, DeltaCon (Algorithm 4.2), which approximates DeltaCon$_0$ and differs in the first step. We still want each node to become a seed exactly once in order to find the affinities of the rest of the nodes to it; but, here, we have multiple seeds at once, instead of having one seed at a time. The idea is to randomly divide our node-set into g groups, and compute the affinity score of each node i to group k, thus requiring only $n \times g$ scores, which are stored in the $n \times g$ matrix $\mathbf{S}'$ ($g \ll n$). Intuitively, instead of using the $n \times n$ affinity matrix $\mathbf{S}$, we add up the scores of the columns that correspond to the nodes of a group, and obtain the $n \times g$ matrix $\mathbf{S}'$ ($g \ll n$). The score s'_{ik} is the affinity of node i to the k^{th} *group* of nodes ($k = 1, \ldots, g$).

Thus, we compute g final scores per node, which denote its affinity to every *group* of seeds, instead of every seed node that we had in Equation 4.2. With careful implementation, DeltaCon is linear on the number of edges and groups g – it takes ~ 160sec, on commodity hardware, for a 1.6-million-node graph. Once we have the reduced affinity matrices $\mathbf{S}'_1$ and $\mathbf{S}'_2$ of the two graphs, we use the RootED, to find the similarity between the $n \times g$ matrices of final scores, where $g \ll n$. The pseudocode of DeltaCon is given in Algorithm 4.2.

In an attempt to see how our random node partitioning algorithm in the first step fares with respect to more principled partitioning techniques, we used METIS [KK95]. Essentially, such an approach finds the influence of *coherent* subgraphs to the rest of the nodes in the graph – instead of the influence of randomly chosen nodes to the latter. We found that the METIS-based variant of our similarity method gave intuitive results for most small, synthetic graphs, but not for the real graphs. This is probably related to the lack of good edge-cuts on sparse real graphs, and also the fact that changes within a group manifest less when a group consists of the nodes belonging to a single community than randomly assigned nodes.

Next we give the time complexity of DeltaCon, as well as the relationship between the similarity scores of DeltaCon$_0$ and DeltaCon.

Algorithm 4.2: DeltaCon

```
INPUT: edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, i.e., $\mathbf{A}_1$ and $\mathbf{A}_2$, and
        $g$ (groups: # of node partitions)
$\{\mathcal{V}_j\}_{j=1}^{g}$ = random_partition($\mathcal{V}, g$)                    //$g$ groups
// estimate affinity vector of nodes $i = 1, \ldots, n$ to group $k$
for $k = 1 \rightarrow g$ do
    $\vec{s}_{0k} = \sum_{i \in \mathcal{V}_k} \vec{e}_i$
    solve $[\mathbf{I} + \epsilon^2 \mathbf{D}_1 - \epsilon \mathbf{A}_1] \vec{s}'_{1k} = \vec{s}_{0k}$
    solve $[\mathbf{I} + \epsilon^2 \mathbf{D}_2 - \epsilon \mathbf{A}_2] \vec{s}'_{2k} = \vec{s}_{0k}$
end for
$\mathbf{S}'_1 = [\vec{s}'_{11} \; \vec{s}'_{12} \; \ldots \; \vec{s}'_{1g}]$;  $\mathbf{S}'_2 = [\vec{s}'_{21} \; \vec{s}'_{22} \; \ldots \; \vec{s}'_{2g}]$
// compare affinity matrices $\mathbf{S}'_1$ and $\mathbf{S}'_2$
$d(G_1, G_2)$ =RootED $(\mathbf{S}'_1, \mathbf{S}'_2)$
return $sim(G_1, G_2) = \frac{1}{1+d(G_1,G_2)}$
```

Lemma 4.1: Linear Time Complexity of DeltaCon

The time complexity of DeltaCon, when applied in parallel to the input graphs, is linear on the number of edges in the graphs, i.e., $O(g \cdot max\{m_1, m_2\})$.

Proof. By using the Power Method [KKK+11], the complexity of solving Equation 4.1 is $O(m_i)$ for each graph $(i = 1, 2)$. The node partitioning needs $O(n)$ time; the affinity algorithm is run g times in each graph, and the similarity score is computed in $O(gn)$ time. Therefore, the complexity of DeltaCon is $O((g+1)n + g(m_1 + m_2))$, where g is a small constant. Unless the graphs are trees, $|\mathcal{E}_i| < n$, so the complexity of the algorithm reduces to $O(g(m_1 + m_2))$. Assuming that the affinity algorithm is run on the graphs in parallel, since there is no dependency between the computations, DeltaCon has complexity $O(g \cdot max\{m_1, m_2\})$. ∎

[KSV+15] contains more details about the relationships between DeltaCon and DeltaCon$_0$ scores. Specifically, one can use the Cauchy-Swartz inequality to show that DeltaCon upper bounds DeltaCon$_0$ given fixed input graphs G_1 and G_2. Intuitively, grouping the nodes blurs the influence information and makes the nodes seem more similar than originally. The proofs are excluded here as they are not relevant to this chapter's main focus. Summarily, DeltaCon is a faster version of DeltaCon$_0$ which computes node-to-group instead of node-to-node influences in order to speed up the computation to linear from quadratic complexity.

4.3 DeltaCon-Attr: Blame Attribution for Nodes and Edges

Thus far, we have broached the intuition and decisions behind developing DeltaCon for calculating graph similarity. Next, we show how this approach can be leveraged for the purposes of node and edge-based blame attribution. Practically, this equates to finding out *why* the graph changed the way it did and which changes were most important.

Equipped with this information, we can draw conclusions with respect to how certain changes impact graph connectivity and apply this understanding in a domain-specific context to assign blame. The resulting findings could also be useful for practitioners to instrument measures to prevent such changes in the future. Additionally, such a feature can be used to measure changes which have not yet happened in order to find information about which nodes and/or edges are most important for preserving or destroying connectivity (useful for stopping the spread of viruses on a network). In this section, we propose DeltaCon-Attr as a complementary approach to DeltaCon which enables node and edge-based blame attribution for this very purpose.

4.3.1 Algorithm Description

4.3.1.1 Node Attribution

Our first goal is to find the nodes which are mostly responsible for the difference between the input graphs. Let the affinity matrices $\mathbf{S}_1'$ and $\mathbf{S}_2'$ be precomputed. Then, the steps of our node attribution algorithm (Algorithm 4.3) can be summarized to:

Step 1 Intuitively, we compute the difference between the affinity of node v to the node groups in graph $\mathbf{A}$ and the affinity of node v to the node groups in graph $\mathbf{A_2}$. To that end, we use the same distance, RootED, that we applied to find the similarity between the whole graphs.

Given that the v_{th} row vector ($v \leq n$) of $\mathbf{S}_1'$ and $\mathbf{S}_2'$ reflects the affinity of node v to the remainder of the graph, the RootED distance between the two vectors provides a measure of the extent to which that node is a *culprit* for change — we refer to this measure as the *impact* of a node. Thus, culprits with comparatively high impact are ones that are most responsible for change between graphs.

More formally, we quantify the contribution of each node to the graph changes by taking the RootED distance between each corresponding pair of row vectors in $\mathbf{S}_1'$ and $\mathbf{S}_2'$ as w_v for $v = 1, \ldots, n$ per Equation 4.4.

$$w_v = \text{RootED}(\mathbf{S}_{1,v}', \mathbf{S}_{2,v}') = \sqrt{\sum_{j=1}^{g} (\sqrt{s_{1,vj}'} - \sqrt{s_{2,vj}'})^2}. \tag{4.4}$$

Step 2 We sort the scores in the $n \times 1$ node impact vector $\vec{w}$ in descending order and report the most important scores and their corresponding nodes.

Algorithm 4.3: DeltaCon-Attr: Node Attribution

```
INPUT: affinity matrices $\mathbf{S}'_1$, $\mathbf{S}'_2$
       edge files of $G_1(\mathcal{V}, \mathcal{E}_1)$ and $G_2(\mathcal{V}, \mathcal{E}_2)$, i.e., $\mathbf{A}_1$ and $\mathbf{A}_2$

for $v = 1 \to n$ do
  // If an edge adjacent to the node has changed, the node is responsible:
  if $\sum |\mathbf{A}_1(v,:) - \mathbf{A}_2(v,:)| > 0$ then
    $w_v = \text{RootED}(\mathbf{S}'_{1,v}, \mathbf{S}'_{2,v})$
  end if
end for
$[\vec{w}_{sorted}, \vec{w}_{sortedIndex}]$ = sortRows($\vec{w}$, 1, 'descend')  // sort rows of vector $w$ on column index 1
                                                    //(node impact score) by descending value
return $[\vec{w}_{sorted}, \vec{w}_{sortedIndex}]$
```

In reality, a practitioner might only want a partial ranking (the topmost influencers) from the entire node set. This can be achieved by using a simple energy-based heuristic (i.e.: reporting the top 80% of changes as per Fukunaga's heuristic [Fuk90]. In practice, we find that impact distributions in real graphs are skewed (although the distribution need not be 80-20).

4.3.1.2 Edge Attribution

Complementarily to the node attribution approach, we also developed an edge attribution method which ranks edge changes (additions and deletions) with respect to the graph changes based on the affinity scores. The steps of our edge attribution algorithm (Algorithm 4.4) are:

Step 1 We assign each changed edge incident to at least one node in the culprit set an impact score. This score is equal to the sum of impact scores for the nodes that the edge connects or disconnects.

Our goal here is to assign edges impact according to the degree that they affect the nodes that they touch. Since even the removal or addition of a single edge does not necessarily impact both incident nodes equally, we choose the sum of both nodes' scores as the edge impact metric. The intuition is that edges which touch two nodes of moderately high impact would be ranked higher than those which touch one node of high impact but another of low impact.

Step 2 We sort the edge impact scores in descending order and report the edges in order of importance.

Analysis of changed edges can reveal important discrepancies from baseline behavior. Specifically, a large number of added edges or removed edges with individually low impact is indicative of star formation or destruction, whereas one or a few added or removed edges with individually high impact are indicative of community expansion or reduction via addition or removal of certain key bridge edges.

Algorithm 4.4: DeltaCon-Attr: Edge Attribution

```
INPUT: adjacency matrices $\mathbf{A}_1$, $\mathbf{A}_2$, culprit set of interest $\vec{w}_{sortedIndex,1\ldots\text{index}}$ and node
impact scores $\vec{w}$

for $v = 1 \to$ length($w_{sortedIndex,1\ldots\text{index}}$) do
    srcNode = $w_{sortedIndex,v}$
    $\vec{r} = \mathbf{A}_{2,v} - \mathbf{A}_{1,v}$
    for $k = 1 \to n$ do
        destNode = $k$
        if $\vec{r}_k = 1$ then
            edgeScore = $w_{\text{srcNode}} + w_{\text{destNode}}$
            append row [srcNode, destNode, '+', edgeScore] to $\mathbf{E}$
        end if
        if $\vec{r}_k = -1$ then
            edgeScore = $w_{\text{srcNode}} + w_{\text{destNode}}$
            append row [srcNode, destNode, '-', edgeScore] to $\mathbf{E}$
        end if
    end for
end for
$\mathbf{E}_{sorted}$ = sortrows($\mathbf{E}$, 4, 'descend')  // sort rows of matrix $\mathbf{E}$ on column index 4 (edge impact
                                                       // score) by descending value
return $\mathbf{E}_{sorted}$
```

4.3.2 Scalability Analysis

Given precomputed $\mathbf{S}'_1$ and $\mathbf{S}'_2$ (precomputation is assumed since attribution can only be conducted after similarity computation), the node attribution component of DeltaCon-Attr is log-linear on the number of nodes, since n influence scores need to be sorted in descending order. In practice, the linear term is generally more expensive for runtime than the sorting operation, given the computational expense of calculating the Matusita distance between each pair of node vectors.

With the same assumptions with respect to precomputed results, the edge attribution portion of DeltaCon-Attr is also log-linear, but on the sum of edge counts, since $m_1 + m_2$ total possible changed edges need to be sorted. In practice, the number of edges needing to be sorted should be far smaller, as we only need to concern ourselves with edges which are incident to nodes in the culprit set of interest. Specifically, the cost of computing impact scores for edges is linear on the number of nodes in the culprit set k and the number of changed edges, but is again paired with the sorting cost.

4.4 Experiments

In this section, we evaluate DeltaCon-Attr on several synthetic "toy" graphs to ascertain obeyance of the properties proposed in subsubsection 4.2.1.4 as well as evaluate quantitative ranking and classification performance with respect to a competing method. We additionally report on DeltaCon-Attr's qualitative performance on a real dataset.

4.4.1 Quantitative Results

Graph A	Graph B	DeltaCon-Attr edges	DeltaCon-Attr nodes	CAD edges: δ for l = 5	CAD nodes	Properties
K5	mK5	✓	✓	✓	✓	
K5	m2K5	✓	✓	✓	✓	IP
B10	mB10 ∪ mmB10	✓	✓	✓	✓	P1, P2, IP
L10	mL10 ∪ mmL10	✓	✓	✓	5,6,4	P1, IP
S5	mS5	✓	✓	✓	1=5	P1 , P2
K100	mK100	✓	✓	✓	✓	
K100	w5K100	✓	✓	✓	✓	P3
mK100	w5K100	✓	✓	✓	✓	P3
K100	m3K100	✓	✓	✓	✓	P3, IP
K100	m10K100	✓	✓	(80,82)=(80,88)=(80,92)*	80,30,88=92*	P3, IP
P100	mP100	✓	✓	✓	✓	P1
w2P100	w5P100	✓	✓	✓	✓	P1, P3
B200	mmB200	✓	✓	✓	✓	P1
w20B200	m3B200	✓	✓	✓	✓	P1, P3, IP
S100	mS100	✓	✓	✓	1=4	P1, P2
S100	m3S100	✓	✓	✓	1,81=67=4	P1, P2, IP
wS100	m3S100	✓	✓	(1,4),(1,67),(1,81)	1,4=67,81	P1, P3, IP
Custom18	m2Custom18	✓	✓	(18,17),(10,11)	18,17,10,11	P1, P2
Custom18	m4Custom18b	✓	✓	✓	5=6,17=18	P1, P3

Table 4.2: **DeltaCon-Attr obeys all the required properties, while CAD does not.** Each row corresponds to a comparison between graph A and graph B, and evaluates the node and edge attribution of DeltaCon-Attr and CAD. The right order of edges and nodes is marked in Figures 4.1 and 4.2. We give the ranking of a method if it is different from the expected one.

We first test DeltaCon-Attr on a number of synthetically created and modified graphs, and compare it to the state-of-the-art methods. We perform two types of experiments: The first experiment examines whether the *ranking* of the culprit nodes by our method agrees with intuition.

In the second experiment, we evaluate DeltaCon-Attr's *classification* accuracy in finding culprits, and compare it to the best-performing competitive approach, CAD[5], which was introduced by Sricharan and Das [SD14] concurrently, and independently from us. CAD uses the idea of commute time between nodes to define the anomalousness of nodes/edges. In a random walk, the commute time is defined as the expected number of steps starting at i, before node j is visited and then node i is reached again. We give a qualitative comparison between DeltaCon-Attr and CAD in Section 4.5 (Node/Edge Attribution).

[5]CAD was originally introduced for finding culprit nodes and edges without ranking them. We extended the proposed method to rank the culprits.

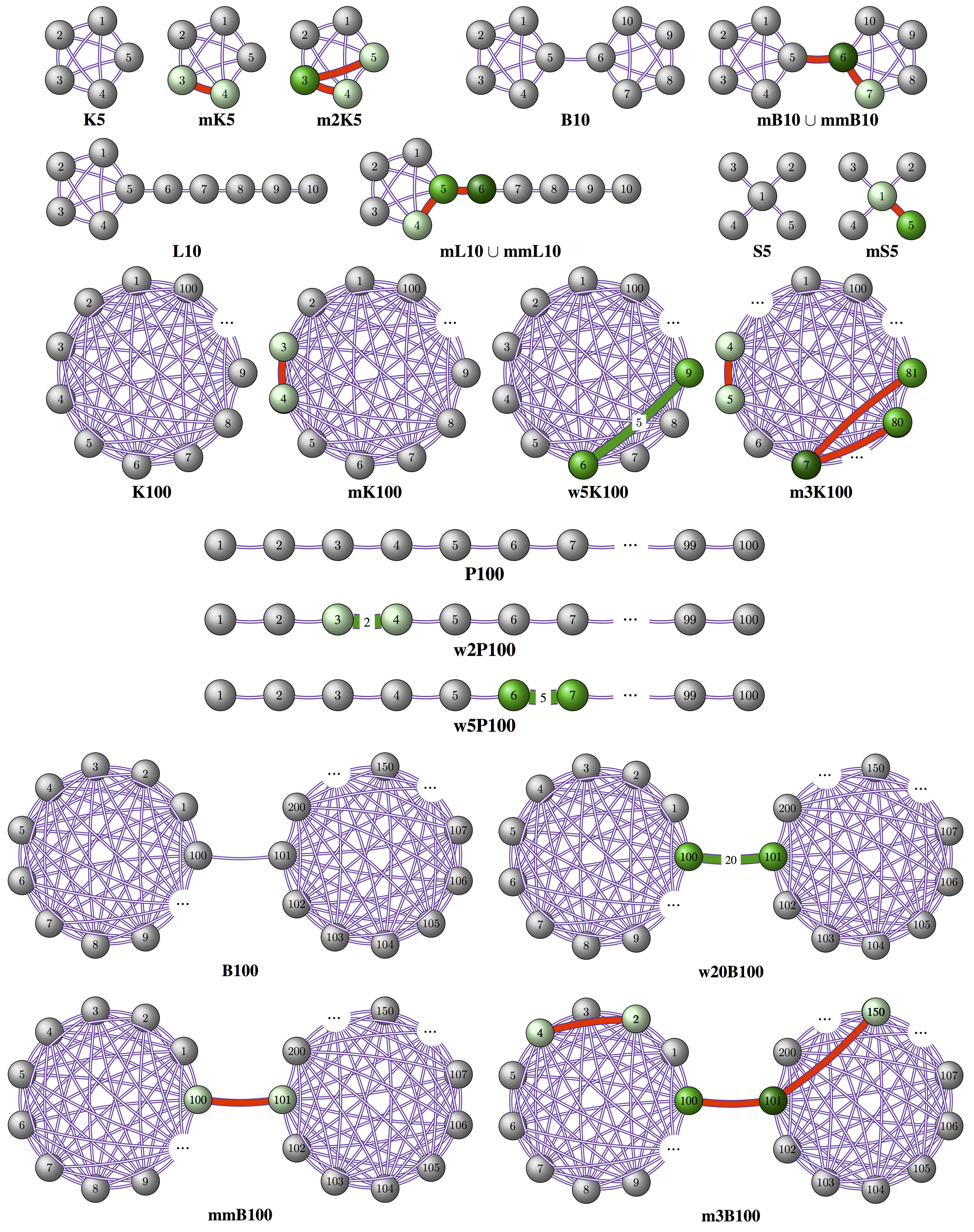


Figure 4.1: **DeltaCon-Attr respects properties P1-P3, and IP.** Nodes marked green are identified as the culprits for the change between the graphs. Darker shade corresponds to higher rank in the list of culprits. Removed and weighted edges are marked red and green, respectively.

4.4.1.1 Ranking Accuracy

We extensively tested DeltaCon-Attr on a number of synthetically created and modified graphs, and compared it with CAD. We note that CAD was designed to simply identify culprits

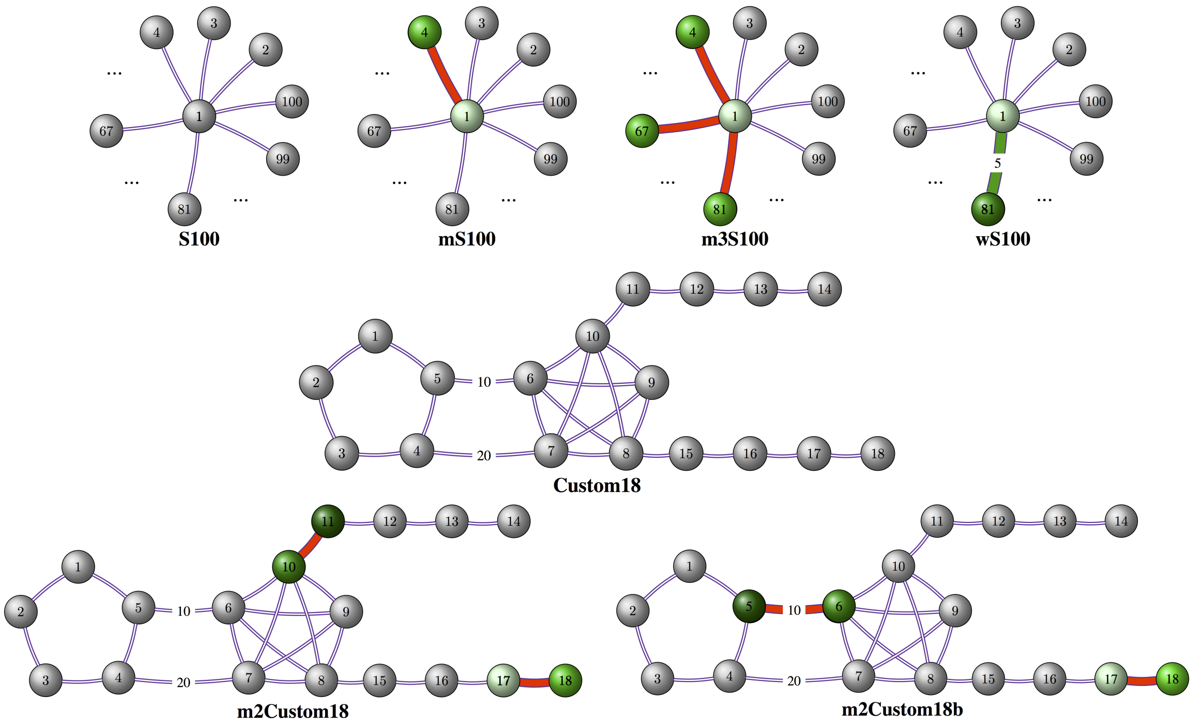


Figure 4.2: **[continued] DeltaCon-Attr respects properties P1-P3, and IP.** Nodes marked green are identified as the culprits for the change between the graphs. Darker shade corresponds to higher rank in the list of culprits. Removed edges are marked red.

in time-evolving graphs without ranking them. In order to compare it with our method, we adapted CAD so that it returns ranked lists of node and edge culprits: (i) We rank the culprit edges in decreasing order of edge score ΔE; (ii) To each node v, we attach a score equal to the sum of the scores of its adjacent edges, i.e., $\sum_{u \in N(v)} \Delta E((v,u))$, where $N(v)$ are the neighbors of v. Subsequently, we rank the nodes in decreasing order of attached score.

We give several of the conducted experiments in Table 4.2, and the corresponding graphs in Figures 4.1 and 4.2. Each row of the table corresponds to a comparison between graph A and graph B. The node and edge culprits that explain the main differences between the compared graphs are annotated in Figures 4.1 and 4.2. The darker a node is, the higher it is in the ranked list of node culprits. Similarly, edges that are adjacent to darker nodes are higher in the ranked list of edge culprits than edges that are adjacent to lighter nodes. If the returned ranked list agrees with the expected list (according to the formal and informal properties), we characterize the attribution of the method correct (checkmark). If there is disagreement, we provide the ordered list that the method returned. If two nodes or edges are tied, we use "=". For CAD we picked the parameter δ such that the algorithm returns 5 culprit edges and their adjacent nodes. Thus, we mark the returned list with "*" if CAD outputs 5 culprits while more exist. For each comparison, we also give the properties (last column) that define the order of the culprit edges and nodes.

Observation 4.1: Adherence to Desiderata

DeltaCon-Attr reflects the desired properties P1, P2, P3, and IP, while CAD fails to return the expected ranked lists of node and edge culprits in several cases.

Next we explain some of the comparisons that we present in Table 4.2:

- **K5-mK5**: The pair consists of a 5-node complete graph and the same graph with one missing edge, $(3, 4)$. DeltaCon-Attr considers nodes 3 and 4 top culprits, with equal rank, due to equivalent loss in connectivity. Edge $(3, 4)$ is ranked top, and is essentially the only changed edge. CAD finds the same results.
- **K5-m2K5**: The pair consists of a 5-node complete graph and the same graph with two missing edges, $(3, 4)$ and $(3, 5)$. Both DeltaCon-Attr and CAD consider node 3 the top culprit, because two of its adjacent edges were removed. Node 3 is followed by 4 and 5, which are tied since they are both missing one adjacent edge (Property IP). The removed edges, $(3, 4)$ and $(3, 5)$, are considered equally responsible for the difference between the two input graphs. We observe similar behavior in larger complete graphs with 100 nodes (K100, and modified graphs mK100, w5K100 etc.). In the case of K100 and m10K100[6], CAD does not return all 13 node culprits and 10 culprit edges because its parameter, δ, was set so that it would return at most 5 culprit edges[7].
- **B10-mB10 $\cup$ mmB10**: We compare a barbell graph of 10 nodes to the same graph that is missing both an edge from a clique, $(6, 7)$, and the bridge edge, $(5, 6)$. As expected, DeltaCon-Attr finds 6, 5 and 7 as top culprits, where 6 is ranked higher than 5, since 6 lost connectivity to both nodes 5 and 7, whereas 5 disconnected only from 6. Node 5 is ranked higher than 7 because the removal of the bridge edge is more important than the removal of $(6, 7)$ within the second clique (Property P1). CAD returns the same results. We observe similar results in the case of the larger barbell graphs (B200, mmB200, w20B200, m3B200).
- **L10-mL10 $\cup$ mmL10**: This pair of graphs corresponds to the lollipop graph, $L10$, and the lollipop variant, $mL10 \cap mmL10$, that is missing one edge from the clique, as well as a bridge edge. Nodes 6, 5 and 4 are considered the top culprits for the difference in the graphs. Moreover, 6 is ranked more responsible for the change than 5, since 6 lost connectivity to a more strongly connected component than 5 (Property P2). However, CAD ranks node 5 higher than node 6 despite the difference in the connectivity of the two components (violation of P2).
- **S5, mS5**: We compare a 5-node star graph, and the same graph missing the edge $(1, 5)$. DeltaCon-Attr considers 5 and 1 top culprits, with 5 ranking higher than 1, as the edge

[6] m10K100 is a complete graph of 100 nodes where we have removed 10 edges: (i) 6 of the edges were adjacent to node 80—$(80, 82)$, $(80, 84)$, $(80, 86)$, $(80, 88)$, $(80, 90)$, $(80, 92)$; (ii) 3 of the edges were adjacent to node 30—$(30, 50)$, $(30, 60)$, $(30, 70)$; and (iii) edge $(1, 4)$.

[7] The input graphs are symmetric. If edge (a, b) is considered culprit, CAD returns both (a, b) and (b, a), which have the same anomalousness score.

removal caused a loss of connectivity from node 5 to all the peripheral nodes of the star, $2, 3, 4$, and the central node, 1. CAD considers nodes 1 and 5 equally responsible, ignoring the difference in the connectivity of the components (violation of P2). Similar results are observed in the comparisons between the larger star graphs–S100, mS100, m3S100, wS100.

- **Custom18-m2Custom18**: The ranking of node culprits that DeltaCon-Attr finds is 11, 10, 18, and 17. The nodes 11 and 10 are considered more important than the nodes 18 and 17, as the edge removal $(10, 11)$ creates a large connected component and a small chain of 4 nodes, while the edge removal $(17, 18)$ leads to a single isolated node (18). Node 10 is higher in the culprit list than node 11 because it loses connectivity to a denser component. The reasoning is similar for the ranking of nodes 18 and 17. CAD does not consider the differences in the density of the components, and leads to a different ranking of the nodes.
- **Custom18-m2Custom18**: The ranking of node culprits that DeltaCon-Attr returns is 5, 6, 18, and 17. This is in agreement with properties P1 and P3, since the edge $(5, 6)$ is more important than the edge $(17, 18)$. Node 5 is more responsible than node 6 for the difference between the two graphs, as node 5 ends up having reduced connectivity to a denser component. This property is ignored by CAD, which thus results in different node ranking.

As we observe, in all the synthetic and easily controlled examples, the ranking of the culprit nodes and edges that DeltaCon-Attr finds agrees with intuition.

4.4.1.2 Classification Accuracy

To further evaluate the accuracy of DeltaCon-Attr in classifying nodes as culprits, we perform a simulation-based experiment and compare our method to CAD. Specifically, we set up a simulation similar to the one that was introduced in [SD14].

We sample 2000 points from a 2-dimensional Gaussian mixture distribution with four components, and construct the matrix $\mathbf{P} \in \mathcal{R}^{2000\times 2000}$, with entries $p(i, j) = \exp ||i - j||$, for each pair of points (i, j). Intuitively, the adjacency matrix $\mathbf{P}$ corresponds to a graph with four clusters that have strong connections within them, but weaker connections across them. By following the same process and adding some noise in each component of the mixture model, we also build a matrix $\mathbf{Q}$, and add more noise to it, which is defined as:

$$\mathbf{R}_{ij} = \begin{cases} 0 & \text{with probability } 0.95 \\ u_{ij} \sim \mathcal{U}(0, 1) & \text{otherwise,} \end{cases}$$

where $\mathcal{U}(0, 1)$ is the uniform distribution in $(0, 1)$. Then, we compare the two graphs, G_A and G_B, to each other, which have adjacency matrices $\mathbf{A} = \mathbf{P}$ and $\mathbf{B} = \mathbf{Q} + (\mathbf{R} + \mathbf{R}')/2$, respectively. We consider culprits (or anomalous) the inter-cluster edges for which $\mathbf{R}_{ij} \neq 0$, and the adjacent nodes. According to property P1, these edges are considered important (major culprits) for the difference between the graphs, as they establish more connections between loosely coupled clusters.

Conceptually, DeltaCon-Attr and CAD are similar because they are based on related methods [KKK+11] (Belief Propagation and Random Walk with Restarts, respectively). As shown in

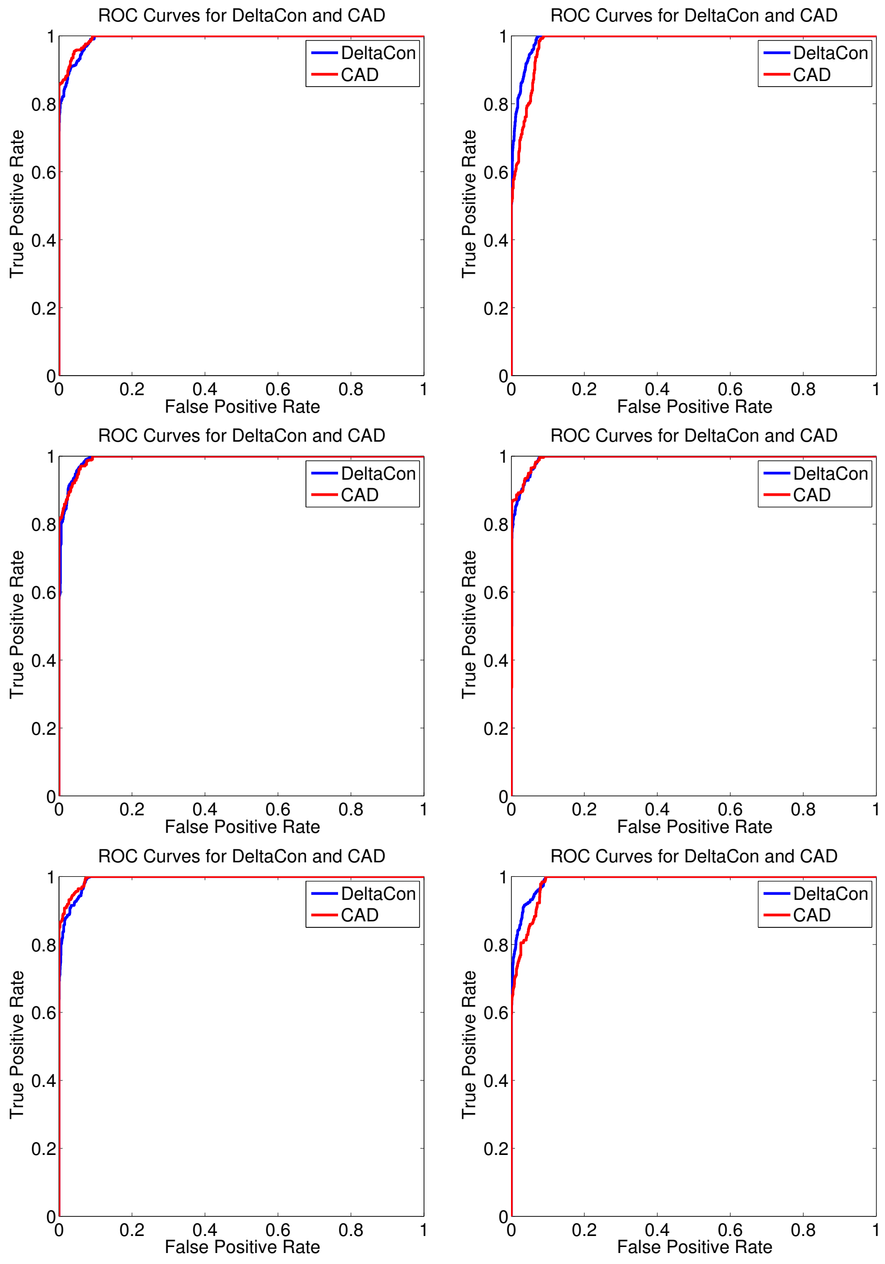


Figure 4.3: **DeltaCon-Attr ties the state-of-the-art with respect to accuracy.** Each plot shows the ROC curves for DeltaCon-Attr and CAD for different realizations of two synthetic graphs. The graphs are generated from points sampled from a 2-dimensional Gaussian mixture distribution with four components.

Figure 4.3, the simulation described above corroborates this argument, and the two methods have comparable performance – i.e., the areas under the ROC curves are similar for various realizations of the data described above. Over 15 trials, the AUC of DeltaCon-Attr and CAD is 0.9922 and 0.9934, respectively.

Observation 4.2: Accuracy on Graphs with Good Cuts

Both methods are very accurate in detecting nodes that are responsible for the differences between two highly-clustered graphs (Property P1).

4.4.1.3 Runtime

The amount of time taken for DeltaCon-Attr is trivial even for large graphs, given that the necessary affinity matrices are already in memory from the DeltaCon similarity computation. Specifically, node and edge attribution are log-linear on the nodes and edges, respectively, given that sorting is unavoidable for the task of ranking.

To compare the runtime of DeltaCon-Attr with the runtime of CAD, we perform the runtime experiment that the authors ran in [SD14]. Specifically, we generate sparse uniformly distributed random and symmetric matrices of $n = O(10^7)$ nodes, and sparsity level $\frac{1}{n}$. Over 10 trials, the combined runtime of DeltaCon and DeltaCon-Attr is 4.01 minutes on average on a less powerful machine[8] than the one used in [SD14], while the reported time for CAD is 5 minutes on average. Thus, our method is faster than CAD, while having comparable or better accuracy on small and large synthetic datasets.

4.4.2 Qualitative Results

We employed DeltaCon-Attr to analyze the ENRON dataset, which consists of emails sent among employees at ENRON over a span of more than two years. The dataset was introduced in [KY04] and contains 36,692 nodes representing email addresses both internal and external to Enron along with 367,662 edges representing e-mail communications. Of these, we ignored e-mail addresses outside of the Enron corporate network, restricting our focus to employees. Furthermore, we restrict application of DeltaCon-Attr to the months of May 2001 and February 2002, which are the most anomalous months according to DeltaCon applied on a month-to-month scale. Based on the node and edge culprit rankings produced as a result, we drew several real-world conclusions as listed below, involving key players in the ENRON financial scandal.

May 2001:

- Top Influential Culprit: John Lavorato, the former head of Enron's trading operations and CEO of Enron America, connected to $\sim$50 new nodes in this month.
- Second Most Influential Culprit: Andy Zipper, VP of Enron Online, maintained contact with all those from the previous month, but also connected to 12 new people.

[8]The source code of [SD14] is not yet available, while we have implemented their naïve algorithm which is cubic. To make a more fair comparison between the two methods, we reproduced the experiment that the authors described, ran only our method, and compared the runtime of DeltaCon to the reported runtime of CAD. For this experiment we used a 64-bit 2.8GHz *single* quad core AMD Opteron (tm) Processor 854 with 32GB RAM, while the authors of CAD used a 64-bit 2.3 GHz *dual* quad core Dell Precision T7500 desktop with 32GB RAM.

- Third Most Influential Culprit: Louise Kitchen, another employee (President of ENRON Online) lost 5-6 connections and made 5-6 connections. Most likely, some of the connections she lost or made were very significant in terms of expanding/reducing her office network.

February 2002:

- Top Influential Culprit: Liz Taylor lost 51 connections this month but made no new ones - it is reasonable to assume that she likely quit the position or was fired.
- Second Most Influential Culprit: Louise Kitchen (third culprit in May 2001) made no new connections, but lost 22 existing ones.
- Third Most Influential Culprit: Stan Horton (CEO of Enron Transportation) made 6 new connections and lost none. Some of these connections are likely significant in terms of expanding his office network.
- Fourth, Fifth and Sixth Most Influential Culprits: Employees Kam Keiser, Mike Grigsby (former VP for Enron's Energy Services) and Fletcher Sturm (VP) all lost many connections and made no new ones. Their situations were likely similar to those of L. Taylor and L. Kitchen.

4.5 Related Work

We categorize relevant prior literature into two main areas: anomaly detection, and blame attribution. We give the related work in each area, and mention what sets our method apart.

Anomaly Detection. Anomaly detection in static graphs has been studied using various data mining and statistical techniques [ATK14, KC10, ACK+12, LKKF13, KLKF14]. Detection of anomalous behaviors in time-evolving networks is more relevant to our work, and is covered in the surveys [ATK14, RSK+15]. A non-inclusive list of works on temporal graph anomaly detection follows. [MGF11], [KPF12] and [MWP+14] employ tensor decomposition to identify anomalous substructures in graph data in the context of intrusion detection. Henderson et al. propose a multi-level approach for identifying anomalous behaviors in volatile temporal graphs based on iteratively pruning the temporal space using multiple graph metrics [HERF+10]. CopyCatch [BXG+13] is a clustering-based MapReduce approach to identify lockstep behavior in Page Like patterns on Facebook. Akoglu and Faloutsos use local features and the node eigen-behaviors to detect points of change – when many of the nodes behave differently –, and also spot nodes that are most responsible for the change point [AF10]. Finally, [SD14] monitors changes in the commute time between all pairs of nodes to detect anomalous nodes and edges in time-evolving networks. All these works use various approaches to detect anomalous behaviors in dynamic graphs, though they are not based on the similarity between graphs, which is the focus of our work.

Blame Attribution. Some of the anomaly detection methods discover anomalous nodes, and other anomalous structures in the graphs. In a slightly different context, a number of techniques have been proposed in the context of node and edge importance in graphs. PageRank, HITS

[Kle99b] and betweenness centrality (random-walk-based [New05] and shortest-path-based [Fre77]) are several such methods for the purpose of identifying important nodes. [TPER+12] proposes a method to determine edge importance for the purpose of augmenting or inhibiting dissemination of information between nodes. To the best of our knowledge, this and other existing methods focus only on identifying important nodes and edges in the context of a single graph. In the context of anomaly detection, [AF10] and [SD14] detect nodes that contribute mostly to change events in time-evolving networks.

Among these works, the most relevant to ours are the methods proposed by [AF10] and [SD14]. The former –which is based on [IKIK04]– extracts node features, computes an "eigen"-behavior per node, and spots changes each node's behavior over time. Therefore, the method relies on the selection of features (*e.g.*, in-degree, out-degree, edge weights, number of triangles). Moreover, because of the focus on local egonet features, it may not distinguish between small and large changes in time-evolving networks, and it also tends to return a large number of false positives [SD14]. At the same time, and independently from us, Sricharan and Das proposed CAD [SD14], a method which defines the anomalousness of edges based on the commute time between nodes. The commute time is the expected number of steps in a random walk starting at i, before node j is visited and then node i is reached again. This method is closely related to DeltaCon-Attr as Belief Propagation and Random Walks with Restarts (the core idea behind CAD) are equivalent under certain conditions [KKK+11]. However, the methods work in different directions: DeltaCon-Attr first identifies the most anomalous nodes, and then defines the anomalousness of edges as a function of the outlierness of the adjacent nodes; CAD first identifies the most anomalous edges, and then defines all their adjacent nodes as anomalous without ranking them. Our method does not only find anomalous nodes and edges in a graph, but also ranks them in decreasing order of anomalousness (which can be used for guiding attention to important changes).

4.6 Conclusion

In this work, we tackled the problem of node- and edge-based blame attribution between two graphs with known node correspondence. To this end, we proposed DeltaCon-Attr, an effective and scalable blame attribution approach which leverages the DeltaCon graph similarity algorithm and adheres to a number of intuitive desiderata such as edge importance, edge "submodularity," weight awareness and focus awareness. Our approach meets these criteria while other state-of-the-art competitors do not. We demonstrates intuitiveness and adherence to the aforementioned properties on a variety of synthetic graphs, and further demonstrate comparable anomaly detection performance to state-of-the-art approaches while also maintaining a shorter runtime. Finally, we show some qualitative results and interesting findings upon applying DeltaCon-Attr to the ENRON e-mail dataset which signal abnormal changes in connectivity across key months of the ENRON scandal.

Based on content published in [LSK15, LSSK]

Chapter 5

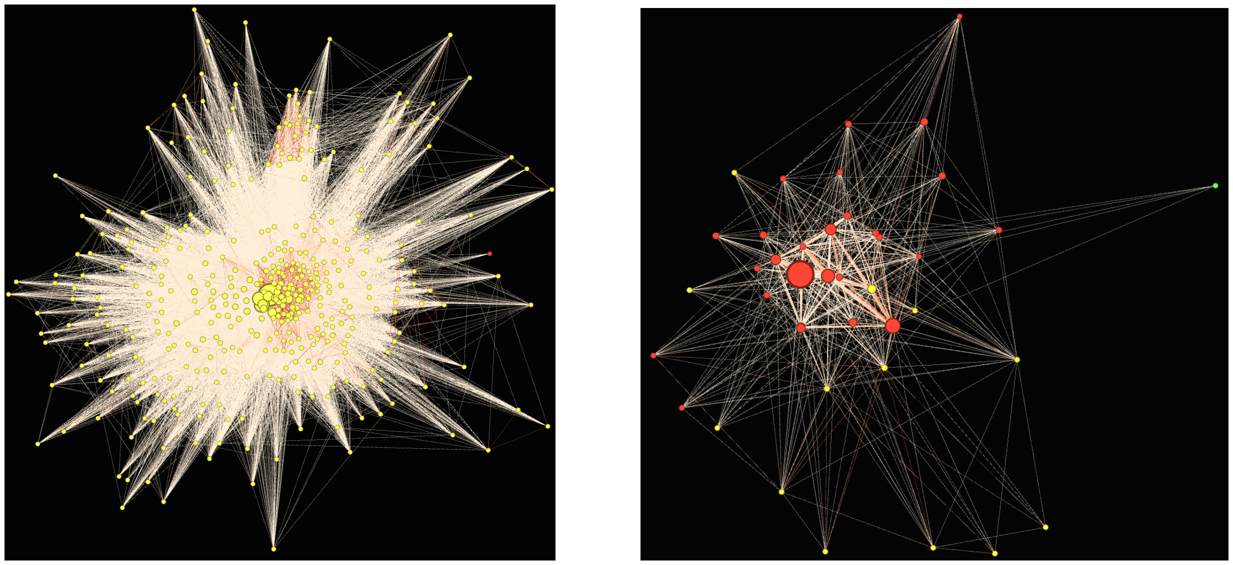
CONDENSE: Reducing Large Graphs to Small Supergraphs

Harnessing the power of various graph decomposition methods for summarization.

Given the increasing size and scale of graphs, summarizing and routing attention to their most important and relevant components is a key task. In this chapter, we propose a new state-of-the-art graph summarization algorithm called CONDENSE which summarizes an input graph using approximate "super-graphs" conditioned on a set of diverse, predefined structural patterns. Our method can incorporate a variety of summary-assembly methods including traditional clustering and partitioning algorithms, efficiently integrate and automatically annotate their outputs, and produce interpretable and sparse visualizations of large graphs.

5.1 Introduction

In an era of continuous generation of large amounts of data, summarization techniques are increasingly crucial as they abstract away noise, help uncover patterns, and hence inform human decision processes. In this paper, we focus on the summarization of graphs, which are powerful structures that capture a number of phenomena from communication between people to interactions between neurons in our brains. Graph summarization methods lead to the reduction of data volume, speedup of graph algorithms, improved storage and query time, and interactive visualization. The graph mining community has mainly studied summarization techniques for the structure of static, plain graphs [CKL$^+$09, NRS08] and to a smaller extent, methods for attributed or dynamic networks [SKZ$^+$15].



(a) Prior work [KKVF14a]. (b) Our method: ConDeNSe-Step.

Figure 5.1: `AS-Oregon` **supergraphs: ConDeNSe generates simpler and more compact supergraphs.** Yellow, red, and green nodes for stars, cliques, and bipartite cores, respectively.

We focus on summarizing the structure of a given large-scale network by selecting a small set of its most informative structural patterns. Inspired by recent work [NRS08, KKVF14a], we formulate graph summarization as an information-theoretic optimization problem in search of local structures that collectively minimize the description length of the graph.

We introduce ConDeNSe (CONditional Diversified Network Summarization), a unified, edge-overlap-aware graph summarization method that summarizes a given graph with approximate "supergraphs" conditioned on diverse, predefined structural patterns. An example is shown in Figure 5.1, where the (super)nodes in subfigure 5.1b correspond to *sets of nodes* in the original graph. Specifically, the predefined patterns include structures that have well-understood graph-theoretical properties and have been found in many real-world graphs [kle99a, AGMF14, FFF99, PSS$^+$10]: cliques, stars, bipartite cores, chains, and patterns with skewed degree distribution. Our approach selects the patterns that minimize the description of the graph in terms of number of bits. Our work effectively addresses three main shortcomings of prior summarization work [KKVF14a], namely: (i) heavy dependence on the structural pattern discovery method and its intrinsic bias towards star-like structures; (ii) inability to handle edge-overlapping patterns in the summary, leading to redundancy; and (iii) heuristic dependence on the order in which structures are considered for inclusion in the summary. Our proposed unified approach effectively handles these issues and results in robust and compact summaries with $5-10\times$ fewer structural patterns (or supernodes), up to 50% better compression and better node coverage of the input graph.

ConDeNSe has three main modules that tackle the above-mentioned shortcomings: (i) A unified structural pattern discovery module leverages the strengths of various popular graph clustering methods (e.g., Louvain [BGLL08], METIS [KK99]) to address the structural biases that each method introduce in the graph summary; (ii) A Minimum Description Length-based (MDL) formulation with a penalty term effectively minimizes redundancy in edge coverage by the structural patterns included in the summary, thereby promoting higher node coverage. This term is paramount when the candidate structural patterns have significant edge overlap, such as in the case of our unified structure discovery module; (iii) An iterative, multi-threaded, and divide-and-conquer-based summary assembly module reduces even more bias during the summary creation process by being *independent* of the order in which the candidate structural patterns are considered. This parallel module is up to $53\times$ faster than its serial version (on a 6-core machine). Additionally, we show how ConDeNSe can be further used for visual interpretation and summarization of large graphs with an approximate supergraph creation module which depicts a large graph concisely using few structural (possibly overlapping) "supernodes" and "superedges" between them.

Our contributions in this paper are as follows:

• **Approach**: We introduce ConDeNSe, an effective unified, edge-overlap-aware graph summarization approach with a powerful parallel summary assembly module (k-Step) that creates compact and easy-to-understand graph summaries with high node coverage and low redundancy.

• **Novel Metric**: We propose a way to leverage ConDeNSe as a proxy to compare graph clustering methods with respect to their summarization performance on large, real-world graphs, complementing the usual evaluation metrics in the related literature (e.g., modularity, conductance).

• **Experiments**: We present a thorough empirical analysis on real networks to evaluate the summary quality and runtime, and study the properties of seven clustering methods.

5.2 Related Work and Background

Our work is related to graph summarization methods, MDL, and graph clustering. We review each of these topics in turn.

Graph Summarization. Most research efforts focus on plain graphs and can be broadly classified as group-based [LT10, RGM03], compression-based [CKL+09, NRS08], simplification-based, influence-based, and pattern-based [CH94]. Dynamic graph summarization has been studied to a much smaller extent [SKZ+15]. Most related to our work are the ideas of node grouping and graph compression. Built on these ideas, two representative methods, MDL-summarization [NRS08] and VoG [KKVF14a], are MDL-based summarization methods that compress the graphs by finding near-structures (e.g., (near-) cliques, (near-) bipartite cores). MDL-summarization, which iteratively combines neighbors into supernodes as long as it helps with minimizing the compression cost, includes mostly cliques and cores in the summaries, and has high runtime complexity. On the other hand, VoG finds structures by employing SlashBurn [KF11a] (explained below) and hence is particularly biased towards stars. Moreover, it creates summaries (i.e., lists of structures) using a greedy heuristic on a pre-ordered set of structures (cf. Section 5.4.3). Unlike these methods, CONDENSE performs ensemble pattern discovery, handles edge-overlapping structures, and its summary assembly is robust to the structure ordering. Thus, it leads to more compact and less biased summaries, creates approximate and easy-to-understand supergraphs, and can be used as a proxy to evaluate clustering methods in a novel way.

MDL in Graph Mining. Many data mining problems are related to summarization and pattern discovery, and, thus, to Kolmogorov complexity [FM07], which can be practically implemented by the MDL principle [Ris83]. Applications include clustering [CV05], community detection [CPMF04], pattern discovery in static and dynamic networks [KKVF14a, SKZ$^+$15], and more.

Graph Clustering. Graph clustering and community detection are of great interest to many domains, including social, biological, and web sciences [GN02, BKM$^+$08, For10]. Here, we leverage several graph clustering methods to obtain diversified graph summaries, since each method is biased toward certain types of structures, such as cliques and bipartite cores [BGLL08, KK99, YL13] or stars [KF11a]. Unlike existing literature [LLM10] where clustering methods are compared with respect to classic quality measures, we also propose to use CONDENSE as a vessel to evaluate the methods' summarization power. We leverage seven decomposition methods, which we compare quantitatively in Table 5.1:

• **SLASHBURN** [KF11a] is a node reordering algorithm initially developed for graph compression. It performs two steps iteratively: (i) It removes high-centrality nodes from the graph; (ii) It reorders nodes such that high-degree nodes are assigned the lowest IDs and nodes from

	SLASHBURN [KF11a]	LOUVAIN [BGLL08]	SPECTRAL [Hes]	METIS [KK99]	HYCOM [AGMF14]	BIGCLAM [YL13]	KCBC [LSK15]
Overlapping Clusters	✔	✘	✘	✘	✔	✔	✔
Cliques	Many	Many	Many	Many	Some	Many	Many
Stars	Many	Some	Some	Some	Many	Some	Some
Bipartite Cores	Some	Few	Many	Some	Some	Few	Few
Chains	Few	Few	Few	Few	Few	Few	Few
Hyperbolic Structures	Few	Few	Few	Few	Many	Few	Few
Complexity	$O(t(m + n \log n))$	$O(n \log n)$	$O(m + nk)$	$O(m \cdot k)$	$O(k(m + h \log h^2 + hm_h))$	$O(d \cdot n \cdot t)$	$O(t(m + n))$
Summarization Power	Excellent	Very Good	Good	Good	Poor	Good	Poor

Table 5.1: Qualitative comparison of the graph clustering techniques included in CONDENSE. Symbols: n = number of nodes, m = number of edges, k = number of clusters/partitions, t = number of iterations, d = average degree, $h(m_h)$ = number of nodes (edges) in hyperbolic structure.

disconnected components get the highest IDs. The process is repeated on the giant connected component. We leverage this process by identifying structures from the egonet of each high-centrality node, and the disconnected components, as subgraphs.

• **Louvain** [BGLL08] is a modularity-based partitioning method for detecting hierarchical community structure. The method is iterative: (i) Each node is placed in its own community. Then, the neighbors j of each node i are considered, and i is moved to j's community if the move produces the maximum modularity gain. The process is applied repeatedly until no further gain is possible. (ii) A new graph is built whose supernodes represent communities, and superedges are weighted by the sum of weights of links between the two communities. The algorithm typically converges in a few passes.

• **Spectral** clustering refers to a class of algorithms that utilize eigendecomposition to identify community structure. We utilize one such spectral clustering algorithm [Hes], which partitions a graph by performing k-means clustering on the top-k eigenvectors of the input graph. The idea behind this clustering is that nodes with similar connectivity have similar eigen-scores in the top-k vectors and form clusters.

• **METIS** [KK99] is a cut-based k-way multilevel graph partitioning scheme based on multilevel recursive bisection (MLRB). Until the graph size is substantially reduced, it first coarsens the input graph by grouping nodes into supernodes iteratively such that the edge-cut is preserved. Next, the coarsened graph is partitioned using MLRB, and the partitioning is projected onto the original input graph G through backtracking. The method produces k roughly equally-sized partitions.

• **HyCoM** [AGMF14] is a parameter-free algorithm that detects communities with hyperbolic structure. It approximates the optimal solution by iteratively detecting important communities. The key idea is to find in each step a single community that minimizes an MDL-based objective function given the previously detected communities. The iterative procedure consists of three steps: community candidates, community construction, and matrix deflation.

• **BigClam** [YL13] is a scalable overlapping community detection method. It is built on the observation that overlaps between communities are densely connected. By explicitly modeling the affiliation strength of each node-community pair, the latter is assigned a nonnegative latent factor which represents the degree of membership to the community. Next, the probability of an edge is modeled as a function of the shared community affiliations. The identification of network communities is done by fitting BigClam to a given undirected network G.

• **KCBC** [LSK15] is inspired by the k-cores algorithm [GTV11] that unveils densely connected structures. A k-core is a maximal subgraph for which each node is connected to at least k other nodes. KCBC iteratively removes k-cores starting by setting k equal to the maximum core number (max value k for which the node is present in the resulting subgraph) across all nodes. Each connected component in the induced subgraphs is identified as a cluster, and is removed from the original graph. The process is repeated on the remaining graph.

Other clustering methods that we considered (e.g., Weighted Stochastic Block Model or WSBM) are not included in CONDENSE due to scalability. For instance, WSBM took more than a week to finish on our smallest dataset.

5.3 CONDENSE: Proposed Model

We formulate the graph summarization problem as a graph compression problem. Let $G(\mathcal{V}, \mathcal{E})$ be a graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, without self-loops. The Minimum Description Length (MDL) problem, which is a practical version of Kolmogorov Complexity [FM07], aims to find the best model M in a given family of models $\mathcal{M}$ for some observed data $\mathcal{D}$ such that it minimizes $L(M) + L(\mathcal{D}|M)$, where $L(M)$ is the description length of M in bits and $L(\mathcal{D}|M)$ is the description length of $\mathcal{D}$ which is encoded by the chosen model M (see Chapter 2 for more background on MDL). Table 5.2 provides the definitions of the recurrent symbols used in this section.

We consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω. One option is to populate Ω with the frequent patterns that occur in the input graph (in a data-driven manner), but frequent subgraph mining is NP-complete and does not scale well. Moreover, even efficient *approximate* approaches are not applicable to unlabeled graphs and can only handle small graphs with a few tens or hundreds

Notation	Description
$G(\mathcal{V}, \mathcal{E})$, $\mathbf{A}$	graph, and its adjacency matrix
$\mathcal{V}$, $n = \|\mathcal{V}\|$	node-set and number of nodes of G, resp.
$\mathcal{E}$, $m = \|\mathcal{E}\|$	edge-set and number of edges of G, resp.
k	# of clusters or communities or patterns
t	# of iterations
h, m_h	size of hyperbolic community, and # of edges in it, resp.
d	average degree of nodes in G
h_{slash}	# of hub nodes to slash per iteration in SLASHBURN
fc, bc, st, ch, hs	full clique, bipartite core, star, chain, hyperbolic structure, resp.
$\|fc\|$, $\|bc\|$, $\|st\|$, $\|ch\|$, $\|hs\|$	number of nodes in the corresponding structure
Ω	predefined set of structural pattern types
M	a model or summary for G
s	structure in M
$\|S\|$, $\|s\|$	cardinality of set S and number of nodes in s, resp.
$\|\|s\|\|$, $\|\|s\|\|'$	# existing and non-existing edges of $\mathbf{A}$ that s describes
$\mathbf{E}$	error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, where $\oplus$ is exclusive OR
$\mathbf{O}$	edge-overlap penalty matrix
$L(G, M)$	# of bits to describe model M, and G using M
$L(M), L(O), L(s)$	# of bits to describe M, the edge overlap O, and structure s

Table 5.2: Major symbols and definitions.

of nodes. To circumvent this problem, we choose set Ω with five patterns that are common in real-world static graphs [kle99a, AGMF14], correspond to interesting real behaviors, and can (approximately) describe a wide range of structural patterns: stars (*st*), *full* cliques (*fc*), bipartite cores (*bc*), chains (*ch*), and hyperbolic structures with skewed degree distribution (*hs*). Under the MDL principle, any approximate structures (e.g., near-cliques) can be easily encoded as their corresponding exact structures (e.g., *fc*) with some errors. Since many communities have hyperbolic structure [AGMF14] and it cannot be expressed as a simple composition of the other structural patterns in Ω, we consider it separately. Motivated by real-world discoveries, we focus on structures that are commonly found in networks, but our framework is not restricted to them; it can be readily extended to other, application-dependent types of structures as well.

Formally, we tackle the following problem:

Problem 5.1: Overlap-Aware Summarization

Given a graph G with adjacency matrix $\mathbf{A}$ and structural pattern types Ω, we seek to find the model M that minimizes the encoding length of the graph and the redundancy in edge coverage:

$$L(G, M) = L(M) + L(\mathbf{E}) + L(\mathbf{O}) \tag{5.1}$$

where $\mathbf{M}$ is $\mathbf{A}$'s approximation induced by M, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix to correct for edges that were erroneously described by M, $\oplus$ is exclusive OR, and $\mathbf{O}$ is the edge-overlap matrix to penalize edges covered by many patterns.

Model M induces a supergraph with each $s \in M$ as an (approximate) supernode, and weighted superedges between them. Before we further formalize the task of encoding the model, the error matrix, and the edge-overlap penalty matrix, we provide a visual illustration of our MDL objective.

An Illustrative Example. *Figure 5.2 shows the original adjacency matrix* $\mathbf{A}$ *of an input graph, which is encoded as (i)* $\mathbf{M}$ *(the matrix that is induced by the model* M*), and (ii) the error matrix* E *(which captures additional/missing edges that are not properly described in* M*). In this example, there are 6 structures in the model (from the top left corner to the bottom right corner: a star, a large clique, a small clique, a bipartite core, a chain, and a hyperbolic structure), where the cliques and the bipartite core have overlapping nodes and edges.*

5.3.1 Encoding the Model

To fully describe a model $M \in \mathcal{M}$ for the input graph G, we encode it as $L(M)$:

$$L(M) = L_{\mathbb{N}}(|M| + 1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1} + \sum_{s \in M} \big(-\log \Pr(x(s) \mid M) + L(s) \big) \tag{5.2}$$

where in the first two terms we encode the number of structural patterns in M using Rissanen's optimal encoding for integers [Ris83] and the number of patterns per type in Ω, respectively.

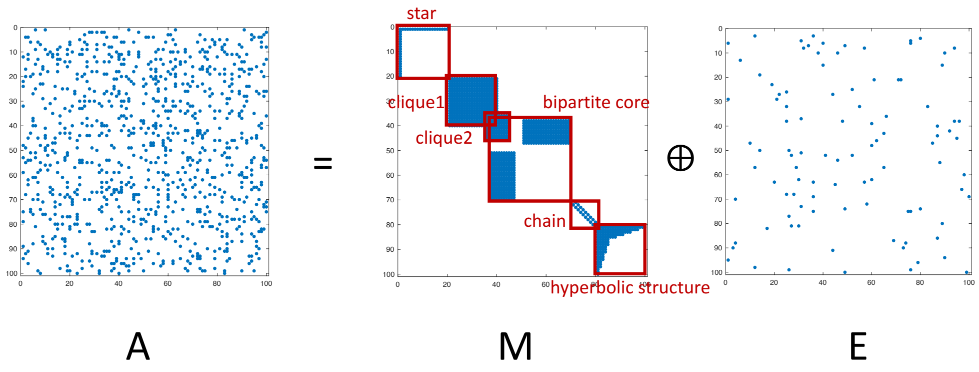


Figure 5.2: **An illustration of MDL encoding of a toy graph.**

Then, for each structure $s \in M$, we encode its type $x(s)$ using optimal prefix codes [CT06], and its connectivity $L(s)$. Next, we introduce the MDL encoding per type of structure in Ω.

• **Stars:** A star consists of a "hub" node connected to two or more "spoke" nodes. We encode it as:

$$L(st) = L_{\mathbb{N}}(|st| - 1) + log_2 n + log_2 \binom{n-1}{|st|-1} \tag{5.3}$$

where we encode in order the number of spokes, the hub ID (we identify it out of n nodes using an index over the combinatorial number system), and the spoke IDs.

• **Cliques:** A clique is a densely connected set of nodes with:

$$L(fc) = L_{\mathbb{N}}(|fc|) + log_2 \binom{n}{|fc|} \tag{5.4}$$

where we encode its number of nodes followed by their IDs.

• **Bipartite Cores:** A bipartite core consists of two non-empty sets of nodes, L and R, which have edges only between them, and $L \cap R = \emptyset$. Stars are a special case of bipartite cores with $|L| = 1$. The encoding cost is given as:

$$L(bc) = L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2 \binom{n}{|L|} + log_2 \binom{n}{|R|}, \tag{5.5}$$

where we encode the number of nodes in L and R followed by the node IDs in each set.

• **Chains:** A chain is a series of nodes that are linked consecutively–e.g. node-set $\{a, b, c, d\}$ in which a is connected to b, b is connected to c, and c is connected to d. Its encoding cost, $L(ch)$, is:

$$L(ch) = L_{\mathbb{N}}(|ch| - 1) + \sum_{i=1}^{|ch|} log_2(n - i + 1) \tag{5.6}$$

where we encode its number of nodes, followed by their node IDs in order of connection.

• **Hyperbolic Structures:** A hyperbolic structure or community [AGMF14] has skewed degree distribution which often follows a power law with exponent between -0.6 and -1.5. The encoding length of a hyperbolic structure hs is given as:

$$L(hs) = r + L_{\mathbb{N}}(|hs|) + \log_2 \binom{n}{|hs|} + \log_2(|\mathbf{A}(hs)|) + ||hs||l_1 + ||hs||' l_0 \tag{5.7}$$

where we first encode the power-law exponent (using Rissanen's encoding [Ris83] for the integer part, the number of decimal values, and the decimal part) with r bits, followed by the number of nodes and their IDs. Then, we encode the number of edges in the structure (=$|\mathbf{A}(hs)|$), and use optimal prefix codes, l_0, l_1, for the missing ($||hs||'$) and present ($||hs||$) edges, respectively. Specifically, $l_1 = -\log((||hs||/(||hs|| + ||hs||'))$, and l_0 is defined similarly.

5.3.2 Encoding the Errors

Given that M is a summary, and $\mathbf{M}$ is only an approximation of $\mathbf{A}$, we also need to encode errors of the model. For instance, a near-clique is represented as a full clique in the model, and, thus, contributes some edges to the error matrix (i.e., the missing edges from the real data). We encode the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ in two parts, $\mathbf{E}^+$ and $\mathbf{E}^-$, since they likely follow different distributions [KKVF14a]. The former encodes the edges induced by M which were not in the original graph, and the latter the original edges that are missing in M:

$$\begin{aligned} L(\mathbf{E}^+) &= log_2(|\mathbf{E}^+|) + ||\mathbf{E}^+||l_1 + ||\mathbf{E}^+||' l_0 && (5.8) \\ L(\mathbf{E}^-) &= log_2(|\mathbf{E}^-|) + ||\mathbf{E}^-||l_1 + ||\mathbf{E}^-||' l_0 && (5.9) \end{aligned}$$

where we encode the number of 1s in $\mathbf{E}^+$ (or $\mathbf{E}^-$), followed by the actual 1s and 0s using optimal prefix codes (as before).

5.3.3 Encoding the Edge-Overlap Penalty

Several of the graph decomposition methods that we consider (e.g., SlashBurn, KCBC in Table 5.1) generate edge-overlapping patterns. The MDL model we have presented so far naturally handles node overlaps—if two structures consist of the same large set of nodes, only one of the them will be chosen during the encoding cost minimization process, because their combination would lead to higher encoding cost. However, up to this point, the model considers a binary state for each edge: that is, an edge is described by the model M, or not described by it. This can lead to summaries with high redundancy in edge coverage and low node coverage, as excessively repeated edge coverage is not penalized.

To explicitly handle extensive edge overlaps in the graph summaries, we add an extra penalty term, $L(\mathbf{O})$, in the optimization function in Equation (5.1). We introduce the matrix $\mathbf{O}$, which maintains the number of times each edge is described by M, i.e., the number of selected structures in which the edge occurs. We encode the description length of $\mathbf{O}$ as:

$$L(\mathbf{O}) = \log_2(|\mathbf{O}|) + ||\mathbf{O}||l_1 + ||\mathbf{O}||'l_0 + \sum_{o \in \mathcal{E}(\mathbf{O})} L_{\mathbb{N}}(|o|) \tag{5.10}$$

where we first encode the number of distinct overlaps, and then use the optimal prefix code to encode the number of the present and missing entries in $\mathbf{O}$. As before, l_0 and l_1 are the lengths of the optimal prefix codes for the present and missing entries, respectively. Finally, we encode the weights in $\mathbf{O}$ using the optimal encoding for integers $L_{\mathbb{N}}$ [Ris83]. We denote with $\mathcal{E}(\mathbf{O})$ the set of non-negative entries in matrix $\mathbf{O}$.

The introduction of the overlap term in the optimization function inherently leads to better node coverage of the graph. Intuitively, a new structure with high edge overlap to an existing structure is commensurately penalized under this scheme for repeating some of the existing structure's edges. This results in choosing structures which explain different regions of the graph without excessively repeating edges, biasing the solution away from redundancy.

Our proposed edge-overlap aware encoding can effectively handle a model family $\mathcal{M}$ that consists of subsets of node- and edge-overlapping structural patterns, and can choose a model M that describes the input graph well, and also minimizes redundant modeling of nodes and edges.

5.4 ConDeNSe: Our Proposed Algorithm

Based on the model from Section 5.3, we propose ConDeNSe, an ensemble, edge-overlap-aware algorithm that summarizes a graph with a compact supergraph consisting of a diverse set of structural patterns (e.g., *fc*, *hs*). ConDeNSe consists of four modules, which we give in Algorithm 5.1 and describe in detail next.

5.4.1 Module A: Unified Pattern Discovery Module

As we mentioned, earlier, in our formulation, we consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω (e.g., a summary with 10 full cliques, 3 bipartite cores, 5 stars and 9 hyperbolic structures). Towards this goal, the first step is to discover subgraphs in the input graph. These can then be used to build its summary. To find the 'perfect' graph summary, we would need to generate all possible (2^n) patterns for a given graph G, and then, from all possible (2^{2^n}) combinations of these patterns pick the set that minimizes Equation (5.1). This is intractable even for small graphs. For example, for $n = 100$ nodes, there are more than $2^{\text{nonillion}}$ (1 nonillion $= 10^{30}$) possible summaries. We reduce the search space by considering patterns that are found via graph clustering methods, and are likely to fit well the structural patterns in Ω.

Algorithm 5.1: CONDENSE

```
1: Input: graph $G$, parameters of clustering methods in Module A
2: // Module A: Pattern discovery: Discovery of a diverse set of patterns $P$.
3: $P$ = SLASHBURN $(G, h_{slash}) \cup$ LOUVAIN $(G, \tau) \cup$ SPECTRAL $(G, k) \cup$ METIS $(G, k)$
4:       $\cup$ HYCOM $(G) \cup$ BIGCLAM $(G) \cup$ KCBC $(G)$          {discussion of parameters in Sec. 5.5}
5: //Module B: MDL-based structural pattern identification as full cliques, bipartite cores, stars,
6: // chains and hyperbolic structures.
7: for $g \in P$
8:     for $\omega \in \Omega$
9:         // e.g., hub identification in star structure $\omega$='st' (Section 5.4.2)
10:        $r(g, \omega)$ = 'best' representation of $g$ as structure type $\omega$
11:    // $s$: type of structure for pattern $g$ using its best representation $r(g, \omega)$
12:    $s = \arg\min_{\omega \in \Omega} L_{r(g,\omega)}(g, \omega) = \arg\min_{\omega \in \Omega}\{L(\omega) + L(E_\omega^+) + L(E_\omega^-)\}$    {using Eq. (5.3)-(5.7)}
13: //Module C: Overlap-aware summary assembly by employing STEP or its faster variants
    (Section 5.4.3).
14: $M = \arg\min L(G, M) = \arg\min\{L(M) + L(\mathbf{E}) + L(\mathbf{O})\}$        {Eq. (5.1),(5.2),(5.8)-(5.10)}
15: // Module D: Approximate supergraph $G_S(\mathcal{V}_S, \mathcal{E}_S)$ creation conditional on the discovered
    patterns
16: // (supernodes linked via weighted superedges).
17: $\mathcal{V}_S = \{s \in M\}$                              {supernodes = structures in $M$}
18: $\mathcal{E}_S = \{(s_i, s_j, w_{ij}) \mid w_{ij} = |\{u, v\}|$, node $u \in s_i$, node $v \in s_j, i \neq j\}$     {superedges}
19: return approximate supergraph $G_S(\mathcal{V}_S, \mathcal{E}_S)$ (summary $M$)
```

The literature is rich in graph clustering methods [BGLL08, KK99, YL13, KF11a]. However, each approach is biased towards specific types of structures, which are most often cliques and bipartite cores. Choosing a decomposition method to generate patterns for the summary depends on the domain, the expected patterns (e.g., mainly clique- or star-like structures), and runtime constraints. To mitigate the biases introduced to the summary by individual clustering methods, and consider a diverse set of candidate patterns, we propose a unified approach that leverages seven existing clustering methods: SLASHBURN, LOUVAIN, SPECTRAL, METIS, HYCOM, BIGCLAM, and KCBC (which we described in Sec. 5.2). In Table 5.1, we present the qualitative advantages, disadvantages, and biases of the methods. Specifically, SLASHBURN tends to provide excellent graph coverage and biased summaries in which stars dominate. Conversely, most other approaches produce primarily full cliques and stars, and some bipartite cores. HYCOM finds mainly hyperbolic communities with skewed degree distributions.

Our proposed unified approach (Algorithm 5.1, lines 2-4) is expected to lead to summaries with a better balanced set of structures (i.e., a good mix of exact and approximate cliques, bipartite cores, stars, chains and hyperbolic structures), and lower encoding cost than any standalone graph clustering method. At the same time, it is expected to take longer to generate all the patterns (although the clustering methods can trivially run in parallel), and the search space for the summary becomes larger—equal to the union of all the subgraphs that the clustering methods generate.

In the experimental evaluation, we use CONDENSE to empirically compare the impact of these methods on the summary quality and evaluate their summarization power.

5.4.2 Module B: Structural Pattern Identification Module

This module (Algorithm 5.1, lines 5-12) identifies and assigns an identifier structural pattern in Ω to all the subgraphs found in module A. In other words, this module seeks to characterize each cluster with its best-suited pattern in $\Omega = \{fc, st, bc, ch, hs\}$. Let g be the induced graph of a pattern generated in Step 1, and ω be a pattern in Ω. Following the reasoning in Section 5.3, we use MDL as a selection criterion. To model g with ω, we first model g with its best representation as structure type ω (explained in detail next), $r(g,\omega)$, and define its encoding cost as $L_{r(g,\omega}(g,\omega) = L(\omega) + L(g|\omega) = L(\omega) + L(E^+_\omega) + L(E^-_\omega)$, where E^+_ω and E^-_ω encode the erroneously modeled and unmodeled edges of g. The pattern type in ω that leads to the smallest MDL cost is used as the identifier of the corresponding subgraph g (lines 11-12 in Alg. 5.1).

Finding the best representation $r(g,\omega)$. Per pattern type ω, each pattern g can be represented by a family of structures—e.g., we can represent g with as many bipartite cores as can be induced on all possible permutations of g's nodes into two sets L (left nodeset) and R (right nodeset) . The only exception is the full clique (*fc*) pattern, which has a unique (unordered) set of nodes. To make the problem tractable, we use the graph-theoretical properties of the pattern types in Ω in order to choose the representation of g which minimizes the incorrectly modeled edges.

Specifically, we represent g as a star by identifying its highest-degree node as the hub and all other nodes as spokes. Representing g as a bipartite core reduces to finding the maximum bipartite pattern, which is NP-hard. To scale-up the computation, we approximate it with semi-supervised classification with two classes L and R, and the prior information that the highest-degree node belongs to L and its neighbors to R. For the classification, we use Fast Belief Propagation [KKK+11] with heterophily between neighbors. Similarly, representing g as a chain reduces to finding its longest path, which is also NP-hard. By starting from a random node, we perform Breadth First Search two times, and end on nodes v_1 and v_2, respectively. Then, we consider the path v_1 to v_2 (based on BFS), and perform local search to further expand it. For the hyperbolic structures, we used power-law fitting (`http://tuvalu.santafe.edu/~aaronc/powerlaws/` by Clauset et al.). Lines 7-10 in Algorithm 5.1 succinctly describe the search of the best representation r for every subgraph g and pattern type ω.

5.4.3 Module C: Structural Pattern Selection Module

This module is key for creating *compact* summaries and is described in lines 13-14 of Alg. 5.1. Ideally, we would consider all possible combinations of the previously identified structures and pick the subset that minimizes the encoding cost in Equation (5.1). If $|\mathcal{S}|$ structures have been found and identified in the previous steps, finding the optimal summary from $2^{|\mathcal{S}|}$ possibilities is not tractable. For reference, we have seen empirically that graphs with about 100,000 nodes, have over $50K$ structures. The optimization function is neither monotonic nor submodular, in which case a greedy hill climbing approach would give a $(1-\frac{1}{\epsilon})$-approximation of the optimal.

Instead of considering all possible combinations of structures for the summary, prior work has proposed GNF, a heuristic that considers the structures in decreasing order of "local" encoding benefit and includes in the model the ones that help further decrease the graph's encoding cost $L(G, M)$. The local encoding benefit [KKVF14a] is defined as $L(g, \emptyset) - L(g, \omega)$, where $L(g, \emptyset)$ represents the encoding of g as noise (i.e., empty model). Although it is efficient, its output summary and performance heavily depend on the structure order. To overcome these shortcomings and obtain more compact summaries, we propose a new structural pattern selection method, Step, as well as a faster serial version and three parallel variants: Step-P, Step-PA, and k-Step.

• **Step.** This method iteratively sifts through all the structures in $\mathcal{S}$ and includes in the summary the structure that decreases the cost in Equation (5.1) the most, until no structure further decreases the cost. Formally, if $\mathcal{S}_i$ is the set of structures that have not been included in the summary at iteration i, Step chooses structure s_i^* s.t.

$$s_i^* = arg \min_{s \in \mathcal{S}_i} L(G, M_{i-1} \cup \{s\})$$

where M_{i-1} is the model at iteration $i-1$, and $M_0 = \emptyset$ is the empty model. CONDENSE with Step finds up to 30% more compact summaries than baseline methods, but its quadratic runtime $O(|\mathcal{S}|^2)$ makes it less ideal for large datasets with many structures $\mathcal{S}$ produced by module A. Therefore, we propose four methods that significantly reduce Step's runtime while maintaining its summary quality.

• **Step-P.** The goal of Step-P is to speed up the computation of Step by iteratively solving smaller, "local" versions of Step in parallel. Step-P begins by dividing the nodes of the graph into p partitions using METIS. Next, each candidate structural pattern is assigned to the partition with the maximal node overlap. Step-P then iterates until convergence, with each iteration consisting of two phases:

1. **Parallelize.** In parallel, a process is spawned for each partition and is tasked with finding the structure that would lower the encoding cost in Eq. (5.1) the most out of all the structures in its partition. For any given partition, there may be no structure that lowers the global encoding cost.
2. **Sync.** From all structures returned in phase 1, the one that minimizes Equation (5.1) the most is added to the summary. If no structure reduces the encoding cost, the algorithm has converged. If not, phase 1 is repeated.

• **Step-PA.** In addition to parallelizing Step, we introduce the idea of "inactive" partitions, which is an optimization designed to reduce the number of processes that are spawned by Step-P. Step-PA differs from Step-P by designating every partition of the graph as active, then if a partition fails x times to find a structure that lowers the cost in Equation (5.1), that partition is declared inactive and is not visited in future iterations. Thus, the partitions with structures not likely to decrease the overall encoding cost of the model get x chances (e.g. 3) before being eventually ruled out, effectively reducing the number of processes spawned for each iteration of Step-PA after the first x iterations.

• **k-Step.** The pseudocode of this variant is given in Algorithm 5.2. k-Step further speeds up Step while maintaining high-quality summaries. This algorithm has two phases: the first applies Step-P k times (lines 3-5) to guarantee that the initial structural patterns included in the summary are of good quality. The second expands the summary by building local solutions of STEP-P per active partition (lines 8-9). If a partition does not return any solution, it is flagged as inactive (lines 10-11). For the partitions that returned non-empty solutions, the best structure per partition is added into a temporary list (line 13), and a parallel "glocal" step applies STEP-P over that list and populates the summary (lines 14-16). We refer to this step as "glocal" because it is a global step within the local stage. The local stage is repeated until no active partitions are left.

Algorithm 5.2: κ-STEP

```
1:  Input: graph $G(\mathcal{V}, \mathcal{E})$; list of structures $\mathcal{S}$; $P$ partitions; κ iterations
2:  ActivePartitions = $\{1, \ldots, P\}$                          {all partitions are active}
3:  // Stage 1: Global
4:  for $i = 1 : \kappa$
5:      run STEP-P ()                                              {summary of κ structures}
6:  // Stage 2: Local Stage
7:  repeat:
8:      for $p \in$ ActivePartitions:                              {2.1: Local sub-stage }
9:          $s$ = run STEP-P-Parallelize()                         {s = best structure in $p$}
10:         if $s = \emptyset$                                     {no structure returned}
11:             ActivePartitions.remove(p)                         {partition $p$ is inactive}
12:         else
13:             bestStructs.add(s)                                 {$s$ is candidate for $M$}
                                                                   {$p$ remains active}
14:     repeat:                                                    {in parallel, add structures to $M$}
15:         run STEP-P-Sync(bestStructs)                           {2.2: Glocal sub-stage }
16:     until bestStructs = $\emptyset$ or Eq. (5.1) is minimized
17: until ActivePartitions = $\emptyset$
18: return $M$
```

5.4.4 Module D: Approximate Supergraph Creation Module

In the empirical analysis (Section 3.6), we show that STEP results in graph summaries with up to 80-90% fewer structures than the baselines, and thus can be leveraged for tractable graph visualization. The last and fourth module of CONDENSE (Algorithm 5.1, lines 15-18), instead of merely outputting a list of structures, creates an "approximate" supergraph which gives a high-level but informative view of large graphs. An *exact* supergraph, $G_S(\mathcal{V}_S, \mathcal{E}_S)$, of a graph $G(\mathcal{V}, \mathcal{E})$ consists of a set of supernodes $\mathcal{V}_S = P(\mathcal{V})$ which is a power set (i.e., family of sets) over $\mathcal{V}$ and a set of superedges $\mathcal{E}_S$. The superweight is often defined as the sum of edge weights between the supernodes' constituent nodes.

Unlike most prior work, CONDENSE creates "approximate," yet powerful supergraphs: (i) the supernodes do not necessarily correspond to a set of nodes with the same connectivity, but to *rich* structural patterns (including hyperbolic structures and chains); (ii) the supernodes may have *node overlap*, which helps to pinpoint bridge nodes (i.e., nodes that span multiple communities);

(iii) the supernodes may show deviations from the perfect corresponding structural patterns (i.e., they correspond to near-structures).

Definition 5.1: CONDENSE Approximate Supergraph

A **CONDENSE approximate supergraph** of G is a supergraph with supernodes that correspond to *possibly-overlapping structural patterns* in Ω. These patterns are approximations of clusters in G.

In other words, the CONDENSE supergraphs consist of supernodes that are *fc*, *st*, *ch*, *bc*, and *hs*. To obtain an approximate supergraph, we map the structural patterns returned in module C to approximate supernodes. Then, for every pair of supernodes, we add a superedge if there were edges between their constituent nodes in $\mathcal{V}$ and set its superweight equal to the number of such (unweighted) edges, as shown in line 18 of Algorithm 5.1.

To evaluate the edge overlap in the summaries, and hence the effectiveness of our overlap-aware encoding, we use the normalized overlap metric. The normalized overlap between two supernodes is their Jaccard similarity. It is 0 if the supernodes do not share any nodes, and close to 1 if they share many nodes compared to their sizes. Although it is not the focus of the current paper, the CONDENSE supergraphs can be used for visualization and potentially for approximation of algorithms on large networks (without specific theoretical guarantees, at least in the general form).

5.4.5 CONDENSE: Complexity Analysis

We discuss the complexity of CONDENSE by considering each module separately:

The first module has complexity $O(m + nk)$, which corresponds to SPECTRAL. However, in practice, HYCOM is often slower than SPECTRAL, likely due to implementation differences (JAVA vs. MATLAB). The complexity of this module can be lowered by selecting the fastest methods. Module B is linear on the number of edges of the discovered patterns. Given that they are overlapping, the computation of $L(G, M)$ is done in $T = O(|M|^2 + m)$, which is $O(m)$ for real graphs with $|M|^2 << m$. In module C, STEP has complexity $O(|S|^2 \times T)$, where S is the set of labeled structures. STEP-P and STEP-PA are $O(t \times \frac{|S|^2}{p} \times T)$, where p is the number of METIS partitions ('active' partitions for STEP-PA) and t is the number of iterations. K-STEP is a combination of STEP-P and a local stage, so it runs in $O(K \times \frac{|S|^2}{p} \times T + t_{lcl} \times (\frac{|S|^2}{p_{active}} + p_{active}^2) \times T)$, where t_{lcl} is the iterations of its local stage. Finally, the supergraph (module D) can be generated in $O(m)$.

5.5 Empirical Analysis

We conduct thorough experimental analysis to answer three main questions:

- How effective is CONDENSE?
- Does it scale with the size of the input graph?

Name	Nodes	Edges	Description
`EUmail` [LK14]	265,214	420,045	EU uni. email comm.
`Enron` [LK14]	80,163	288,364	Enron email comm.
`AS-Caida` [LK14]	26,475	106,762	BGP routing table
`AS-Oregon` [LK14]	13,579	37,448	Router connections
`Choc`	2,899	5,467	Co-editor wiki graph

Table 5.3: Summary of graphs used in our experiments.

- How do the clustering methods compare in terms of summarization power?

Setup. We ran experiments on the real graphs given in Table 5.3. As far as the parameter setting for the clustering methods is concerned, for SlashBurn, we choose the number of hub nodes to slash per iteration $h_{slash} = 2$ in order to achieve better granularity of clusters. For Louvain, we choose resolution $\tau = 0.0001$ as it generates comparable number of clusters with other clustering methods for all our datasets. For Spectral and METIS, the number of clusters k are set to $\sqrt{n/2}$ according to a rule of thumb [SMO+03], where n is the number of nodes in the graph. As for other clustering methods, they are parameter-free hence no need to set up parameters. Unless otherwise specified, we followed the same rule of thumb for setting the number of input METIS partitions p for all the Step variants. In subsections 5.5.1 and 5.5.2, we set the number of chances $x = 3$ for Step-PA.

5.5.1 Effectiveness of ConDeNSe

Ideally, we want a summary to be: (i) concise, with a small number of structures/supernodes; (ii) minimally redundant, i.e., capturing dependencies such as *overlapping* supernodes, but without overly encoding overlaps; and (iii) covering in terms of nodes and edges. Our proposed method, ConDeNSe, constitutes an (almost) unbiased way of analyzing the structure of a given graph. How does it fare in terms of these properties? To answer the question, we perform experiments on the real data in Table 5.3.

Baselines. The first baseline is VoG [KKVF14a], which we describe in Section 5.2. For our experiments, we used the code that is online at `https://github.com/GemsLab/VoG_Graph_Summarization`. The second baseline is our proposed method, ConDeNSe, combined with the GnF heuristic (described in Section 5.4.3) from prior literature.

A1. Conciseness. In Table 5.4, we compare our proposed method (for different selection methods) and the baselines with respect to their compression rates, i.e., the percentage of bits needed to encode a graph with the composed summary over the number of bits needed to encode the corresponding graph with an empty model/summary (that is, all the edges are in the error matrix). In parentheses, we also give the total number of structures in the summaries. We see that compared to the baselines, ConDeNSe with the Step variants gives significantly more compact summaries, with 30%-50% lower compression rate and about 80-90% fewer structures. The Step variants give comparable results in summarization power.

Dataset	VoG [KKVF14a]	CONDENSE GNF	CONDENSE with STEP Variants			
			STEP	STEP-P	STEP-PA	K-STEP
Choc	88%(101)	88%(101)	56%(24)	56%(24)	56%(21)	56%(22)
AS-Oregon	71%(400)	69%(379)	35%(41)	35%(41)	35%(35)	35%(36)
AS-Caida	-	71%(572)	42%(51)	42%(51)	42%(46)	44%(60)
Enron	75%(2330)	74%(2044)	-	26%(50)	25%(201)	25%(218)
EUmail	-	65%(1440)	-	-	-	59%(15[a])

Table 5.4: CONDENSE: Compression rate with respect to the empty model. In parentheses, number of structures in the corresponding summary. A "-" means that the corresponding method was terminated after 4 days. Notice that CONDENSE with STEP variants achieves better graph compression with even fewer structures than alternatives.

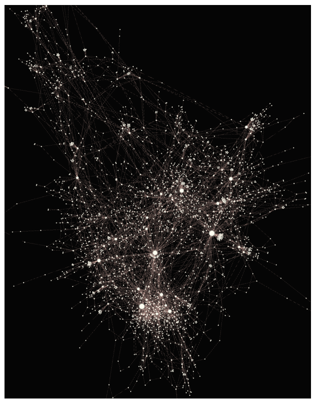
[a]In the interest of time, the summary size was limited to 15.

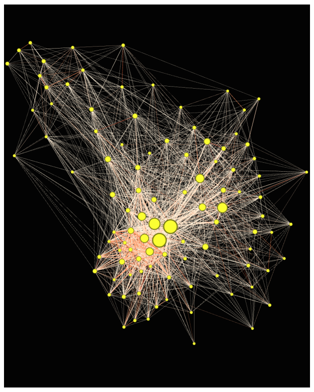
Dataset	VoG [KKVF14a]	CONDENSE
Choc	900 (0.04)	74 (0.029)
AS-Oregon	15875 (0.047)	126 (0.026)
AS-Caida	-	382 (0.018)
Enron	447052 (0.02)	509 (0.015)
EUmail	-	0

Table 5.5: Overlapping supernode pairs and average similarity in parentheses. A "-" means that the corresponding method was terminated after 4 days. Notice that CONDENSE produces structures with less overlap, leading to better visualization and interpretability.

A2. Minimal Redundancy. In Figures 5.1 and 5.3, we visualize the supergraphs for AS-Oregon and Choc, which are generated from the selected structures of VoG and CONDENSE-STEP. It is clear that the CONDENSE supergraphs are significantly more compact. In Table 5.5, we also provide information about the number of overlapping supernode pairs and their average Jaccard similarity, as an overlap quantifier (in parentheses). For brevity, we only give results for K-STEP, since the results of the rest STEP-series are similar. We observe that CONDENSE has significantly fewer supernode overlaps, and the overlaps are smaller in magnitude. We also note that the overlap encoding module achieves 10-20% reduction in overlapping edges, showing its effectiveness for minimizing redundancy.

A3. Coverage. We give the summary node/edge coverage (as a ratio of the original) for different assembly methods in Figure 5.4. We observe that the baselines have better edge coverage than the STEP variants, which is expected as they include significantly more structures in their summaries. However, in most cases, K-STEP and STEP-PA achieve better node coverage than the baselines. Taking into account the desired property for summary conciseness, CONDENSE with STEP variants has better performance, balancing coverage and summary size well.


(a) Original graph


(b) VoG [KKVF14a]

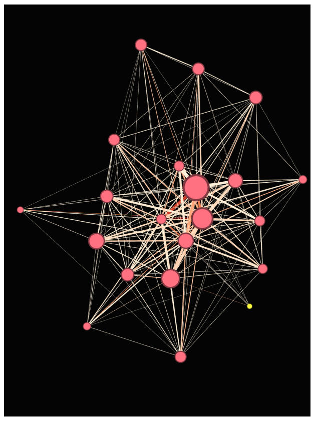

(c) CONDENSE-STEP

Figure 5.3: **CONDENSE-STEP generates more compact supergraphs.** b-c: The full supergraphs of `Choc` by VoG-GNF, and CONDENSE-STEP, respectively. Yellow for stars, red for cliques, green for bipartite cores. The edge weights correspond to the number of inter-supernode edges.

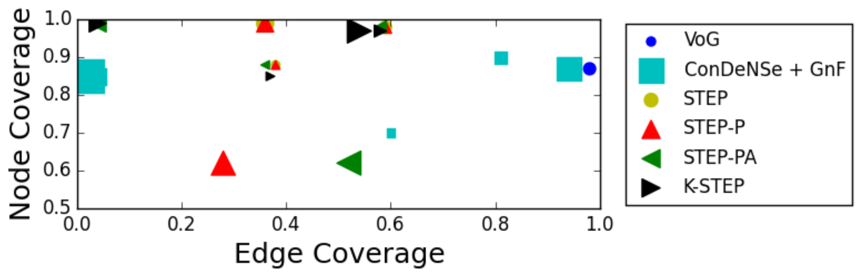


Figure 5.4: **STEP variants have better node coverage over alternatives, and handle the summary coverage-conciseness trade-off well.** Marker size corresponds to the graph size.

What other properties do the various summaries have? What are the main structures found in different types of networks (e.g., email vs. routing networks)? In Table 5.6, we show the number of in-summary structures per type. We note that no chains and hyperbolic structures were included in the summaries of the networks that we show here (although some were found by the pattern discovery module, and there are synthetic examples in which they are included in the final summaries). This is possibly because stars are extreme cases of hyperbolic structures, and the encoding of (approximate) hyperbolic structures is of the same order, yet often more expensive than the encoding of stars with errors. As for chains, they are not 'typical' clusters found by popular clustering methods, but rather by-products of the decomposition methods that we consider. Moreover, given that the chain encoding considers the sequence of node IDs, and errors in the real data increase the encoding cost, very often encoding them in the error matrix yields better compression. One observation is that STEP gives less biased summaries than the baselines. For email networks, we see that stars are dominant (e.g., users emailing multiple employees that do not contact each other, administrators sending e-mails to large mailing lists, etc.), with several of cliques and bipartite cores too. For routing networks (`AS-Caida` and

Dataset	VoG [KKVF14a]	ConDeNSe-GnF	ConDeNSe with Step Variants			
			Step	Step-P	Step-PA	k-Step
Choc	[0,101,0]	[1,100,0]	[21,3,0]	[21,3,0]	[20,1,0]	[21,1,0]
AS-Oregon	[1,399,0,]	[19,355,5]	[27,13,1]	[27,13,1]	[26,9,0]	[26,10,0]
AS-Caida	-	[2,557,13]	[38,7,6]	[38,7,6]	[37,5,4]	[43,12,5]
Enron	[2,2323,5]	[160,1676,208]	-	[45,2,3]	[60,108,33]	[61,124,33]
EUmail	-	[0,1261,179]	-	-	-	[15,0,0]

Table 5.6: ConDeNSe: Number of structures per type in the summaries in the format $[fc, st, bc]$, for VoG we have $[fc + nc, st, bc + nb]$, where nc is near-clique and nb is near-bipartite core. The ConDeNSe summaries are more balanced, *without* a specific pattern type dominating in all the graphs. In the interest of time, we find the top-50 and top-15 structures for Enron and EUmail, respectively. A "-" means that the corresponding method was terminated after 4 days.

AS-Oregon), we mostly see cliques between communities of autonomous systems, and a few stars and bipartite cores. In collaboration networks, cliques are the most common structures. VoG and ConDeNSe-GnF are biased towards stars, which exceed the other structures by an order of magnitude. Overall, ConDeNSe fares well with respect to the desired properties for graph summaries.

5.5.2 Runtime Analysis of ConDeNSe

We give the runtime of pattern discovery and the Step methods in Figure 5.5. "Discovery" represents the maximum time of the clustering methods, and "Disc.-Fast" corresponds to the slowest among the fastest methods (KCBC, Louvain, METIS, BigClam). We ran the experiment on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz, and 256GB memory.

We see that the fast unified discovery is up to $80\times$ faster than the original one. As expected, Step is the slowest method. The parallel variants Step-P, Step-PA, and k-Step are more scalable, with k-Step being the most efficient. Taking into account the similarity of the heuristics in both conciseness and coverage, Figure 5.5 further suggests that k-Step is the best-performing heuristic given that it exhibits the shortest runtime.

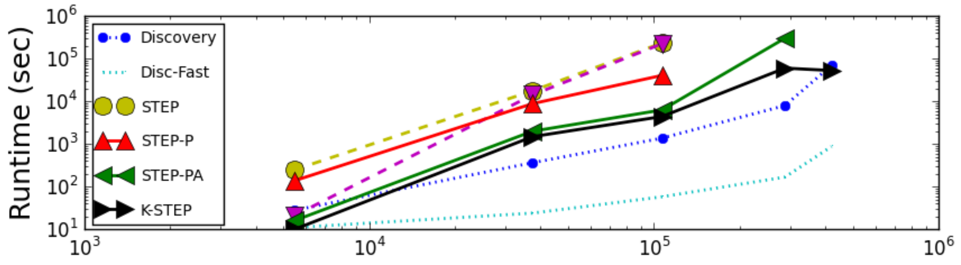


Figure 5.5: **Runtime vs. # of edges: k-Step is more efficient than the other methods, and scales to larger graphs.**

Dataset	STEP-P	STEP-PA	K-STEP
Choc	1	0.9886	0.9667
AS-Oregon	1	0.9704	0.9285
AS-Caida	1	0.9865	0.8238
Enron	1	0.5012*	0.446[a]

Table 5.7: Agreement of STEP and its variants. They approximate STEP quite well.

[a]Agreement based on the top-50 structures for efficiency reasons.

5.5.3 Sensitivity Analysis of CONDENSE: Agreement between STEP and STEP-variants

Our analysis so far has shown that K-STEP leads to the best combination of high compression and low runtime compared to the other methods. But how well does it approximate STEP in terms of the generated summary? To answer this question, we evaluate the "agreement" between the generated summaries, which in this section we view as ordered lists of structures based on the iteration they were included in the final summary (which defines the rank of each structure). Since popular rank correlation measures, such as Spearman's ρ, Kendall's τ, only work on permuted lists or lists of the same length, while the generated summaries can have different constituent structures and lengths, we propose AG as a measure of agreement. This measure effectively handles summaries of different lengths, and penalizes with different, adaptive weights 'rank' disagreement between structures included in both summaries, and disagreement for missing structures from one summary. Let M_1 and M_2 be the two summaries, and $rank(s, M_i)$ be the ranking of structure s in summary M_i (i.e., the order in which it was included in the summary while minimizing Eq. (5.1)). We define the agreement of the two summaries as:

$$AG(M_1, M_2) = 1 - 1/Z[\alpha D + (1 - \alpha/2)D_1 + (1 - \alpha/2)D_2]$$

where $D = \sum_{s \in M_1 \cap M_2} |rank(s, M_1) - rank(s, M_2)|$ is the rank disagreement for structures that are in both summaries, $D_1 = \sum_{s \in M_1 \cap M_2'} |(|M_2| + 1) - rank(s, M_1)|$ is the disagreement for structures in M_1 but not in M_2, D_2 is defined analogously to capture structures in M_2 but not in M_1. Finally, Z is a normalization factor that guarantees that AG is in $[0, 1]$: $Z = (1 - \frac{\alpha}{2}) \sum_{s \in M_1} |(|M_2| + 1) - rank(s, M_1)| + \sum_{s \in M_2} |(|M_1| + 1) - rank(s, M_2)|$. $AG = 1$ means identical summaries, while 0 completely different summaries. In order to penalize more the structures that appear in one summary but not in the other, we set $\alpha = 0.3$ (the results are consistent for other values of α). In Table 5.7, we give the agreement between STEP and its faster variants. As a side note, the agreement with VoG is almost 0 in all the cases. As expected, STEP-P produces the same summaries as STEP, while STEP-PA and K-STEP preserve the agreement well.

5.5.4 Sensitivity to the number of partitions

All parallel variants of Step take p METIS partitions as input. To analyze the effects of varying p on runtime and agreement, we ran k-Step and increased p from 12 to 96 in increments of 12.

We only give the results on `AS-Oregon`, since other datasets lead to similar results. We observe that while agreement is robust, runtime decreases as p increases and especially so with the smaller values of p. This observation is consistent with our motivation for parallelizing Step: by decreasing the number of structures in any given partition, the "local" subproblems of Step become smaller and thus less time-consuming. Figure 5.6a shows the effect of the number of partitions on runtime and agreement, both averaged over three trials.

5.5.5 Sensitivity of Step-PA

We also experimented with varying the number of "chances" allowed for partitions in the Step-PA variant. Step-PA speeds up Step-P by forcing partitions to drop out after not returning structures for a certain number of attempts (x). However, while giving partitions fewer chances can speed up the algorithm, smaller values of x can compromise compression and agreement.

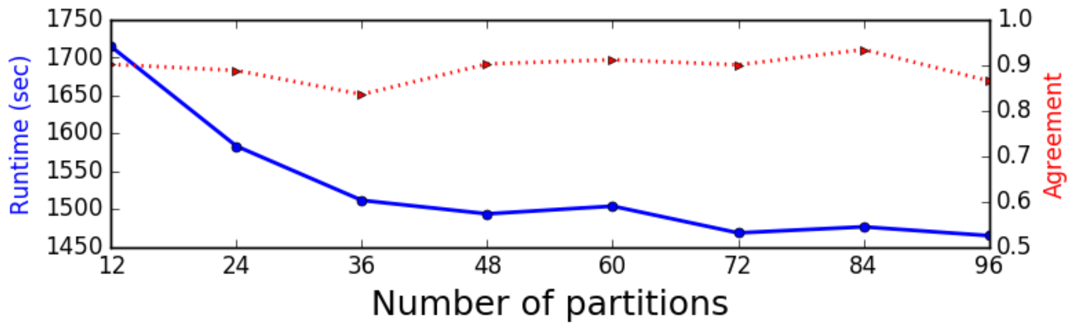
In Figure 5.6b, we give the agreement and runtime of Step-PA on `Choc` and `AS-Oregon` setting $x = \{1, 2, 3, 4, 5\}$. We found that both runtime and agreement increased with x, and plateaued after $x = 3$. This suggests that forcing partitions to drop out early, while better for runtime, can lead to the loss of candidate structures that may be useful for compression later.

5.5.6 ConDeNSe as a Clustering Evaluation Metric

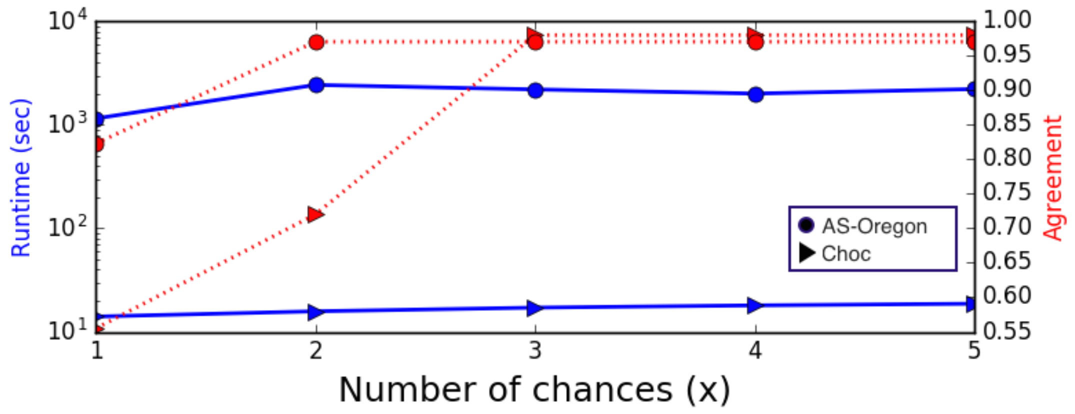
Given the independence of Step from the structure ordering, we use ConDeNSe to evaluate the different clustering methods and give their individual compression rates in Table 5.8. For number and type of structures we give our observations based on `AS-Oregon` (Figure 5.7), which is consistent with other datasets. As we see in the case of `AS-Oregon`, SlashBurn mainly finds stars (136 out of 138 structures); Louvain, Spectral, KCBC, and BigClam reveal mostly cliques (9/9, 15/17, 9/9, and 28/29, respectively); METIS has a less biased distribution (18 cliques, 12 stars), and HyCoM, though looks for hyperbolic structures, tends to find structures more concisely described as cliques in our experiments (45 out of 52 structures). Also, SlashBurn

Dataset	Clustering Methods						
	SlashBurn	**Louvain**	**Spectral**	**METIS**	**HyCoM**	**BigClam**	**KCBC**
`Choc`	88%	99%	99%	100%	100%	87%	**78%**
`AS-Oregon`	76%	94%	82%	85%	98%	83%	**65%**
`AS-Caida`	**70%**	100%	100%	98%	98%	91%	74%

Table 5.8: ConDeNSe as an evaluation metric: Compression rate of clustering methods with respect to the empty model (i.e., percentage of bits for encoding the graph given the chosen model vs. the empty model).



(a) AS-Oregon: Runtime and agreement vs. number of partitions.



(b) Choc + AS-Oregon: Runtime and agreement vs. number of chances (x).

Figure 5.6: **The agreement is robust to the number of partitions and chances, while runtime decreases with more partitions and is unaffected by number of chances.**

and BigClam discover more structural patterns than other methods, which partially explains their good compression rate in Table 5.8. One notable takeaway from these results is that decomposition methods traditionally do not find chains and hyperbolic structures, which is likely both a function of traditional notions of "community" behavior as well as a reflection on the difficulty of identifying such structures automatically.

We perform an ablation study to evaluate the graph clustering methods in the context of summarization. Specifically, we create a leave-one-out unified model for each clustering method and evaluate the contribution of each clustering method to the final summary. The results are shown in Table 5.9. We see that Louvain appears to be the most important method: when included, it contributes the most; and when dropped, the compression rate reduces (worse). When KCBC is dropped, SlashBurn gets to the top, but Louvain also has considerable contribution. In the missing-Louvain case, the contribution gets redistributed among other clustering methods to make up for it, this effect differs by dataset, e.g., METIS gets boosted for AS-Oregon, while it is Spectral for Choc.

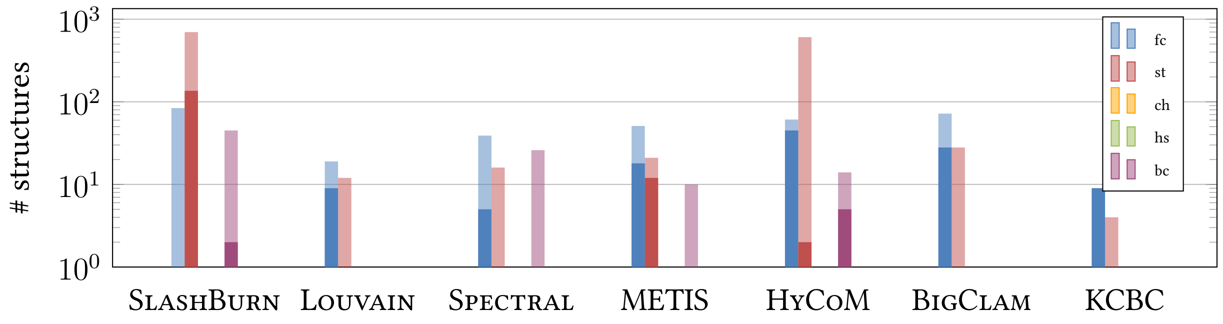


Figure 5.7: **Number of structures found by various clustering methods for `AS-Oregon`.** Transparent/solid rectangles for before/after the structure selection step. Notation: fc: full clique, st: star, ch: chain, bc: bipartite core, hs: hyperbolic structure.

Clustering Method	Compression Rate	Contribution per Method						
		SlashBurn	Louvain	Spectral	METIS	HyCoM	BigClam	KCBC
SlashBurn	22%	-	**63%**	10%	7%	7%	0	13%
Louvain	**30%**	30%	-	16%	**45%**	0	3%	7%
Spectral	22%	32%	**51%**	-	3%	0	0	14%
METIS	22%	34%	**46%**	5%	-	2%	0	12%
HyCoM	22%	35%	**48%**	3%	3%	-	0	13%
BigClam	22%	34%	**46%**	2%	2%	2%	-	12%
KCBC	25%	**50%**	35%	6%	2%	2%	6%	-

Table 5.9: Ablation study for `AS-Oregon`. Louvain and SlashBurn contribute most to the ConDeNSe summaries.

In terms of runtime, for modules A and B (pattern discovery and identification), Spectral and HyCoM take the longest time, while KCBC, Louvain, METIS, and BigClam are the fastest ones, with SlashBurn falling in the middle. For Module C (summary assembly), the trade-off between runtime and candidate structures is given in the complexity analysis (Appendix 5.4.5). In practice, HyCoM usually takes the longest time, followed by Spectral and SlashBurn.

5.6 Conclusion

In this work we proposed ConDeNSe, a method that summarizes large graphs as small, approximate and high-quality supergraphs conditioned on diverse pattern types. ConDeNSe features a new selection method, Step, which generates summaries with high compression and node coverage. However, this comes at the cost of increased runtime, which we addressed by introducing faster parallel approximations to Step. We provided a thorough empirical analysis of ConDeNSe, and contributed a novel evaluation of clustering methods in terms of summarization power, complementing the literature that focuses on classic quality measures. We showed that each clustering approach has its strengths and weaknesses and make different contributions to the final summary. Moreover, ConDeNSe leverages their strengths, handles edge-overlapping structures, and shows results superior to baselines, including significant improvement in the bias of summaries with respect to the considered pattern types.

Ideally without the constraint of time, we naturally recommend the application of as many clustering methods in Module A of ConDeNSe. On the other hand, to deal with the additional complexity of having more structures, we recommend choosing faster clustering methods or a mixture of fast and "useful" methods (depending on the application at hand) that contribute good structures, as shown in our analysis.

Part II

Mining Dynamic Graphs

Based on content published in [SKZ+15].

Chapter 6

TimeCrunch: Interpretable Dynamic Graph Summarization

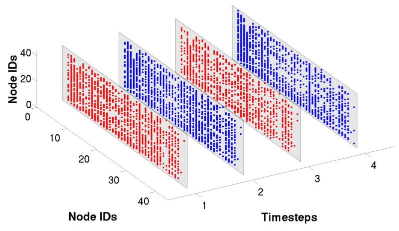
Identifying temporal structures and recurrence patterns in dynamic graphs with a compression paradigm.

Many real-world occurrences can be aptly represented as dynamic graphs which represent interactions over time. While many previous works tackle the problem of extracting structure from static graphs via clustering or partitioning, few such tools exist in the dynamic case. In this chapter, we propose TimeCrunch, an unsupervised method which aims to concisely summarize dynamic graphs using a lexicon of temporal structures which describe recurrence patterns and graph connectivity. TimeCrunch scalably extracts interpretable temporal structures from real-world graphs with millions of nodes and edges and demonstrates that such graphs do in fact exhibit rich temporal structures.

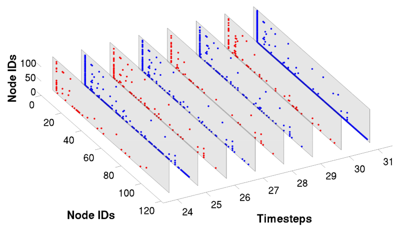
6.1 Introduction

Given a large phonecall network over time, how can we describe it to a practitioner with just a few phrases? Other than the traditional assumptions about real-world graphs involving degree skewness, what can we say about the connectivity? For example, is the dynamic graph characterized by many large cliques which appear at fixed intervals of time, or perhaps by several large stars with dominant hubs that persist throughout? Our work aims to answer these questions, and specifically, we focus on constructing concise summaries of large, real-world dynamic graphs in order to better understand their underlying behavior.

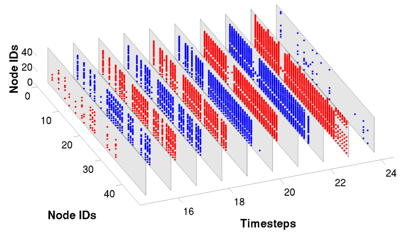
This problem has numerous practical applications. Dynamic graphs are ubiquitously used to model the relationships between various entities over *time*, which is a valuable feature in



(a) 40 users of Yahoo! Messenger forming a *constant near clique* with unusually high 55% density, over 4 weeks in April 2008.



(b) 111 callers in a large phonecall network, forming a *periodic star*, over the last week of December 2007 – note the heavy activity on holidays



(c) 43 collaborating biotechnology authors forming a *ranged near clique* in the DBLP network, jointly publishing through 2005-2012.

Figure 6.1: **TimeCrunch finds coherent, interpretable temporal structures**. We show the reordered subgraph adjacency matrices, over the timesteps of interest, each outlined in gray; edges are plotted in alternating red and blue, for discernibility.

almost all applications in which nodes represent users or people. Examples include online social networks, phone-call networks, collaboration and coauthorship networks and other interaction networks.

Though numerous graph algorithms suitable for static contexts such as modularity-based community detection, spectral clustering, and cut-based partitioning exist, they do not offer direct dynamic counterparts. Furthermore, the traditional goals of clustering and community detection tasks are not quite aligned with the endeavor we propose. These algorithms typically produce groupings of nodes which satisfy or approximate some optimization function. However, they do not offer characterization of the outputs – are the detected groupings stars or chains, or perhaps dense blocks? Furthermore, the lack of explicit ordering in the groupings leaves a practitioner with limited time and no insights on where to begin understanding his data.

In this work, we propose TimeCrunch, an effective approach to concisely summarizing large, dynamic graphs which extend beyond traditional dense and isolated "cavemen" communities. Our method works by leveraging MDL (Minimum Description Length) in order to identify and appropriately describe graphs over time using a lexicon of *temporal phrases* which describe temporal connectivity behavior. Figure 6.1 shows several interesting results found from applying TimeCrunch to real-world dynamic graphs.

- Figure 6.1a shows a *constant near-clique* with 55% density of 40 users in the Yahoo! messaging network over 4 weeks in April 2008. These users are likely bots messaging each other in an effort to appear normal and avoid suspension.
- Figure 6.1b depicts a *periodic star* of 111 callers in the phone-call network of a large, anonymous Asian city during the last week of December 2007. Notice that the star behavior oscillates over time – specifically, odd-numbered timesteps have stronger star structure than the even-numbered ones. Furthermore, the appearance of the star is strongest on Dec. 25th and 31st, corresponding to major holidays.

- Lastly, Fig. 6.1c shows a *ranged near clique* of 43 authors in the DBLP network who jointly published in biotechnology journals such as *Nature* and *Genome Research* from 2005-2012, agreeing with intuition as works in this field typically have many co-authors. The first and last timesteps serve only to demarcate the range of activity.

In this work, we seek to answer the following informally posed problem:

Problem 6.1: (Informal) Dynamic Graph Summarization

Given a dynamic graph, **find** a set of possibly overlapping temporal subgraphs to **concisely describe** the given dynamic graph in a **scalable** fashion.

Our main contributions are as follows:

1. **Problem Formulation:** We show how to define the problem of dynamic graph understanding in in a compression context.
2. **Effective and Scalable Algorithm:** We develop TimeCrunch, a fast algorithm for dynamic graph summarization.
3. **Practical Discoveries:** We evaluate TimeCrunch on multiple real, dynamic graphs and show quantitative and qualitative results.

Reproducibility: Our code for TimeCrunch is open-sourced at `www.cs.cmu.edu/~neilshah/code/timecrunch.tar`.

6.2 Related Work

The related work falls into three main categories: static graph mining, temporal graph mining, and graph compression and summarization. Table 6.1 gives a visual comparison of TimeCrunch with existing methods.

Static Graph Mining. Most works find specific, tightly-knit structures, such as (near-) cliques and bipartite cores: eigendecomposition [SBGF14] (as we saw in Chapter 3, cross-associations [CPMF04], modularity-based optimization methods [NG04, BGLL08]. Dhillon et al. [DMM03] propose information theoretic co-clustering based on mutual information optimization. However, these approaches have limited vocabularies and are unable to find other types of interesting structures such as stars or chains. [KK00, KG10] propose cut-based partitioning, whereas [AKY99] suggests spectral partitioning using multiple eigenvectors – these schemes seek hard clustering of all nodes as opposed to identifying communities, and are not usually parameter-free. Subdue [CH94] and other fast frequent-subgraph mining algorithms [JWP$^+$05] operate on labeled graphs. Our work involves unlabeled graphs and lossless compression.

Temporal Graph Mining. [AY05] aims at change detection in streaming graphs using projected clustering. This approach focuses on anomaly detection rather than finding recurrent temporal patterns. GraphScope [SFPY07] uses graph search for hard-partitioning of temporal

	Temporal	Time-consecutive	Time-agnostic	Dense blocks	Stars	Chains	Interpretable	Scalable	Parameter-free
GraphScope[SFPY07]	✓	✓	✗	✓	✗	✗	✗	✓	✓
Com2[APG+14]	✓	✓	✓	✓	✓	✗	✗	✓	✗
VoG[KKVF14b]	✗	✗	✗	✓	✓	✓	✓	✓	✓
Graph partitioning[KK00, KG10, AKY99]	✗	✗	✗	✓	✗	✗	✗	✓	✗
Community detection[SBGF14, NG04, BGLL08]	✗	✗	✗	✓	✗	✗	✗	?	?
TimeCrunch	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 6.1: Feature-based comparison of TimeCrunch with alternative approaches.

graphs to find dense temporal cliques and bipartite cores. Com2 [APG+14] uses CP/PARAFAC tensor decomposition with MDL for the same. [FFL+08] uses incremental cross-association for change detection in dense blocks over time, whereas [PJZ05] proposes an algorithm for mining cross-graph quasi-cliques (though not in a temporal context). These approaches have limited vocabularies and do not offer temporal interpretability. Dynamic clustering [XKH11] aims to find stable clusters over time by penalizing deviations from incremental static clustering. Our work focuses on interpretable structures, which may not appear at every timestep.

Graph Compression and Summarization. SlashBurn [KF11a], as we previously discussed in Chapter 5, is a recursive node-reordering approach which leverages run-length encoding for graph compression. [TZHH11] uses structural equivalence to collapse nodes/edges to simplify graph representation. These approaches do not compress the graph for pattern discovery, nor do they operate on dynamic graphs. VoG [KKVF14b] uses MDL to label subgraphs in terms of a vocabulary on static graphs, consisting of stars, (near) cliques, (near) bipartite cores and chains. ConDeNSe (introduced in Chapter 5) improves upon VoG by reducing redundancy and improving graph coverage, allowing for multiple graph decomposition algorithms and producing more concise summaries. However, both these approaches only apply to static graphs and do not offer clear extension to dynamic graphs. Our work proposes a suitable lexicon for dynamic graphs, uses MDL to label *temporally coherent* subgraphs and proposes an effective and scalable algorithm for finding them.

6.3 Problem Formulation

In this section, we give the first main contribution of our work: formulation of dynamic graph summarization as a compression problem, using MDL. For clarity, see Table 6.2 for a reference of the recurrent symbols used in this section.

Symbol	Definition
G, $\mathbf{A}$	dynamic graph and adjacency tensor resp.
$\mathcal{V}, n$	node-set, # of nodes of G resp.
$\mathcal{E}, m$	edge-set, # of edges of G resp.
G_x, $\mathbf{A}_x$	x^{th} timestep, adjacency matrix of G resp.
$\mathcal{E}_x, m_x$	edge-set and # of edges of G_x resp.
Δ	set of temporal signatures
Ω	set of static identifiers
Φ	lexicon, set of temporal phrases $\Phi = \Delta \times \Omega$
$\times$	Cartesian set product
M, s	model M, temporal structure $s \in M$ resp.
$\lvert S \rvert$	cardinality of set S
$\lvert s \rvert$	# of nodes in structure s
$u(s)$	timesteps in which structure s appears
$v(s)$	temporal phrase of structure s, $v(s) \in \Phi$
st, ch	star, chain resp.
fc, nc	full, near clique resp.
bc, nb	full, near bipartite core resp.
o, c	oneshot, constant resp.
r, p, f	ranged, periodic, flickering resp.
$\mathbf{M}$	approximation of $\mathbf{A}$ induced by M
$\mathbf{E}$	error matrix $\mathbf{E} = \mathbf{M} \oplus \mathbf{E}$
$\oplus$	exclusive OR
$L(G, M)$	# of bits used to encode M and G given M
$L(M)$	# of bits to encode M

Table 6.2: Frequently used symbols and definitions

The Minimum Description Length (MDL) principle aims to be a practical version of Kolmogorov Complexity [LVV90], often associated with the motto *Induction by Compression*. MDL states that given a model family $\mathcal{M}$, the best model $M \in \mathcal{M}$ for some observed data $\mathcal{D}$ is that which minimizes $L(M) + L(\mathcal{D}|M)$, where $L(M)$ is the length in bits used to describe M and $L(\mathcal{D}|M)$ is the length in bits used to describe $\mathcal{D}$ encoded using M. MDL enforces lossless compression for fairness in the model selection process. Refer to Chapter 2 for more detail on MDL.

We focus on analysis of undirected dynamic graphs using fixed-length, discretized time intervals. However, our notation will reflect the treatment of the problem as one with a series of individual snapshots of graphs, rather than a tensor, for readability purposes. We consider a dynamic graph $G(\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, $m = |\mathcal{E}|$ edges and t timesteps, without self-loops. Here, $G = \cup_x G_x(\mathcal{V}, \mathcal{E}_x)$, where G_x and E_x correspond to the graph and edge-set for the x^{th} timestep. The ideas proposed in this work, however, can easily be generalized to other types of dynamic graphs.

For our summary, we consider the set of temporal phrases $\Phi = \Delta \times \Omega$, where Δ corresponds to the set of temporal signatures, Ω corresponds to the set of static structure identifiers and $\times$ denotes Cartesian set product. Though we can include arbitrary temporal signatures and static structure identifiers into these sets depending on the types of temporal subgraphs we expect to find in a given dynamic graph, we choose 5 temporal signatures which we anticipate to find in real-world dynamic graphs [APG+14] : oneshot (*o*), ranged (*r*), periodic (*p*), flickering (*f*) and constant (*c*), and 6 very common structures found in real-world static graphs [KKR+99, SBGF14] – stars (*st*), *full* and *near* cliques (*fc*, *nc*), *full* and *near* bipartite cores (*bc*, *nb*) and chains (*ch*) . Summarily, we have the signatures $\Delta = \{o, r, p, f, c\}$, static identifiers $\Omega = \{st, fc, nc, bc, nb, ch\}$ and temporal phrases $\Phi = \Delta \times \Omega$. We will further describe these signatures, identifiers and phrases after formalizing our objective.

In order to use MDL for dynamic graph summarization using these temporal phrases, we next define the model family $\mathcal{M}$, the means by which a model $M \in \mathcal{M}$ describes our dynamic graph and how to quantify the cost of encoding in terms of bits.

6.3.1 Using MDL for Dynamic Graph Summarization

We consider models $M \in \mathcal{M}$ to be composed of ordered lists of temporal graph structures with node, but not edge overlaps. Each $s \in M$ describes a certain region of the adjacency tensor $\mathbf{A}$ in terms of the interconnectivity of its nodes. We will use $area(s, M, \mathbf{A})$ to describe the edges $(i, j, x) \in \mathbf{A}$ which s induces, writing only $area(s)$ when context for M and $\mathbf{A}$ is clear.

Our model family $\mathcal{M}$ consists of all possible permutations of subsets of $\mathcal{C}$, where $\mathcal{C} = \cup_v \mathcal{C}_v$ and $\mathcal{C}_v$ denotes the set of all possible temporal structures of phrase $v \in \Phi$ over all possible combinations of timesteps. That is, $\mathcal{M}$ consists of all possible models M, which are ordered lists of temporal phrases $v \in \Phi$ such as flickering stars (*fst*), periodic full cliques (*pfc*), etc. over all possible subsets of $\mathcal{V}$ and $G_1 \cdots G_t$. Through MDL, we seek the model $M \in \mathcal{M}$ which best mediates between the encoding length of the model M and the adjacency tensor $\mathbf{A}$ given M.

Our fundamental approach for transmitting the adjacency tensor $\mathcal{A}$ via the model M is described next. First, we transmit M. Next, given M, we induce the approximation of the adjacency tensor $\mathbf{M}$ as described by each temporal structure $s \in M$ – for each structure s, we induce the edges in $area(s)$ in $\mathbf{M}$ accordingly. Given that $\mathbf{M}$ is a summary approximation to $\mathbf{A}$, $\mathbf{M} \neq \mathbf{A}$ most likely. Since MDL requires lossless encoding, we must also transmit the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, obtained by taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$. Given M and $\mathbf{E}$, a recipient can construct the full adjacency tensor $\mathbf{A}$ in a lossless fashion.

Thus, we formalize the problem we tackle as follows:

Problem 6.2: Minimum Dynamic Graph Description

Given a dynamic graph G with adjacency tensor $\mathbf{A}$ and temporal phrase lexicon Φ, **find** the smallest model M which minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E})$$

where $\mathbf{E}$ is the error matrix computed by $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ and $\mathbf{M}$ is the approximation of $\mathbf{A}$ induced by M.

In the following subsections, we further formalize the task of encoding the model M and the error matrix $\mathbf{E}$.

6.3.2 Encoding the Model

To fully describe a model $M \in \mathcal{M}$, we have the following:

$$\begin{aligned} L(M) \quad = \quad & L_{\mathbb{N}}(|M|+1) + log_2\binom{|M| + |\Phi| - 1}{|\Phi - 1|} \\ & + \sum_{s \in M} (-log_2 P(v(s)|M) + L(c(s)) + L(u(s))) \end{aligned}$$

We begin by transmitting the total number of temporal structures in M using $L_{\mathbb{N}}$, Rissanen's optimal encoding for integers greater than or equal to 1 [Ris78]. Next, we optimally encode the number of temporal structures for each phrase $v \in \Phi$ in M. Then, for each structure s, we encode the type $v(s)$ for each structure $s \in M$ using optimal prefix codes [CT06], the connectivity $c(s)$ and the temporal presence of the s, consisting of the ordered list of timesteps $u(s)$ in which s appears.

In order to have a coherent model encoding scheme, we next define the encoding for each phrase $v \in \Phi$ such that we can compute $L(c(s))$ and $L(u(s))$ for all structures in M. The connectivity $c(s)$ corresponds to the edges in $area(s)$ which are induced by s, whereas the temporal presence $u(s)$ corresponds to the timesteps in which s is present. We consider the connectivity and temporal presence separately, as the encoding for a temporal structure s described by a phrase v is the sum of encoding costs for the connectivity of the corresponding static structure identifier in Ω and its temporal presence as indicated by a temporal signature in Δ.

6.3.2.1 Encoding Connectivity

In this section, we describe how to compute the encoding cost $L(c(s))$ for the connectivity for each type of static structure identifier in our identifier set Ω. The encoding costs are defined similarly to Chapter 5, which focuses on summarizing plain graphs.

Stars: A star is characteristic of a single "hub" node connected to a set of 2 or more "spoke" nodes. We compute $L(st)$ of a star *st* as follows:

$$L(st) \;=\; L_{\mathbb{N}}(|st|-1) + log_2 n + log_2\binom{n-1}{|st|-1}$$

First, we identify the number of spokes of the star. Next, we identify the hub out of n nodes using an index over the combinatorial number system. Lastly, we identify the spokes from the remainder.

Cliques: Cliques are comprised of densely connected sets of nodes. For a full clique *fc*, in which all nodes are directly connected to all other nodes in the clique, we give the cost $L(fc)$ as follows:

$$L(fc) \;=\; L_{\mathbb{N}}(|fc|) + log_2\binom{n}{|fc|}$$

In this case, we encode the number of nodes in the clique followed by their ids. Note that as M is an approximation of G, fc need not *actually* be a full clique in G. If only a few edges of the full clique are not present in G, it may be worthwhile from a compression standpoint to describe it as such. In this case, each falsely represented edge will add to the error cost $\mathbf{E}$. Errors in connectivity encoding will be elaborated on in Sec. 6.3.3.1.

Less dense near-cliques are still interesting from a graph understanding perspective, provided they stand out from the background. For a near clique *nc*, we give $L(nc)$ as follows:

$$\begin{aligned} L(nc) \;&=\; L_{\mathbb{N}}(|nc|) + log_2\binom{n}{|nc|} + log_2(|area(nc)|) \\ &\quad + ||nc||\rho_1 + ||nc||'\rho_0 \end{aligned}$$

Here, we encode the number of nodes and their ids as in the full clique case. However, we additionally encode the edges in the near clique by encoding the number of total edges in $area(nc)$ by optimal prefix codes. We use $||nc||$ and $||nc||'$ to denote the counts for existing and non-existing edges in $area(nc)$. Then, $\rho_1 = -log(||nc||/(||nc|| + ||nc||'))$ and $\rho_0 = -log(||nc||'/(||nc|| + ||nc||'))$ represent the length of the optimal prefix codes for the existing and non-existing edges respectively. Intuitively, the more sparse or dense the near clique is, the cheaper its encoding becomes. As the encoding in this case is exact, we do not add any edges to $\mathbf{E}$.

Bipartite Cores: Bipartite cores consist of non-empty, non-intersecting node-sets L and R for which there only exist edges from L and R, but not within L or R. Note that stars can be construed as a fixed case of bipartite cores in which $|L| = 1$. The encoding cost $L(bc)$ for a full bipartite core *bc* is as follows:

$$L(fb) \;=\; L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2\binom{n}{|L|} + log_2\binom{n}{|R|}$$

In this case, we encode the number of nodes in L and R followed by the node ids in each set.

As with near cliques, near bipartite cores are also interesting if they stand out from the background. In this case, encoding is given analogously as follows:

$$\begin{aligned} L(nb) &= L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2\binom{n}{|L|} + log_2\binom{n}{|R|} \\ &\quad + log_2(|area(nb)|) + ||nb||\rho_1 + ||nb||'\rho_0 \end{aligned}$$

Furthermore, as with near-cliques, encoding in this case is exact so we do not add any edges to **E**.

Chains: A chain is characterized by series of nodes in which each node has an edge connecting it to the next node – for example, consider the node-set $\{1, 2, 3, 4\}$ in which 1 is connected to 2, 2 is connected to 3, and 3 is connected to 4. Given the right permutation, a perfect chain in an undirected graph will have edges only along two diagonals of the adjacency matrix. For a chain *ch*, we have the encoding cost $L(ch)$ as follows:

$$L(ch) = L_{\mathbb{N}}(|ch| - 1) + \sum_{i=1}^{|ch|} log_2(n - i + 1)$$

We first encode the number of nodes in the chain, followed by their node ids in order of connection.

While in this chapter, we only discuss the above set of basic graph structures, any arbitrary graph pattern can be similarly handled provided that we carefully derive an expression to losslessly encode its connectivity.

6.3.2.2 Encoding Temporal Presence

For a given phrase $v \in \Phi$, it is not sufficient to only encode the connectivity of the underlying static structure. We must also encode the temporal presence $u(s)$, consisting of a set of ordered timesteps in which s appears, for each structure. In this section, we describe how to compute the encoding cost $L(u(s))$ for each of the temporal signatures in the signature set Δ.

We note that describing a set of timesteps $u(s)$ in terms of temporal signatures in Δ is yet another model selection problem for which we can leverage MDL. As with connectivity encoding, labeling $u(s)$ with a given temporal signature may not be *precisely* accurate – however, any mistakes will add to the cost of transmitting the error. Errors in temporal presence encoding will be further detailed in Sec. 6.3.3.2.

Oneshot: Oneshot structures appear at only one timestep in $G_1 \cdots G_t$ – that is, $|u(s)| = 1$. These structures represent graph anomalies, in the sense that they are non-recurrent interactions which are only observed once. The encoding cost $L(o)$ for the temporal presence of a oneshot structure o can be written as:

$$L(o) = log_2(t)$$

As the structure occurs only once, we only have to identify the timestep of occurrence from the t observed timesteps.

Ranged: Ranged structures are characterized by a short-lived existence. These structures appear for several timesteps in a row before disappearing again – they are defined by a single burst of activity. The encoding cost $L(r)$ for a ranged structure r is given by:

$$L(r) \quad = \quad L_{\mathbb{N}}(|u(s)|) + log_2 \binom{t}{2}$$

We first encode the number of timesteps in which the structure occurs, followed by the timestep ids of both the start and end timestep marking the span of activity.

Periodic: Periodic structures are an extension of ranged structures in that they appear at fixed intervals. However, these intervals are spaced greater than one timestep apart. As such, the same encoding cost function we use for ranged structures suffices here. That is, $L(p)$ for a periodic structure p is given by $L(p) = L(r)$.

For both ranged and periodic structures, periodicity can be inferred from the start and end markers along with the number of timesteps $|u(s)|$, allowing reconstruction of the original $u(s)$.

Flickering: A structure is flickering if it appears only in some of the $G_1 \cdots G_t$ timesteps, and does so without any discernible ranged/periodic pattern. The encoding cost $L(f)$ for a flickering structure f is as follows:

$$L(f) \quad = \quad L_{\mathbb{N}}(|u(s)|) + log_2 \binom{n}{|u(s)|}$$

We encode the number of timesteps in which the structure occurs in addition to the ids for the timesteps of occurrence.

Constant: Constant structures persist throughout all timesteps. That is, they occur at each timestep $G_1 \cdots G_t$ without exception. In this case, our encoding cost $L(c)$ for a constant structure c is defined as $L(c) = 0$. Intuitively, information regarding the timesteps in which the structure appears is "free," as it is already given by encoding the phrase descriptor $v(s)$.

6.3.3 Encoding the Errors

Given that M is a summary and the $\mathbf{M}$ induced by M is only an approximation of $\mathbf{A}$, it is necessary to encode errors made by M. In particular, there are two types of errors we must consider. The first is error in connectivity – that is, if $area(s)$ induced by structure s is not *exactly* the same as the associated patch in $\mathbf{A}$, we encode the relevant mistakes. The second is the error induced by encoding the set of timesteps $u(s)$ with a fixed temporal signature, given that $u(s)$ may not precisely follow the temporal pattern used to encode it.

6.3.3.1 Encoding Errors in Connectivity

We encode the error tensor $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ as two different pieces – specifically, we encode $\mathbf{E}^+$ and $\mathbf{E}^-$ where the former refers to the area of $\mathbf{A}$ which M models and $\mathbf{M}$ includes extraneous edges not present in the original graph, and the latter consists of the area of $\mathbf{A}$ which M does not model and therefore does not describe. Our reasoning for encoding these two separately is that they likely have different error distributions. Given that near cliques and near bipartite cores are encoded exactly per our model, we ignore the associated areas when encoding $\mathbf{E}^+$. The encoding for $\mathbf{E}^+$ and $\mathbf{E}^-$, denoted as $L(\mathbf{E}^+)$ and $L(\mathbf{E}^-)$ respectively is as follows:

$$\begin{aligned} L(\mathbf{E}^+) &= log_2(|\mathbf{E}^+|) + ||\mathbf{E}^+||\rho_1 + ||\mathbf{E}^+||'\rho_0 \\ L(\mathbf{E}^-) &= log_2(|\mathbf{E}^-|) + ||\mathbf{E}^-||\rho_1 + ||\mathbf{E}^-||'\rho_0 \end{aligned}$$

In both cases, we encode the number of 1s in $\mathbf{E}^+$ (or $\mathbf{E}^-$), followed by the actual 1s and 0s using optimal prefix codes.

6.3.3.2 Encoding Errors in Temporal Presence

For encoding errors induced by identifying $u(s)$ as one of the temporal signatures, we turn to optimal prefix codes applied over the error distribution for each structure s. Given the information encoded for each signature type in Δ, we can reconstruct an approximation $\tilde{u}(s)$ of the original timesteps $u(s)$ such that $|u(s)| = |\tilde{u}(s)|$. Using this approximation, the encoding cost $L(e_u(s))$ for the error $e_u(s) = u(s) - \tilde{u}(s)$ is defined as:

$$L(e_u(s)) = \sum_{k \in h(e_u(s))} \big(log_2(k) + log_2 c(k) + c(k)\rho_k\big)$$

where $h(e_u(s))$ denotes the set of elements with unique magnitude in $e_u(s)$, $c(k)$ denotes the count of element k in $e_u(s)$ and ρ_k denotes the length of the optimal prefix code for k. For each magnitude error, we encode the magnitude of the error, the number of times it occurs and the actual errors using optimal prefix codes. Using the model in conjunction with temporal presence and connectivity errors, a recipient can first recover the $u(s)$ for each $s \in M$, approximate $\mathbf{A}$ with $\mathbf{M}$ induced by M, produce $\mathbf{E}$ from $\mathbf{E}^+$ and $\mathbf{E}^-$, and finally recover $\mathbf{A}$ losslessly through $\mathbf{A} = \mathbf{M} \oplus \mathbf{E}$.

Remark: For a dynamic graph G of n nodes, the search space $\mathcal{M}$ for the best model $M \in \mathcal{M}$ is intractable, as it consists of all permutations of all possible temporal structures over the lexicon Φ, over all possible subsets over the node-set $\mathcal{V}$ and over all possible graph timesteps $G_1 \cdots G_t$. Furthermore, $\mathcal{M}$ is not easily exploitable for efficient search. As a result, we propose several practical approaches for the purpose of finding good and interpretable temporal models/summaries for G.

6.4 Proposed Method: TimeCrunch

Thus far, we have described our strategy of formulating dynamic graph summarization as a problem in a compression context for which we can leverage MDL. Specifically, we have detailed how to encode a model and the associated error which can be used to losslessly reconstruct the original dynamic graph G. Our models are characterized by ordered lists of temporal structures which are further classified as *phrases* from the lexicon Φ – that is, each $s \in M$ is identified by a phrase $p \in \Phi$ – over the node connectivity $c(s)$ (an induced set of edges depending on the

Algorithm 6.1: TimeCrunch

```
1: **Generating Candidate Static Structures**: Generate static subgraphs for each $G_1 \cdots G_t$
   using traditional static graph decomposition approaches.
2: **Labeling Candidate Static Structures**: Label each static subgraph as a static structure
   corresponding to the identifier $x \in \Omega$ which minimizes the *local encoding cost*.
3: **Stitching Candidate Temporal Structures**: *Stitch* the static structures from $G_1 \cdots G_t$
   together to form temporal structures with coherent connectivity behavior and label them
   according to the the phrase $p \in \Phi$ which minimizes temporal presence encoding cost.
   Populate the candidate set $\mathcal{C}$.
4: **Composing the Summary**: Compose a model $M$ of important, non-redundant temporal
   structures which summarize $G$ using the Vanilla, Top-10, Top-100 and Stepwise heuristics.
   Choose $M$ associated with the heuristic that produces the smallest total encoding cost.
```

static structure identifier st, fc, etc.) and the associated temporal presence $u(s)$ (ordered list of timesteps captured by a temporal signature o, r, etc. and deviations) in which the temporal structure is active, while the error consists of those edges which are not covered by $\mathbf{M}$, or the approximation of $\mathbf{A}$ induced by M.

Next, we discuss how we find good candidate temporal structures to populate the candidate set $\mathcal{C}$, as well as how we find the best model M with which to summarize our dynamic graph. The pseudocode for our algorithm is given in Alg. 6.1 and the next subsections detail each step of our approach.

6.4.1 Generating Candidate Static Structures

TimeCrunch takes an incremental approach to dynamic graph summarization. Our approach begins by considering potentially useful subgraphs over static graphs $G_1 \cdots G_t$. Sec. 6.2 mentions several such algorithms for community detection and clustering including EigenSpokes, METIS, SlashBurn, etc. Summarily, for each $G_1 \cdots G_t$, a set of subgraphs $\mathcal{F}$ is produced.

6.4.2 Labeling Candidate Static Structures

Once we have the set of static subgraphs from $G_1 \cdots G_t$, $\mathcal{F}$, we next seek to label each subgraph in $\mathcal{F}$ according to the static structure identifiers in Ω that best fit the connectivity for the given subgraph. That is, for each subgraph construed as a set of nodes $\mathcal{L} \in \mathcal{V}$ for a fixed timestep, does the adjacency matrix of $\mathcal{L}$ best resemble a star, near or full clique, near or full bipartite core or a chain? To answer this question, we leverage the encoding scheme discussed in Sec. 6.3.2.1: we try encoding the subgraph $\mathcal{L}$ using each of the static identifiers in Ω and label it with the identifier $x \in \Omega$ which minimizes the encoding cost.

Consider the model ω which consists of only the subgraph $\mathcal{L}$ and a yet to be determined static identifier. In practice, instead of computing the global encoding cost $L(G, \omega)$ when encoding $\mathcal{L}$ as each static identifier in Ω to find the best fit, we compute the *local* encoding cost defined as $L(\omega) + L(\mathbf{E}_\omega^+) + L(\mathbf{E}_\omega^-)$ where $L(\mathbf{E}_\omega^+)$ and $L(\mathbf{E}_\omega^-)$ indicate the encoding costs for the extraneous and unmodeled edges for the subgraph $\mathcal{L}$ respectively. This is done for purpose of efficiency

– intuitively, however, the static identifier that best describes $\mathcal{L}$ is independent of the edges outside of $\mathcal{L}$.

The challenge in this labeling step is that before we can encode $\mathcal{L}$ as any type of identifier, we must identify a suitable permutation of nodes in the subgraph so that our model encodes the correct edges. For example, if $\mathcal{L}$ is a star, which is the hub? Or if $\mathcal{L}$ is a bipartite core, how can we distinguish the parts?

For stars, we identify the highest-degree node as the hub and all other nodes as spokes. For near and full bipartite cores, finding the right permutation can be reduced to finding the maximum bipartite subgraph, which is equivalent to finding the maximum cut and is NP-hard. As a result, we use a heuristic approach which formulates the problem as a two-class classification task. To this end, we initialize L to contain the highest-degree node in $\mathcal{L}$, and R to contain its neighbors. We then use Fast Belief Propagation [KKK+11] with heterophily (assuming connected nodes belong to different classes) to propagate the class labels and determine L and R. For near and full cliques, any permutation is equally good. Lastly, for chains, finding the right permutation is equivalent to finding the longest path, which is NP-hard. As a result, we again employ a heuristic approach in which we select a node in $\mathcal{L}$ at random, use BFS to find the furthest node away, and repeat with the resulting node while extending the chain through local search iteratively. For both near cliques and bipartite cores, we do not encode $\mathbf{E}^+_{nc}$ and $\mathbf{E}^+_{nb}$ as $L(nc)$ and $L(nb)$ encode the relevant edges exactly.

6.4.3 Stitching Candidate Temporal Structures

Thus far, we have a set of static subgraphs $\mathcal{F}$ over $G_1 \cdots G_t$ labeled with the associated static identifiers which best represent subgraph connectivity (from now on, we refer to $\mathcal{F}$ as a set of static *structures* instead of *subgraphs* as they have been labeled with identifiers). From this set, our goal is to find meaningful temporal structures – namely, we seek to *find* static subgraphs which have the same patterns of connectivity over one or more timesteps and *stitch* them together. Thus, we formulate the problem of finding coherent temporal structures in G as a clustering problem over $\mathcal{F}$. Though there are several criteria we could use for clustering static structures together, we employ the following based on their intuitive meaning: two structures in the same cluster should have (a) substantial overlap in the node-sets composing their respective subgraphs, and (b) exactly the same, or similar (full and near clique, or full and near bipartite core) static structure identifiers. These criteria, if satisfied, allow us to find groups of nodes that share interesting connectivity patterns over time.

Conducting the clustering by naively comparing each static structure in $\mathcal{F}$ to the others will produce the desired result, but is *quadratic* on the number of static structures and is thus undesirable from a scalability point of view. Instead, we propose an incremental approach using repeated rank-1 Singular Value Decomposition (SVD) for clustering the static structures, which offers *linear* time complexity on the number of edges m in G.

We begin by defining $\mathbf{B}$ as the *structure-node membership matrix* (SNMM) of G. $\mathbf{B}$ is defined to be of dimensions $|\mathcal{F}| \times |\mathcal{V}|$, where $\mathbf{B}_{i,j}$ indicates whether the ith row (structure) in $\mathcal{F}$ ($\mathbf{B}$) contains node j in its node-set. Thus, $\mathbf{B}$ is a matrix indicating the membership of nodes in $\mathcal{V}$ to each of the static structures in $\mathcal{F}$. We note that any two equivalent rows in $\mathbf{B}$ are characterized by structures that share the same node-set (but possibly different static identifiers). As our clustering criteria mandate that we cluster only structures with the same or similar static identifiers, in our algorithm, we construct 4 SNMMs – $\mathbf{B}_{st}$, $\mathbf{B}_{cl}$, $\mathbf{B}_{bc}$ and $\mathbf{B}_{ch}$ corresponding to the associated matrices for stars, near and full cliques, near and full bipartite cores and chains respectively. Now, any two equivalent rows in $\mathbf{B}_{cl}$ are characterized by structures that share the same-node set and the same, or similar static identifiers, and analogue for the other matrices. Next, we utilize SVD to cluster the rows in each SNMM, effectively clustering the structures in $\mathcal{F}$.

Recall that the rank-k SVD of an $m \times n$ matrix $\mathbf{A}$ factorizes $\mathbf{A}$ into 3 matrices – the $m \times k$ matrix of left-singular vectors $\mathbf{U}$, the $k \times k$ diagonal matrix of singular values $\mathbf{\Sigma}$ and the $n \times k$ matrix of right-singular vectors $\mathbf{V}$, such that $\mathbf{A} = \mathbf{U\Sigma V^T}$. A rank-$k$ SVD effectively reduces the input data into the best k-dimensional representation, each of which can be mined separately for clustering and community detection purposes. However, one major issue with using SVD in this fashion is that identifying the desired number of clusters k upfront is a non-trivial task. To this end, [PSB13] evidences that in cases where the input matrix is sparse, repeatedly clustering using k rank-1 decompositions and adjusting the input matrix accordingly approximates the batch rank-k decomposition. This is a valuable result in our case – as we do not initially know the number of clusters needed to group the structures in $\mathcal{F}$, we eliminate the need to define k altogether by repeatedly applying rank-1 SVD using power iteration and removing the discovered clusters from each SNMM until all clusters have been found (when all SNMMs are fully sparse and thus *deflated*). However, in practice, full deflation is unneeded for summarization purposes, as most "important" clusters are found in early iterations due to the nature of SVD. For each of the SNMMs, the matrix $\mathbf{B}$ used in the $(i+1)^{th}$ iteration of this iterative process is computed as

$$\mathbf{B}^{i+1} = \mathbf{B}^{i} - I^{\mathcal{G}_i} \circ \mathbf{B}^{i}$$

where $\mathcal{G}_i$ denotes the set of row ids corresponding to the structures which were clustered together in iteration i, $I^{\mathcal{G}_i}$ denotes the indicator matrix with 1s in rows specified by $\mathcal{G}_i$ and $\circ$ denotes the Hadamard matrix product. This update to $\mathbf{B}$ is needed between iterations, as without subtracting out the previously-found cluster, repeated rank-1 decompositions would find the same cluster ad infinitum and the algorithm would not converge.

Although this algorithm works assuming we can remove a cluster in each iteration, the question of how we find this cluster given a singular vector has yet to be answered. First, we sort the singular vector, permuting the rows by magnitude of projection. The intuition is that the structure (rows) which projects most strongly to that cluster is the best representation of the cluster, and is considered a *base* structure which we attempt to find matches for. Starting from the base structure, we iterate down the sorted list and compute the Jaccard similarity, defined as $J(\mathcal{L}_1, \mathcal{L}_2) = |\mathcal{L}_1 \cap \mathcal{L}_2|/|\mathcal{L}_1 \cup \mathcal{L}_2|$ for node-sets $\mathcal{L}_1$ and $\mathcal{L}_2$, between each structure and the base. Other structures which are composed of the same, or similar node-sets will also project strongly to the cluster, and be stitched to the base. Once we encounter a series of structures which fail to match by a predefined similarity criterion, we adjust the SNMM and continue with the next iteration.

Having stitched together the relevant static structures, we label each temporal structure using the temporal signature in Δ and resulting phrase in Φ which minimizes its encoding cost using the temporal encoding framework derived in Sec. 6.3.2.2. We use these temporal structures to populate the candidate set $\mathcal{C}$ for our model.

6.4.4 Composing the Summary

Given the candidate set of temporal structures $\mathcal{C}$, we next seek to find the model M which best summarizes G. However, actually finding the best model is combinatorial, as it involves considering all possible permutations of subsets of $\mathcal{C}$ and choosing the one which gives the smallest encoding cost. As a result, we propose several heuristics that give fast and approximate solutions without entertaining the entire search space. To reduce the search space, we associate with each temporal structure a metric by which we measure quality, called the *local encoding benefit*. The local encoding benefit is defined as the ratio between the cost of encoding the given temporal structure as error and the cost of encoding it using the best phrase (local encoding cost). Large local encoding benefits indicate high compressibility, and thus meaningful structure in the underlying data. Our proposed heuristics are as follows:

Vanilla: This is the baseline approach, in which our summary contains all the structures from the candidate set, or $M = \mathcal{C}$.

Top-k: In this approach, M consists of the top k structures of $\mathcal{C}$, sorted by local encoding benefit.

Stepwise: This approach involves considering each structure of $\mathcal{C}$, sorted by local encoding benefit, and adding it to M if the global encoding cost decreases. If adding the structure to M increases the global encoding cost, the structure is discarded as redundant or not worthwhile for summarization purposes.

In practice, TimeCrunch uses each of the heuristics and identifies the best summary for G as the one that produces the minimum encoding cost.

6.5 Experiments

In this section, we evaluate TimeCrunch and seek to answer the following questions: Are real-world dynamic graphs well-structured, or noisy and indescribable? If they are structured, how so – what temporal structures do we see in these graphs and what do they mean? Lastly, is TimeCrunch scalable?

6.5.1 Datasets and Experimental Setup

For our experiments, we use 5 real dynamic graph datasets – they are summarized in Table 6.3 and described below.

Enron: The `Enron` e-mail dataset is publicly available. It contains 20 thousand unique links between 151 users based on e-mail correspondence, over 163 weeks (May 1999 - June 2002).

Graph	Nodes	Edges	Timesteps
`Enron` [SA04]	151	20 thousand	163 weeks
`Yahoo-IM` [Yah]	100 thousand	2.1 million	4 weeks
`Honeynet`	372 thousand	7.1 million	32 days
`DBLP` [dbl14]	1.3 million	15 million	25 years
`Phonecall`	6.3 million	36.3 million	31 days

Table 6.3: Dynamic graphs used for empirical analysis

Yahoo! IM: The `Yahoo-IM` dataset is publicly available. It contains 2.1 million sender-receiver pairs between 100 thousand users over 5709 zip-codes selected from the Yahoo! messenger network over 4 weeks starting from April 1st, 2008.

Honeynet: The `Honeynet` dataset is not publicly available. It contains information about network attacks on *honeypots* (i.e., computers which are left intentionally vulnerable to attackers) It contains source IP, destination IP and attack timestamps of 372 thousand (attacker and honeypot) machines with 7.1 million unique daily attacks over a span of 32 days starting from December 31st, 2013.

DBLP: The `DBLP` computer science bibliography is publicly available, and contains yearly co-authorship information, indicating joint publication. We used a subset of DBLP spanning 25 years, from 1990 to 2014, with 1.3 million authors and 15 million unique author-author collaborations over the years.

Phonecall: The `Phonecall` dataset is not publicly available. It describes the who-calls-whom activity of 6.3 million individuals from a large, anonymous Asian city and contains a total of 36.3 million unique daily phonecalls. It spans 31 days, starting from December 1st, 2007.

In our experiments, we use SlashBurn for generating candidate static structures, as it is scalable and designed to extract structure from real-world, non-"cavemen" graphs. We note that including other graph decomposition methods can only improve results given MDL. Furthermore, when clustering each sorted singular vector during the stitching process, we move on with the next iteration of matrix deflation after 10 failed matches with a Jaccard similarity threshold of 0.5 – we choose 0.5 based on experimental results which show that it gives the best encoding cost and balances between excessively terse and overlong (error-prone) models. Lastly, we run TimeCrunch for a total of 5000 iterations for all graphs (each iteration uniformly selects one SNMMs to mine, resulting in 5000 total temporal structures), except for the `Enron` graph which is fully deflated after 563 iterations and the `Phonecall` graph which we limit to 1000 iterations for efficiency.

6.5.2 Quantitative Analysis

In this section, we use TimeCrunch to summarize each of the real-world dynamic graphs from Table 6.3 and report the resulting encoding costs. Specifically, evaluation is done by comparing

Graph	Original (bits)	TimeCrunch			
		Vanilla	Top-10	Top-100	Stepwise
Enron	86, 102	89% (563)	88%	81%	**78%** (130)
Yahoo-IM	16, 173, 388	97% (5000)	99%	98%	**93%** (1523)
Honeynet	72, 081, 235	82% (5000)	96%	89%	**81%** (3740)
DBLP	167, 831, 004	97% (5000)	99%	99%	**96%** (1627)
Phonecall	478, 377, 701	100% (1000)	100%	99%	**98%** (370)

Table 6.4: TimeCrunch finds temporal structures that can compress real graphs. Original denotes the cost in bits for encoding each graph with an empty model. Columns under TimeCrunch show relative costs for encoding the graphs using the respective heuristic (size of model is parenthesized). The lowest description cost is bolded.

the compression ratio between encoding costs of the resulting models to the null encoding (Original) cost, which is obtained by encoding the graph using an empty model.

We note that although we provide results in a compression context, compression is *not* our main goal for TimeCrunch, but rather the means to our end for identifying suitable structures with which to summarize dynamic graphs and route the attention of practitioners. For this reason, we do not evaluate against other, compression-oriented methods which prioritize leveraging any correlation within the data to reduce cost and save bits. Other temporal clustering and community detection approaches which focus only on extracting dense blocks are also not compared to for similar reasons.

In our evaluation, we consider (a) Original and (b) TimeCrunch summarization using the proposed heuristics. In the Original approach, the entire adjacency tensor is encoded using the empty model $M = \emptyset$. As the empty model does not describe any part of the graph, all the edges are encoded using $L(\mathbf{E}^-)$. We use this as a baseline to evaluate the savings attainable using TimeCrunch. For summarization using TimeCrunch, we apply the Vanilla, Top-10, Top-100 and Stepwise model selection heuristics. We note that we ignore small structures of <5 nodes for Enron and <8 nodes for the other, larger datasets.

Table 6.4 shows the results of our experiments in terms of encoding costs of various summarization techniques as compared to the Original approach. Smaller compression ratios indicate better summaries, with more structure explained by the respective models. For example, Stepwise was able to encode the Enron dataset using just 78% of the bits compared to 89% using Vanilla. In our experiments, we find that the Stepwise heuristic produces models with considerably fewer structures than Vanilla, while giving even more concise graph summaries (Fig. 6.2). This is because it is highly effective in pruning redundant, overlapping or error-prone structures from the candidate set $\mathcal{C}$, by evaluating new structures in the context of previously seen ones.

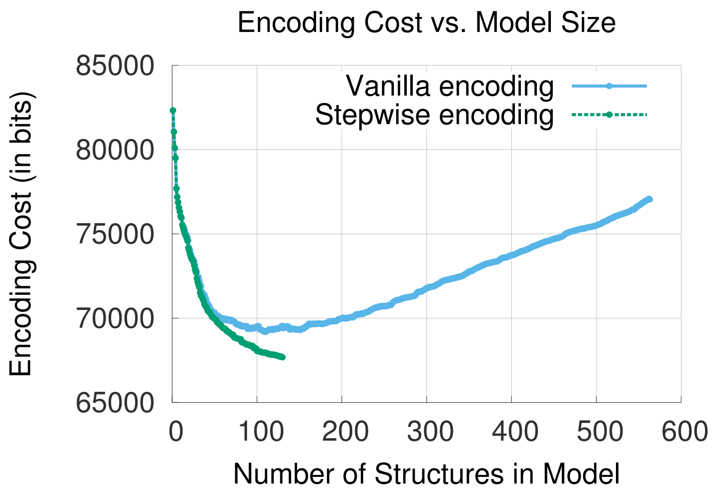


Figure 6.2: **TimeCrunch-Stepwise summarizes** `Enron` **using just 78% of Original's bits and 130 structures, compared to 89% and 563 structures of TimeCrunch-Vanilla by pruning unhelpful structures from the candidate set.**

Observation 6.1: Structure in Dynamic Graphs

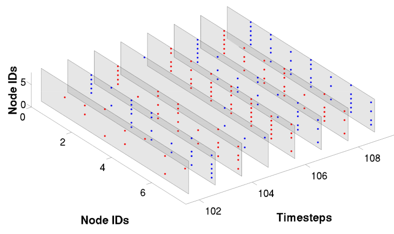
Real-world dynamic graphs are not unstructured. TimeCrunch gives better encoding cost than Original, indicating the presence of temporal graph structure.

6.5.3 Qualitative Analysis

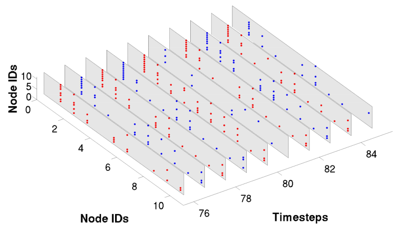
In this section, we discuss qualitative results from applying TimeCrunch to the graphs mentioned in Table 6.3.

Enron: The `Enron` graph is characteristic of many periodic, ranged and oneshot stars and several periodic and flickering cliques. Periodicity is reflective of office e-mail communications (e.g. meetings, reminders). Figure 6.3a shows an excerpt from one flickering clique which corresponds to the several members of Enron's legal team, including Tana Jones, Susan Bailey, Marie Heard and Carol Clair – all lawyers at Enron. Figure 6.3b shows an excerpt from a flickering star, corresponding to many of the same members as the flickering clique – the center of this star was identified as the boss, Tana Jones (Enron's Senior Legal Specialist). Note that the satellites of the star oscillate over time. Interestingly, the flickering star and clique extend over most of the observed duration. Furthermore, several of the oneshot stars corresponds to company-wide emails sent out by key players John Lavorato (Enron America CEO), Sally Beck (COO) and Kenneth Lay (CEO/Chairman).

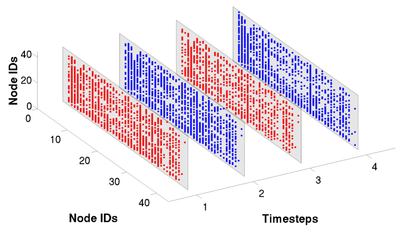
Yahoo! IM: The `Yahoo-IM` graph is composed of many temporal stars and cliques of all types, and several smaller bipartite cores with just a few members on one side (indicative of friends who share mostly similar friend-groups but are themselves unconnected). We observe several



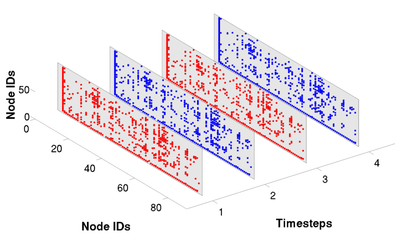
(a) 8 employees of the `Enron` legal team forming a *flickering near clique*



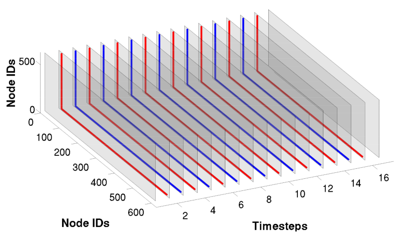
(b) 10 employees of the `Enron` legal team forming a *flickering star* with the boss as the hub



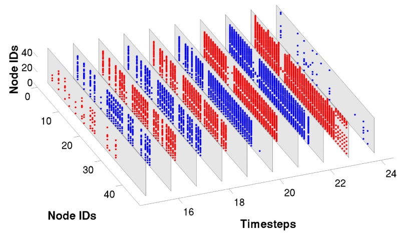
(c) 40 users in `Yahoo-IM` forming a *constant near clique* with 55% density over the observed 4 weeks



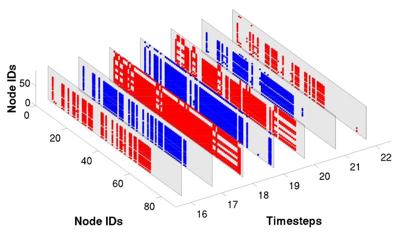
(d) 82 users in `Yahoo-IM` forming a *constant star* over the observed 4 weeks



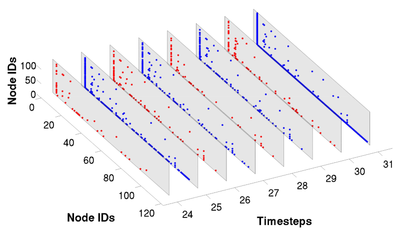
(e) 589 honeypot machines were attacked on `Honeynet` over 2 weeks, forming a *ranged star*



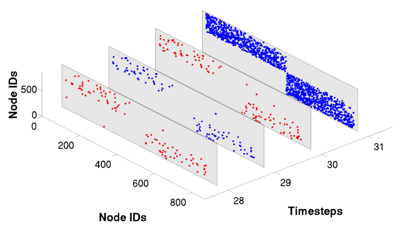
(f) 43 authors that publish together in biotechnology journals forming a *ranged near clique* on `DBLP`



(g) 82 authors forming a *ranged near clique* on `DBLP`, with burgeoning collaboration from timesteps 18-20 (2007-2009)



(h) 111 callers in `Phonecall` forming a *periodic star* appearing strongly on odd numbered days, especially Dec. 25 and 31



(i) 792 callers in `Phonecall` forming a *oneshot near bipartite core* appearing strongly on Dec. 31

Figure 6.3: **TimeCrunch finds meaningful temporal structures in real graphs.** We show the reordered subgraph adjacency matrices over multiple timesteps. Individual timesteps are outlined in gray, and edges are plotted with alternating red and blue color for discernibility.

interesting patterns in this data – Fig. 6.3d corresponds to a constant star with a hub that communicates with 70 users consistently over 4 weeks. We suspect that these users are part of a small office network, where the boss uses group messaging to notify employees of important updates or events – we notice that very few edges of the star are missing each week and the average degree of the satellites is roughly 4, corresponding to possible communication between employees. Figure 6.3c depicts a constant clique between 40 users, with an average density over 55% – we suspect that these may be spam-bots messaging each other in an effort to appear normal.

	st	fc	ch
r	9	-	-
p	93	7	1
f	3	1	-
c	-	-	-
o	15	1	-

(a) `Enron`

	st	fc	nc	bc	nb	ch
r	147	43	-	1	45	6
p	59	25	-	-	42	3
f	179	55	-	1	62	3
c	185	118	-	-	66	-
o	295	129	1	2	56	-

(b) `Yahoo-IM`

	st	bc
r	56	-
p	125	1
f	39	-
c	-	-
o	3512	7

(c) `Honeynet`

	st	fc	nb	ch
r	43	80	-	5
p	19	26	-	-
f	1	-	-	-
c	-	-	-	-
o	516	840	97	-

(d) `DBLP`

	st	fc	nc	bc
r	15	-	-	-
p	68	-	-	1
f	88	-	-	-
c	5	-	-	-
o	187	4	1	1

(e) `Phonecall`

Table 6.5: Frequency of each temporal structure type discovered using TimeCrunch-Stepwise for each dataset.

Honeynet: `Honeynet` is a bipartite graph between attacker and honeypot (victim) machines. As such, it is characterized by temporal stars and bipartite cores. Many of the attacks only span a single day, as indicated by the presence of 3512 oneshot stars, and no attacks span the entire 32 day duration. Interestingly, 2502 of these oneshot star attacks (71%) occur on the first and second observed days (Dec. 31 and Jan. 1st) indicating intentional "new-year" attacks. Figure 6.3e shows a ranged star, lasting 15 consecutive days and targeting 589 machines for the entire duration of the attack.

DBLP: Agreeing with intuition, `DBLP` consists of a large number of oneshot temporal structures corresponding to many single instances of joint publication. However, we also find numerous ranged/periodic stars and cliques which indicate coauthors publishing in consecutive years or intermittently. Figure 6.3f shows a ranged clique spanning from 2007-2012 between 43 coauthors who jointly published each year. The authors are mostly members of the NIH NCBI (National Institute of Health National Center for Biotechnology Information) and have published their work in various biotechnology journals such as *Nature*, *Nucleic Acids Research* and *Genome Research*. Figure 6.3g shows another ranged clique from 2005 to 2011, consisting of 83 coauthors who jointly publish each year, with an especially collaborative 3 years (timesteps 18-20) corresponding to 2007-2009 before returning to status quo.

Phonecall: The `Phonecall` dataset is largely comprised of temporal stars and few dense clique and bipartite structures. Again, we have a large proportion of oneshot stars which occur only at single timesteps. Further analyzing these results, we find that 111 of the 187 oneshot stars

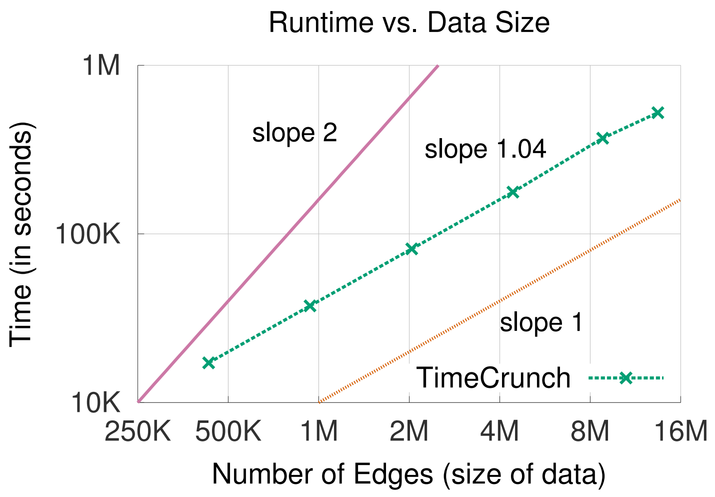


Figure 6.4: **TimeCrunch scales near-linearly on the number of edges in the graph.** Here, we use several induced temporal subgraphs from DBLP, up to 14M edges in size.

(59%) are found on Dec. 24, 25 and 31st, corresponding to Christmas Eve/Day and New Year's Eve holiday greetings. Furthermore, we find many periodic and flickering stars typically consisting of 50-150 nodes, which may be associated with businesses regularly contacting their clientele, or public phones which are used consistently by the same individuals. Figure 6.3h shows one such periodic star of 111 users over the last week of December, with particularly clear star structure on Dec. 25th and 31st and other odd-numbered days, accompanied by substantially weaker star structure on the even-numbered days. Figure 6.3i shows an oddly well-separated oneshot near-bipartite core which appears on Dec. 31st, consisting of two roughly equal-sized parts of 402 and 390 callers. Though we do not have ground truth to interpret these structures, we note that a practitioner with the appropriate information could better interpret their meaning.

6.5.4 Scalability

All components of TimeCrunch are carefully designed to be linear or near-linear on the number of nonzero edges. Figure 6.4 shows the near-linear runtime of TimeCrunch on several induced temporal subgraphs (up to 14M edges) taken from the DBLP dataset at varying time-intervals. Our experiments were conducted on a machine with 80 Intel Xeon(R) 4850 2GHz cores and 256GB RAM. We use MATLAB for candidate subgraph generation and temporal stitching and Python for model selection heuristics.

Furthermore, much of the TimeCrunch pipeline (per-timestep summarization) is embarrassingly parallelizable and can be easily split over nodes. Distributed eigensolver implementations also exist in practice for the stitching component.

6.6 Conclusion

In this work, we tackle the problem of identifying significant and structurally interpretable temporal patterns in large, dynamic graphs. Specifically, we formalize the problem of finding important and coherent temporal structures in a graph as *minimizing the encoding cost* of the graph from a compression standpoint. To this end, we propose TimeCrunch, a fast and effective, incremental technique for building interpretable summaries for dynamic graphs which involves *generating* candidate subgraphs from each static graph, *labeling* them using static identifiers, *stitching* them over multiple timesteps and *composing* a model using practical approaches. Finally, we apply TimeCrunch on several large, dynamic graphs and find numerous patterns and anomalies which indicate that real-world graphs *do* in fact exhibit temporal structure.

Based on content published in [JST+17].

Chapter 7

M3A: Modeling Interarrival Time in Web Searches

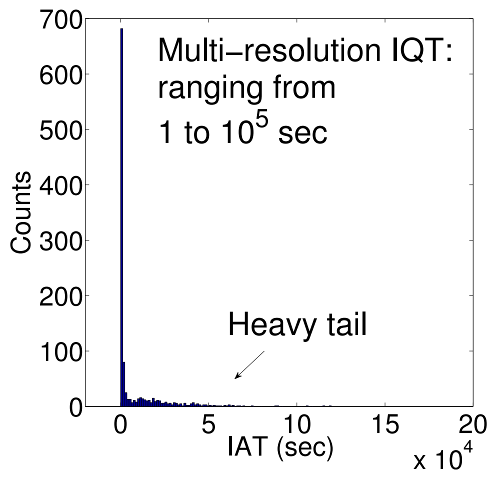
Modeling the temporal search behavior of users and finding inhuman abnormalities.

How do real users use search platforms to look for content, and how often do they form new queries? Can we characterize normal and abnormal user behavior on such platforms based on their temporal usage habits? These are the questions we aim to answer in this chapter. Specifically, we study multiple large query and online commenting logs and propose a new, intuitive and empirically effective approach called M3A, which models inter-arrival times (IATs) between individual user queries (Camel-Log) and group parameter distributions (Meta-Click) and can be used for anomaly detection. M3A models real IATs more accurately than existing approaches and discovers abnormal search behaviors in line with human intuition.

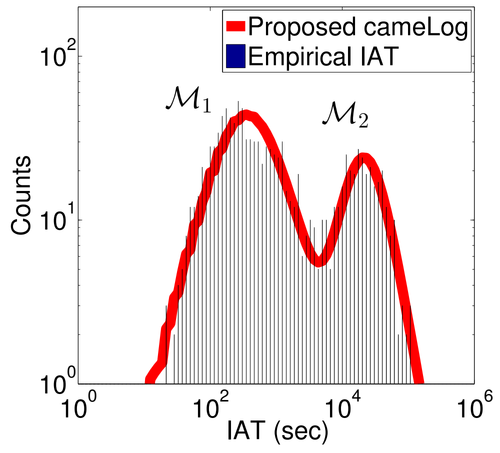
7.1 Introduction

Say a user submits one web search every five minutes, for three hours in a row – is this normal? How can we detect abnormal search behaviors amongst this user and other such users? Is there any distinct pattern in these users' search behaviors? These three questions serve as the major motivations of this work.

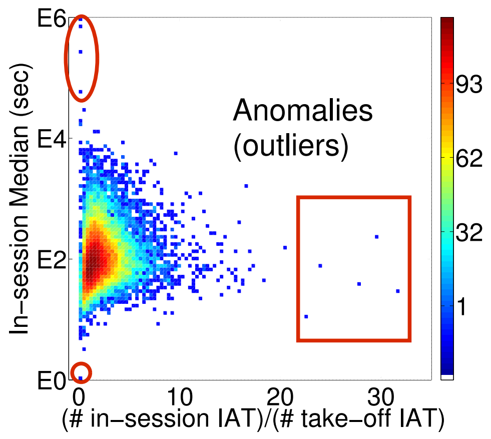
Conventionally, each of a user's queries is assumed (1) to be submitted independently and (2) to follow a constant rate λ, which results in a simple and elegant model, Poisson process (PP). PP generates independent and identically distributed (i.i.d.) inter-arrival time (IAT) that follows an (negative) exponential distribution [FMH93]. In reality, however, does PP accurately model search behavior?



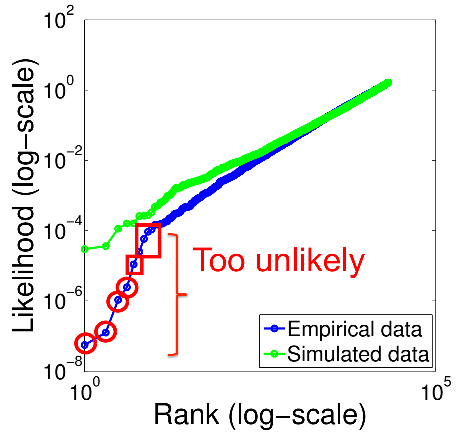
(a) Empirical IAT in lin. scale



(b) Empirical IAT in log scale and "Camel-Log" fit



(c) Group-level analysis



(d) Rank-weirdness plot

Figure 7.1: **M3A models search patterns and detects anomalous behaviors.**: (a) Histogram of inter-arrival time (IAT) for a single user in linear scale. No prevailing patterns are shown. (b) Logarithmic binning (equally-spaced in log-scale) of IAT with Camel-Log fit. A *bi-modal* distribution can be seen: $\mathcal{M}_1$ at 5 minutes (typical inter-query time) and $\mathcal{M}_2$ at hours (typical time between sessions). (c) illustrates group-level analysis with scatter plot of the ratio (in-session/take-off queries) vs. the median of in-session intervals. Anomalies are spotted: anomalies (circled by red) cannot be detected by using only the marginal PDF of X-variable, whereas anomalies (marked by the red rectangle) cannot be detected by using the Y-variable. (d) shows an automated way of spotting anomalies through Meta-Click: the blue deviants (within red circles/boxes) correspond to the outliers (in circles/boxes) in (c).

To answer this question, we investigate a large, industrial query log that contains more than 30 million queries submitted by 0.6 million users. Figure 7.1 illustrates the histogram of a user's

IAT. The temporal resolution is one second. As Figure 7.1(a) shows, this distribution has a "heavy tail" as opposed to an (negative) exponential distribution whose tail decays exponentially fast. In the logarithmic scale as Figure 7.1(b) shows, surprisingly, *two* distinct modes (denoted as $\mathcal{M}_1$ and $\mathcal{M}_2$) with approximately symmetric shapes can be seen. This distribution (or a mixture of distributions) clearly does not follow an (negative) exponential distribution, which has a strictly right-skewed shape in logarithmic scale and therefore cannot depict such shapes. This phenomenon suggests that the assumptions of PP rarely hold, since the arrival rate may change, or certain queries may be submitted depending on the previous queries. For example, in the query log, we found that a user first submitted a query (and clicked through) with the terms "wedding theme," and upon refining his/her previous query submitted a follow-up query with terms "African jungle wedding theme." These in-session query dependencies violate the assumptions and thus invalidate the use of PP.

Furthermore, detecting abnormal user behaviors is also a critical task for analyzing web activities. These anomalies—for example, frequently issuing queries with adverse clicks—may alter the corresponding search results, which in turn reduces the quality of recommendations. To make matter worse, this spam increases the load of datacenters, leading to extra operational expenses. Consequently, a systematic framework to spot web anomalies is urgently required.

In this paper we aim at solving the following problems:

- **P1: Pattern discovery and interpretation.** Is there any pattern in individual user IATs?
- **P2: Behavioral modeling.** How can we characterize the marginal distribution of IATs?
- **P3: Anomaly detection.** Given many users' IAT distributions, how can we determine whether a given user's IAT distribution is abnormal?

M3A's contributions are exactly the answers to these proposed questions:

- **A1: Pattern discovery and interpretation.** We observe that search IAT is a bi-modal ($\mathcal{M}_1$, $\mathcal{M}_2$) distribution (the superposition of two distributions) with $\mathcal{M}_1$ referring to *in-session*, and $\mathcal{M}_2$ referring to *take-off* (*e.g.*, sleep time) queries.
- **A2: Behavioral modeling.** Specifically, we propose:
    - "Camel-Log[1]" to parametrically characterize individual user IATs by mixing two heavy-tail distributions.
    - "Meta-Click" to describe the joint probability of two parameters of Camel-Log by using a copula.
- **A3: Anomaly detection.** Camel-Log generates IATs with the same statistical properties as in the real data shown in Figure 7.1(b), and Meta-Click can detect abnormal users in line with human intuition as in Figure 7.1(c)(d).
- **A4: Generality**. We conduct analysis and model user behaviors from AOL logs [PCT06] and show generality with respect to Reddit[2] comment posting behavior.

The remainder of this chapter is organized as follows. Section 7.2 provides the problem definition. Section 7.3 details the user-level model Camel-Log and Section 7.4 details the group-level

[1]The bi-modal distribution of a user's IAT is analogous to a baktrian **Camel**'s back, in **Log** scale.
[2]www.reddit.com

metamodel Meta-Click. Section 7.5 provides a guide to practical use of M3A. Section 7.6 surveys previous work. Finally, Section 7.7 concludes the chapter.

7.2 Problem Definition

In this work, we use a large-scale, industrial query log released by AOL [PCT06] and powered by Google [BI07]. The basic statistics of this query log are provided here:

- Duration: three months, from March 1st to May 31st, 2006[3].
- 36 millions queries submitted from 657,000 users:
    - 19 millions queries WITH click-through
      (we refer to these as *landed queries*).
    - 17 millions queries WITHOUT click-through
      (we refer to these as *orphan queries*).
- The temporal resolution is 1 second.

7.2.1 Terminology and problem formulation

Table 7.1 provides the symbols and the corresponding definitions used throughout this paper. By the convention in statistics, random variables are represented in upper-case (*e.g.*, M) and the corresponding values (*e.g.*, m) are in lower-case.

As mentioned in Section 7.1, we aim to solve the following three problems:

Problem 7.1: Pattern discovery and interpretation

Given each user ID and the time stamp of each query, **find** and *interpret* the most distinct pattern sufficient to characterize the IAT distribution of each user.

Problem 7.2: Behavioral modeling

Given the pattern found in P1:

1. **Design** a model (and a metamodel) that matches the statistical properties of the empirical data.
2. **Estimate** the parameters (and the hyper-parameters).

[3]Despite the datedness, the log faithfully reflects how humans actually interface with search (active bursts, sleep time, etc), as we will show later in this paper.

Symbol	Definition
IAT	Inter-arrival time
$t_{i,j}$	IAT between j^{th} and $(j+1)^{\text{th}}$ query submitted by user i.
$F_T(\cdot)$	Cumulative distribution function (CDF) for: (a) the random variable T or (b) the distribution T
$f_T(\cdot)$	Probability density function (PDF) for: (a) the random variable T or (b) the distribution T (*e.g.*, $f_{\mathcal{LL}}$ is the PDF of log-logistic)
$\mathcal{LL}$	Log-logistic distribution: a skewed (in linear scale), heavy-tail distribution
Camel-Log	Proposed mixture of two log-logistic distribution: modeling marginal IAT
Meta-Click	Proposed 2-d log-logistic distribution using Gumbel's copula: metamodeling the parameters of Camel-Log
Symbols used by Camel-Log	
α_{in}, β_{in}	Parameters: median and shape of log-logistic distribution (for modeling in-session IAT)
α_{off}, β_{off}	Parameters: median and shape of log-logistic distribution (for modeling take-off IAT)
θ	Proportion parameter: $\theta \in$ [0,1] for in-session IAT, and $(1-\theta)$ for take-off IAT
Symbols used by Meta-Click	
R	Random variable representing the ratio of in-session and take-off IAT: $R \triangleq \theta/(1-\theta)$
M	Random variable representing the log-median of in-session IAT: $M \triangleq \log(\alpha_{in})$
α_R, β_R	Hyper-parameters: median and shape of log-logistic distribution (for modeling R)
α_M, β_M	Hyper-parameters: median and shape of log-logistic distribution (for modeling M)
$C(\cdot,\cdot)$	Copula: Joint CDF of two random variables considering their dependency $[0,1]\times[0,1]\rightarrow[0,1]$
η	Parameter in Gumbel's copula that captures correlations between random variables R and M

Table 7.1: Symbols and definitions

Problem 7.3: Anomaly detection

Given

1. The model (and metamodel) from P2.
2. The time stamp of each query from a user.

determine if her/his query behavior in terms of IAT is abnormal.

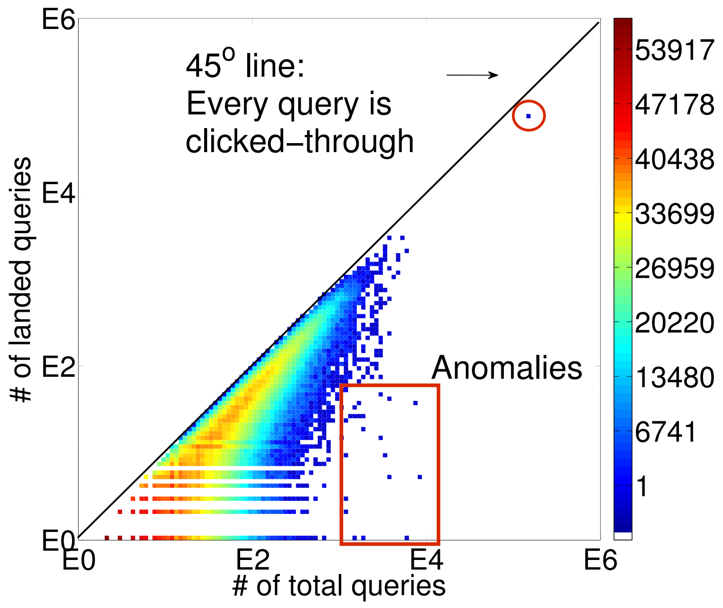


Figure 7.2: **Users with high proportion of "orphan" queries are suspicious (see the red box).** One user (circled by red) has submitted ≈ 130,000 queries, with the longest IAT of only 20 minutes (no sleep time).

7.2.2 Preliminary observations on non-landed "orphan" queries

In Figure 7.2, notice that certain users (marked by the red rectangle) have submitted more than 1,000 queries but clicked through very few (less than 100, or even zero!) of them, resulting in abnormally-many of orphan queries. Another obvious evidence is: these orphan queries usually submitted (a) consecutively and (b) with the same keyword, leading to a clear robotic behavior. Therefore, we provide the following qualitative observation.

Observation 7.1: Orphan queries

Users who have submitted many (usually more than 1,000) queries but have clicked through very few (less than 100) of them are abnormal.

Furthermore, one user (circled by red) in the upper-right corner of Figure 7.2 has submitted more queries (by two orders of magnitude, ≈ 130,000) than typical users (≈ hundreds to thousands), with the longest IAT of only 20 minutes (no sleep time). Clearly, this user is suspicious and therefore an anomaly.

After being able to detect obvious anomalies with orphan queries, we again ask the major motivating question (as mentioned in Section 7.1): *How frequently does a given user submit a* *web query and click through the search results?* Starting immediately, we ignore orphan queries and focus on the IATs of landed queries.

7.3 Single User Analysis: Camel-Log

In this section, we first detail the proposed Camel-Log distribution (Section 7.3.1), provide validations (Section 7.3.2) and give comparisons with other well-known models (Section 7.3.3). For convenience, we preview the mathematical form of Camel-Log here:

$$\begin{aligned} f_{Camel-Log}(t) &= \theta \cdot f_{\mathcal{LL}}(t; \alpha_{in}, \beta_{in}) + \\ & (1-\theta) \cdot f_{\mathcal{LL}}(t; \alpha_{off}, \beta_{off}) \end{aligned}$$

where $t \geq 0$, $f_{\mathcal{LL}}(\cdot)$ stands for the probability density function (PDF) of the log-logistic ($\mathcal{LL}$) distribution as shown in Eq(7.2).

7.3.1 Camel-Log distribution

The main idea of Camel-Log is to use a mixture of two log-logistic ($\mathcal{LL}$) distributions to model the bi-modal pattern in Figure 7.1(b). $\mathcal{LL}$ is a skewed (in linear scale), power-law-like (heavy-tail) distribution, and there are two reasons for the choice of $\mathcal{LL}$: (a) it outperforms competitors (see Section 7.3.3); (b) it has an intuitive explanation (the longer a person has waited, the longer (s)he will wait). $\mathcal{LL}$ has been used successfully for modeling the IAT of the Internet communications of humans, such as posts on web blogs and comments on Youtube[4][dMFAL13]. $\mathcal{LL}$ is defined as follows:

Definition 7.1: Log-logistic distribution

Let T be a non-negative continuous random variable and $T \sim \mathcal{LL}(t; \alpha, \beta)$. The CDF of a log-logistically distributed T is given as:

$$F_{\mathcal{LL}}(t; \alpha, \beta) = \frac{1}{1 + (t/\alpha)^{-\beta}} \tag{7.1}$$

where $\alpha > 0$ is the median (or called scale parameter), and $\beta > 0$ is the shape parameter. The support $t \in [0, \infty)$. The PDF of T is given as:

$$f_{\mathcal{LL}}(t; \alpha, \beta) = \frac{(\beta/\alpha)(t/\alpha)^{\beta-1}}{[1 + (t/\alpha)^{\beta}]^2} \tag{7.2}$$

With the knowledge of $\mathcal{LL}$, we present the definition of the proposed Camel-Log distribution:

[4]www.youtube.com

Definition 7.2: Camel-Log distribution

Let T be a non-negative random variable following Camel-Log distribution. The probability density function (PDF) can be written as:

$$\begin{aligned} f_{Camel-Log}(t) &= \theta \cdot f_{\mathcal{LL}}(t; \alpha_{in}, \beta_{in}) + \\ & \quad (1-\theta) \cdot f_{\mathcal{LL}}(t; \alpha_{off}, \beta_{off}) \end{aligned} \tag{7.3}$$

where $t \geq 0$, $\theta \in [0, 1]$, $\alpha_{in}, \beta_{in}, \alpha_{off}, \beta_{off} > 0$.

The proposed Camel-Log distribution has the following properties:

- A mixture of two $\mathcal{LL}$ (heavy-tail) distributions to qualitatively describe: in-session and take-off IAT.
- Five parameters to characterize a user's search behavior:
  - θ controls the proportion of in-session and take-off IAT.
  - α_{in} represents the median of in-session IAT.
  - β_{in} is the "concentration[5]" of in-session IAT.
  - α_{off} represents the median of take-off IAT.
  - β_{off} is the concentration of take-off IAT.

Camel-Log distribution seems to model the marginal distribution of IATs very well, at least for the users shown in Figure 7.1(b), and also provides intuitive interpretations. Next, we aim to answer the following questions:

- Is Camel-Log sufficiently general and accurate to model and interpret other users' search behavior?
- Even so, does $\mathcal{LL}$ outperform other famous "conceivable" distributions, like Exponential or Pareto (power-law)?

The answers to both questions are *yes*, and the details are provided in the following two sections.

7.3.2 Validation against empirical data

Figure 7.3 illustrates the empirical IAT from several "prolific" users who have issued many queries. Each sub-figure shows the marginal distribution of IATs (in logarithmic binning) from a user, and the red curve is depicted by fitting a Camel-Log distribution via expectation maximization (EM). For brevity, we show the fits for only 9 users, but most of the others had similar behavior (see Figure 10(a), where the vast majority of users have very similar model parameters). Notice:

- The consistency of bi-modal behaviors. All the users have the distinct "in-session" and "take-off" modes.

[5]The reciprocal of β_{in} represents (approximately) the standard deviation of $\mathcal{LL}$.

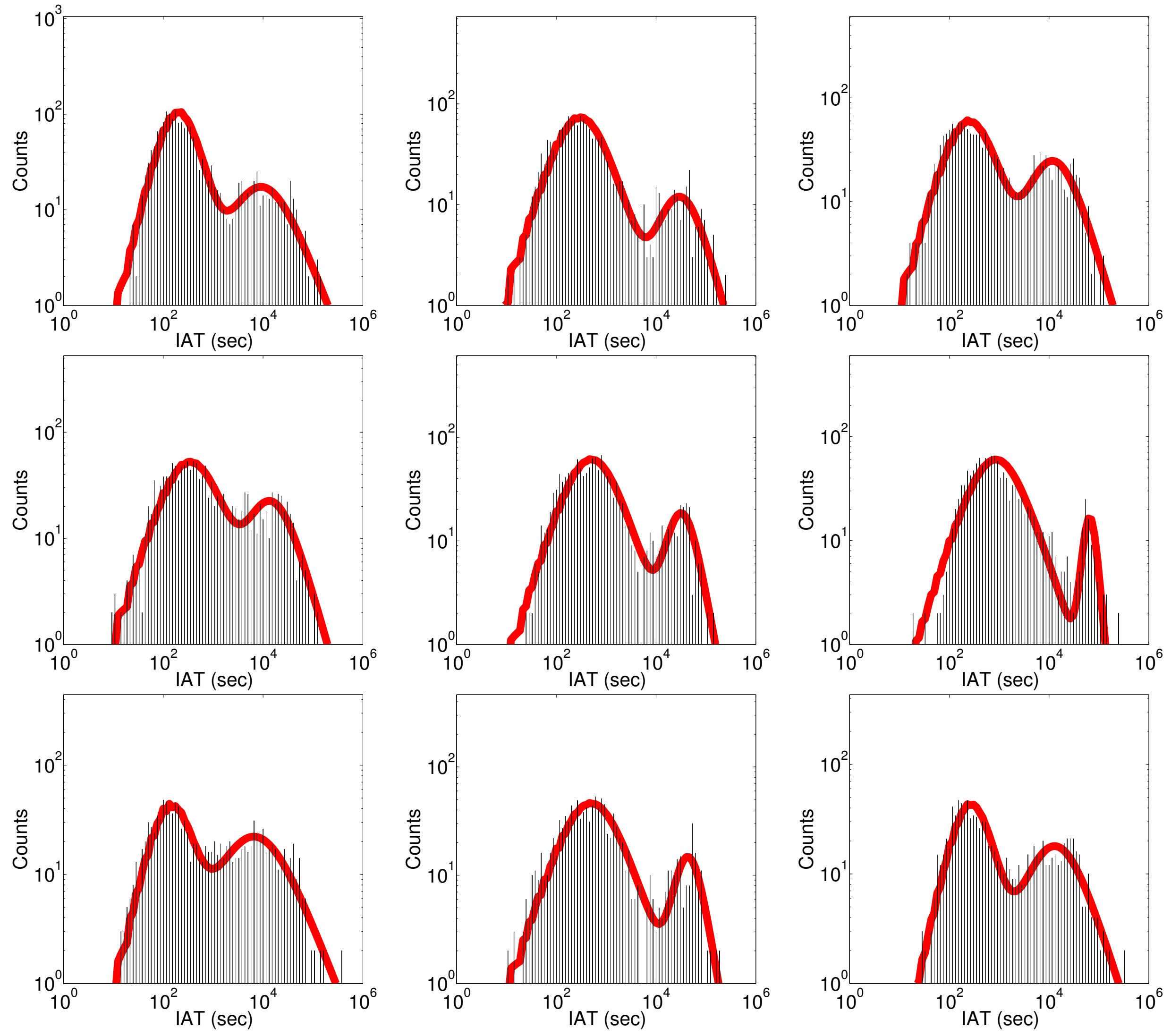


Figure 7.3: **Camel-Log captures consistently bi-modal search behaviors: in-session and take-off.** Each sub-figure shows the marginal distribution of IATs (in logarithmic binning) from a user. The red curve is the Camel-Log fit, with parameters computed via expectation maximization (EM).

- The generality of the proposed Camel-Log. Camel-Log is able to flexibly and accurately model the marginal distribution of IATs across different users. (Camel-Log also well-models other datasets – see Section 7.3.4 for details.)

Also from Figure 7.3, we provide the following observation:

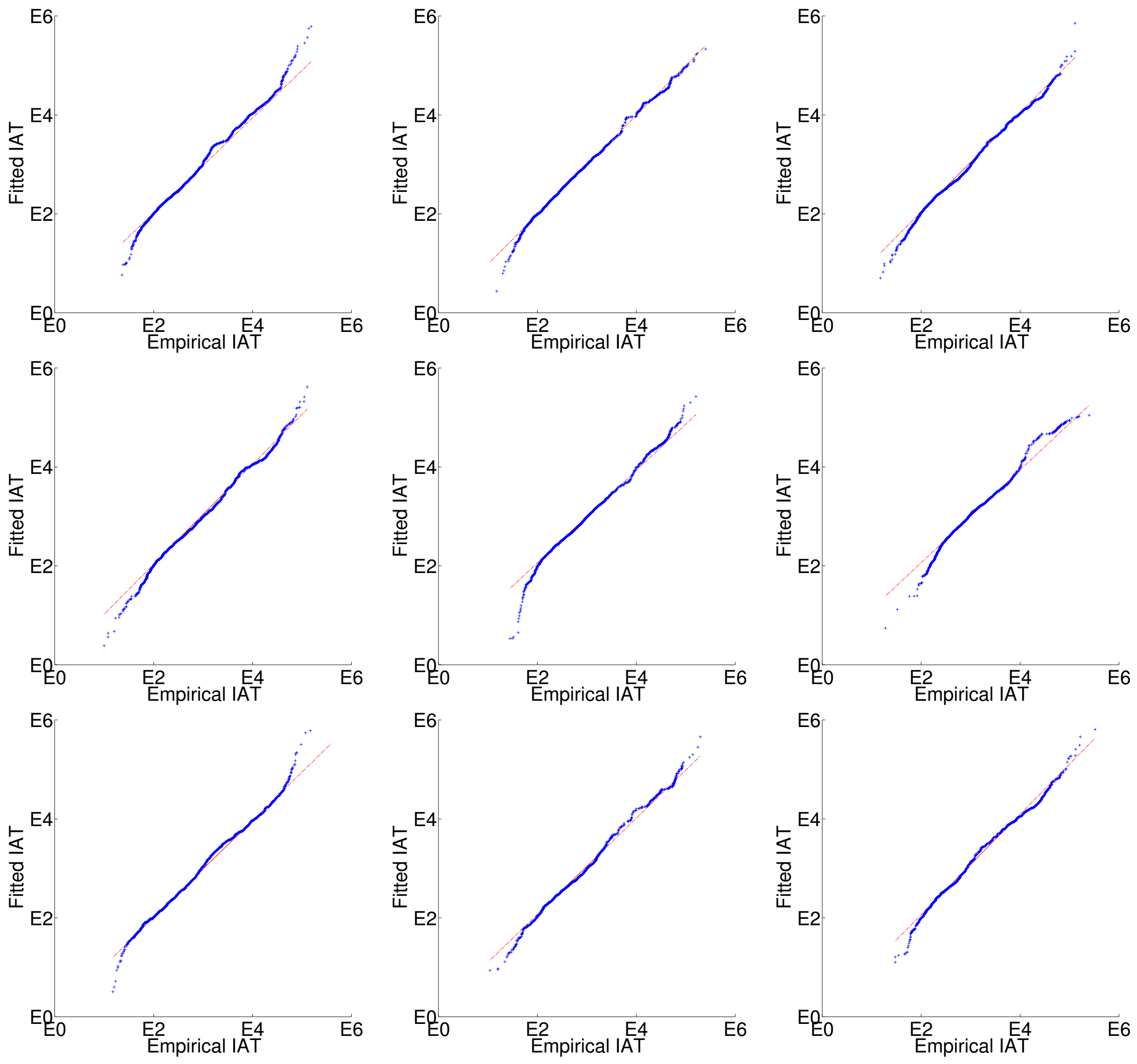


Figure 7.4: **Quantile-Quantile (QQ) plots show that Camel-Log closely fits real data.** 45° line is ideal: all quantiles of the empirical data match the corresponding quantiles of the fitted samples. In each sub-figure, the majority of quantiles are matched very well by the proposed Camel-Log distribution.

Observation 7.2: In-session and take-off

The median in-session IAT is about five minutes, whereas the median of take-off IAT is approximately seven hours.

The median of in-session IAT is about 5 minutes, which approximately represents the duration when a user is interested in the query results. On the other hand, the IAT of take-off queries is longer, ranging from tens of minutes (*e.g.*, lunch break), hours (*e.g.*, sleep time), to days (*e.g.*, weekends). The median of take-off IAT is approximately seven hours, which corresponds to sleep time very well.

More validations are provided by Figure 7.4. For each user, Figure 7.4 provides the Quantile-Quantile plot (Q-Q plot) between the empirical IAT and the samples drawn from the fitted Camel-Log distribution. In each sub-figure, X axis represents the IAT from a user and Y axis are the samples randomly drawn from the fitted Camel-Log distribution. 45° line is ideal, meaning that the empirical data and the fitted samples follow the same distribution. As evidenced by the sub-figures, the majority of quantiles are matched very well by the proposed Camel-Log distribution.

Thus far, we have shown strong evidence supporting the goodness of fit for Camel-Log, but we still need to answer the question: why not use a mixture of other well-known "named" distributions, like Exponential or Pareto (power-law)?

7.3.3 Why not other well-known distributions?

We compare the goodness of fit among the following three candidates:

- A mixture of two Exponential distributions.
- A mixture of two Pareto distributions.
- The proposed Camel-Log distribution.

by using the following criteria:

- p-value reported by two-sample Kolmogorov-Smirnov (K-S) test.
- Data log-likelihood.
- Bayesian information criterion (BIC).

It turns out that Camel-Log outperforms other candidates in all three criteria. Note that for each user, the candidate models are fitted by the training set (randomly drawn from her/his IAT), whereas the p-value and log-likelihood are reported by using the testing set (data not in the training set). Figure 7.5 provides the p-value reported by K-S test on each user, with the null hypothesis (H_0): the user's IAT follows the fitted candidate distribution. If H_0 is true, the p-value will follow a *Uniform(0,1)* distribution, depicted by the 45° straight line. From Figure 7.5, the proposed Camel-Log is the candidate closest to the true model; exponential mixture fits well but not as close, whereas Pareto mixture does not fit at all (with very low p-values). We also provide log-likelihoods to show Camel-Log better explains users' behaviors. Table 7.2 presents %-of users that Camel-Log explains better (achieves higher likelihood), compared to other candidates. The proposed Camel-Log achieves a higher log-likelihood on 78% of the users (compared to Exponential mixture), and more than 99% of the users (compared to Pareto mixture).

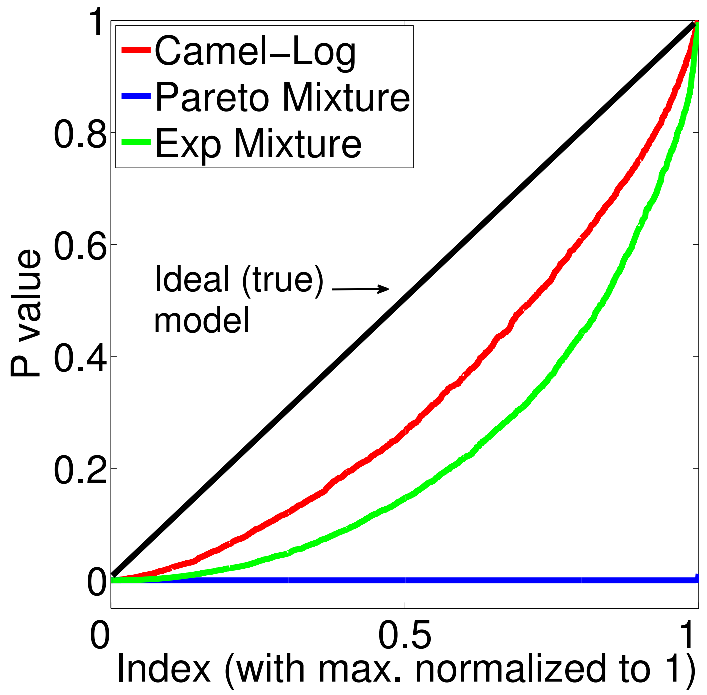


Figure 7.5: **Camel-Log outperforms competitors with respect to K-S tests**. Sorted p-values are reported from K-S tests on: the proposed Camel-Log (in red), Pareto mixture (in blue) and Exponential mixture (in green). The 45° straight line represents the ideal (true) model: p-value follows the *Uniform(0,1)* distribution. The proposed Camel-Log is the closest to the true model.

Log-likelihood (of the testing set)		
Compared against:	Exponential mix.	Pareto mix.
Camel-Log	78%	$> 99\%$

Bayesian information criterion (BIC)		
Compared against:	Exponential mix.	Pareto mix.
Camel-Log	66%	$> 99\%$

Table 7.2: Evaluation with log-likelihood and BIC: %-of users that Camel-Log explains better (higher is better)

Furthermore, since each candidate model uses different number of parameters: Camel-Log (five), Exponential mixture (three), and Pareto mixture (three), we also evaluate the BIC that strongly[6] penalizes using more parameters and therefore prefers a parsimonious model. Table 7.2 presents the BIC scores: the proposed Camel-Log achieves a lower BIC[7] on 66% of the users (compared to Exponential mixture), and more than 99% of the users (compared to Pareto mixture).

[6]Compared to Akaike information criterion (AIC).
[7]Given any two estimated models, the model with the lower value of BIC is the one to be preferred.

From the evaluation of p-value, log-likelihood and BIC among three candidate models, we summarize:

- Exponential mixture fits well, and the proposed Camel-Log fits best.
- Camel-Log still performs best even if we penalize for model complexity (BIC).
- Pareto mixture performs very poorly.

Both qualitative (Section 7.3.2) and quantitative (this section) evidence favorably supports Camel-Log's goodness-of-fit. Now we ask: how general is Camel-Log? Does Camel-Log model other Internet-based, human behaviors? The answer is *yes*: Camel-Log models the IAT between posts on Reddit[8] very well.

7.3.4 Generality of Camel-Log

Here, we evaluate the proposed Camel-Log on modeling the IAT from the Reddit dataset[9]. Figure 7.6 shows 9 typical users behaviors and the Camel-Log fits. Notice that (a) the Camel-Log fits the marginal distribution well, and (b) the consistency of the bi-modal (in-session, take-off) behaviors. Here, the median of in-session IAT is is approximately nine minutes, whereas the median of take-off IAT is around 10 hours. Recall in the Observation 7.2 (for web queries), the median of in-session IAT is about five minutes, whereas the median of take-off IAT is approximately seven hours. This makes sense, since compared to web queries, (a) each post/comment on Reddit requires few more minutes to compose (longer in-session IAT); (b) people post on Reddit less frequently (longer take-off IAT).

Figure 7.7 also shows that Camel-Log fits the Reddit dataset well by Q-Q plot. Notice that the majority of quantiles match very well. Therefore, the generality of the proposed Camel-Log is demonstrated: Camel-Log fits and explains diverse datasets (both Google queries and Reddit posts).

Since Camel-Log characterizes each user's search behavior by five parameters, we ask: how to use these parameters, specifically the ratio (R) and the log-median (M), to detect anomalies as Figure 7.1(c) shows?

7.4 Group-level analysis: Meta-Click

Are there regularities in the parameters of all the users? It turns out that yes, some of the parameters are in fact correlated. The two that show a stronger correlation are the ratio R ($\triangleq \frac{\theta}{1-\theta}$) and the log-median M ($\triangleq \log(\alpha_{IN})$). Thus, our goal is to model the joint distribution.

Jumping ahead, given that both their marginals follow $\mathcal{LL}$ (see Section 7.4.1), how should we combine them, to reach a joint distribution that models Figure 7.1(c)? The main idea is to use a powerful statistical tool, Copulas (see Section 7.4.3). For convenience, the final CDF of the

[8] http://www.reddit.com/

[9] The dataset contains 16,927 unique users; for each user, we collect the timestamp of 500 his/her most recent posts.

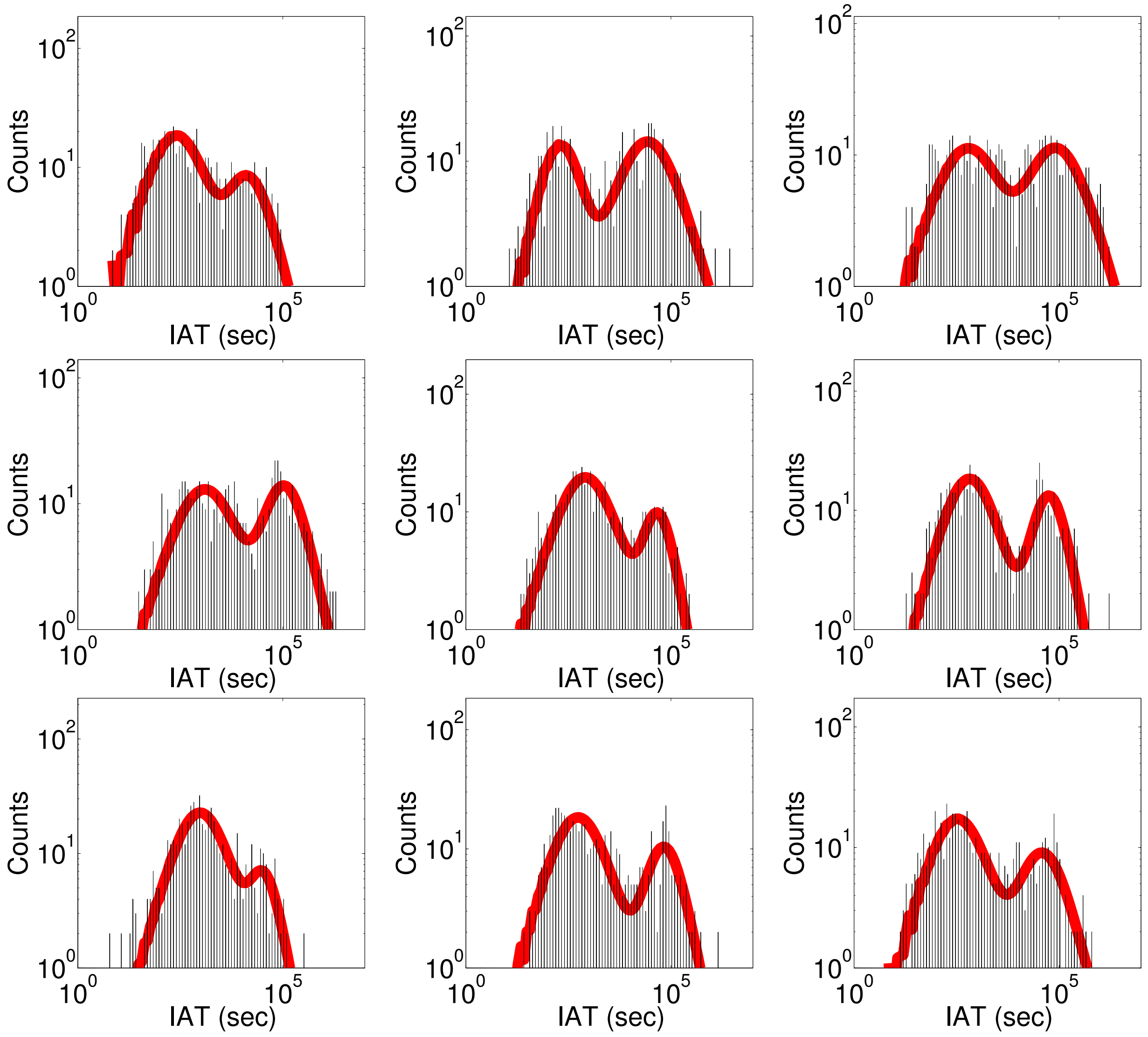


Figure 7.6: **Camel-Log fits the Reddit dataset (marginal PDF)**. Each sub-figure shows the marginal distribution of IATs and the proposed Camel-Log fitting results (in red) for the 9 most prolific users. Notice that Camel-Log fits well. Further notice the consistency of the bi-modal (in-session, take-off) behaviors.

proposed Meta-Click (details in Section 7.4.4) is provided here:

$$\begin{aligned} & F_{Meta-Click}(r, m; \eta, \alpha_R, \beta_R, \alpha_M, \beta_M) \\ = \;& e^{-([\log(1+(r/\alpha_R)^{-\beta_R})]^{\eta}+[\log(1+(m/\alpha_M)^{-\beta_M})]^{\eta})^{1/\eta}} \end{aligned}$$

7.4.1 Marginal distribution of R and M

With the parameters extracted by Camel-Log (specifically, θ and α_{in} for each user), we define two random variables that are particularly useful for anomaly detection:

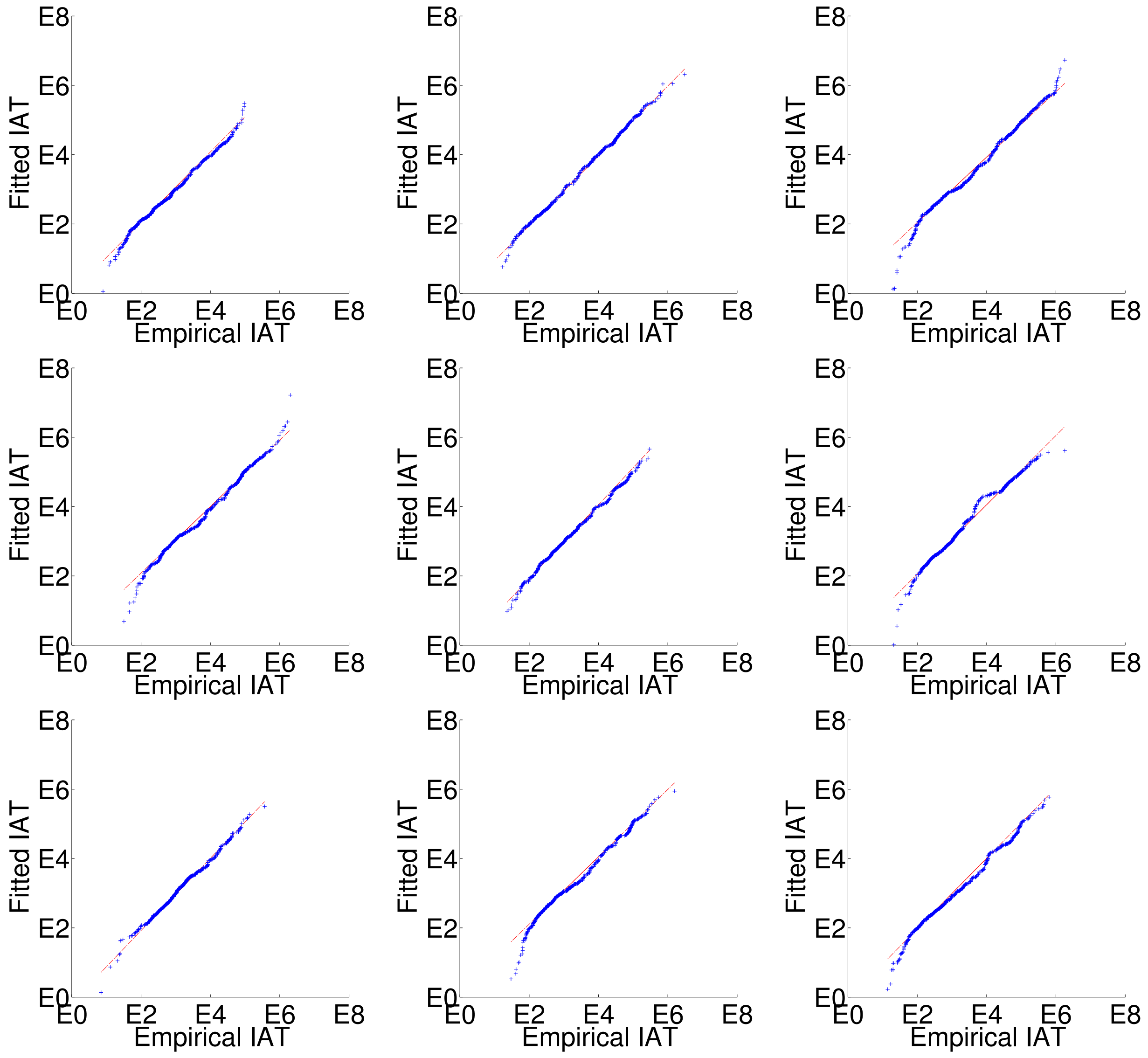
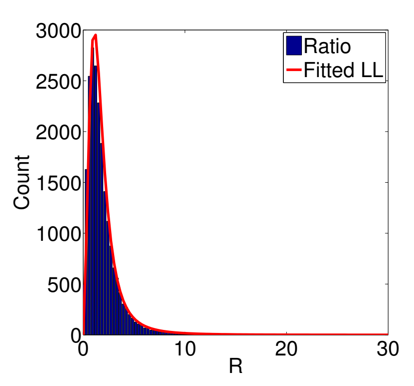


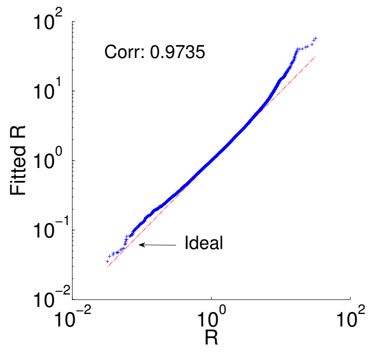
Figure 7.7: **Camel-Log fits the Reddit dataset (Q-Q plot)**. Each sub-figure shows the Q-Q plot (ideal: 45° line) between the real data (the same, 9 most prolific users in Figure 7.6) and the samples randomly drawn from the corresponding fitted Camel-Log distribution. Notice that the majority of quantiles match very well.

- *Ratio*: $R \triangleq \theta/(1-\theta)$ that represents how many "query and click"s are happening within a search session (in-session) versus take-off.
- *Log-median*: $M \triangleq \log(\alpha_{in})$ represents the median of in-session IAT in log scale.

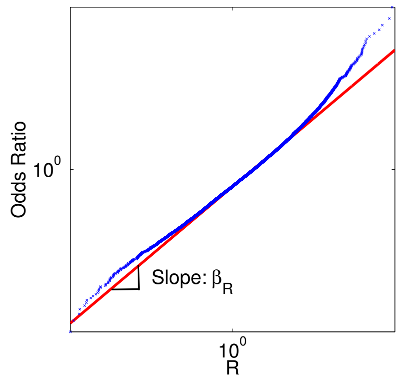
Intuitively, R and M represent an aggregate behavior, in terms of a statistical distribution of parameters (specifically, θ and α_{in}) used to characterize each user. Figure 7.8 illustrates the marginal distribution of R in (a) and M in (d), respectively. Note that all the $\mathcal{LL}$ fittings are done by using Maximum Likelihood Estimate (MLE).



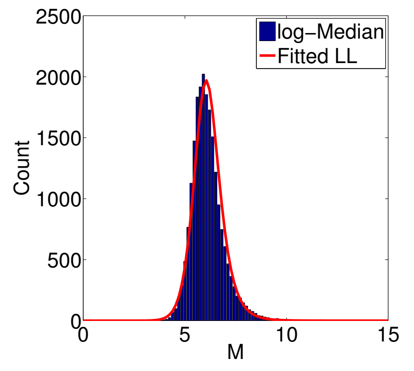
(a) Marginal distribution of R (lin. scale)



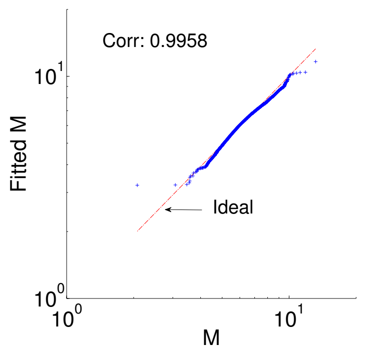
(b) Q-Q plot of R (log scale)



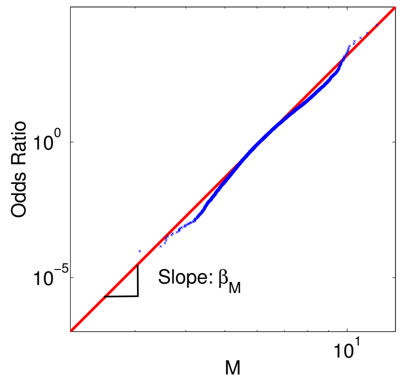
(c) Odds Ratio of R



(d) Marginal distribution of M (lin. scale)



(e) Q-Q plot of M (log scale)



(f) Odds Ratio of M

Figure 7.8: **Marginal distributions follow $\mathcal{LL}$ distributions**: (a) Marginal distribution of the ratio R and the $\mathcal{LL}$ fitting. (b) Q-Q plot between empirical R and fitted $\mathcal{LL}$. (c) Odds Ratio (OR) between empirical R and fitted $\mathcal{LL}$. (d)(e)(f) provide the corresponding plots for the median in-session IAT M. In (c), the OR of R is fitted by a line, indicating that its marginal distribution follows a $\mathcal{LL}$. The same statement also holds for (d). K-S tests for both R and M indicate that the empirical data follows the fitted $\mathcal{LL}$.

To better examine the distribution behavior both in the head and tail, we propose to use the Odds Ratio (OR) function. Figure 7.8(c)(f) show the OR of R and M, respectively. For both random variables, ORs seem to entirely follow the line. In fact, this follows from the following lemma:

Lemma 7.1: Odds Ratio

If T follows $\mathcal{LL}$, then $OR(t)$ behaves linearly with slope β and intercept $(-\beta \log \alpha)$ in logarithmic scale.

Proof. From the definition of OR function, we have:

$$OddsRatio(t) = OR(t) = \frac{F_T(t)}{1 - F_T(t)} = \left(\frac{t}{\alpha}\right)^{\beta} \tag{7.4}$$
$$\Rightarrow \log OR(t) = \beta \log(t) - \beta \log \alpha$$

■

This evidences that R and M's marginal distributions follow $\mathcal{LL}$. K-S tests for both R and M show that under 95% confidence level, we retain the null hypothesis: R (and M) follow the fitted $\mathcal{LL}$.

Observation 7.3: Common user behavior

The mode of the ratio R is approximately three, which suggests a common user behavior: "click-click-click—take off—then click (new session)."

The marginals of R and M follow $\mathcal{LL}$, but how about their two-dimensional joint distribution ($F_{R,M}$)? Can we use a multivariate normal (MVN) distribution to describe them?

7.4.2 Why not multivariate normal (MVN)?

Modeling multivariate distributions is a rather challenging task. One popular method is to use a multivariate normal (MVN) distribution. However, we provide four reasons against the use of MVN in modeling the joint distribution of R and M:

- *Marginals are not Normal.* As shown in Section 7.4.1, the marginals of R and M follow $\mathcal{LL}$, as opposed to MVN's marginals being normally distributed.
- *Contour of covariance is not an ellipsoid.* As shown in Figure 7.1(c) and later in Fig 7.9(d), the contour of R and M do not follow MVN's ellipsoid contour.
- *MVN models negative values.* The support of MVN includes negative values whereas both R and M are non-negative.
- *Low log-likelihood.* The log-likelihood of MVN is an order magnitude lower than the log-likelihood achieved by the proposed Meta-Click distribution.

We ask: is there any other candidate that models a multivariate distribution, with marginals following $\mathcal{LL}$? The short answer is *yes*: the proposed Meta-Click by using the Gumbel Copula.

7.4.3 A crash introduction to Copulas

In statistics, copulas are widely-used to model a multivariate, joint distribution considering the dependency structures between random variables (*e.g.*, R and M). The main concept of copulas is to associate univariate marginals (*e.g.*, F_R, F_M) with their full multivariate distribution. Below, we refresh the reader on the mathematical definition of a copula:

Definition 7.3: Copula

A copula $C(u, v)$ is a dependence function defined as:

$$C : [0,1] \times [0,1] \to [0,1] \tag{7.5}$$

Given two random variables R, M and their marginal CDFs F_R, F_M, a copula $C(u, v)$ generates a joint CDF $F_{R,M}(r, m)$ that captures the correlation between R and M: $F_{R,M}(r, m) = C(F_R(r), F_M(m))$.

In theory, copulas can capture any type of dependency between variables: positive, negative, or independence. The existence of such a copula is guaranteed by Sklar's Theorem[10].

One type of copula is very popular in modeling joint distribution of random variables with heavy tails: the *Gumbel copula*. The definition is as follows:

Definition 7.4: Gumbel Copula

A Gumbel Copula is defined as:

$$C(u, v) = e^{-[\phi(u)^\eta + \phi(v)^\eta]^{1/\eta}} \tag{7.6}$$

where $\eta \geq 1$ and $\phi(\cdot) = -\log(\cdot)$.

Notice that $C(u, v) = u \cdot v$ when $\eta = 1$, indicating that u, v are independent.

With this tool, we are ready to describe the proposed Meta-Click.

7.4.4 Proposed Meta-Click

The goal of Meta-Click is to model the joint distribution of R and M. As the results presented in Section 7.4.1, their marginals follow $\mathcal{LL}$. By using a Gumbel copula, we present the definition of the proposed Meta-Click here:

[10]The details of Sklar's theorem can be found in [SS11].

Definition 7.5: Meta-Click

Let R and M be non-negative random variables following Meta-Click distribution, the CDF of their joint distribution is:

$$\begin{aligned} & F_{Meta-Click}(r, m; \eta, \alpha_R, \beta_R, \alpha_M, \beta_M) \\ = \quad & e^{-([\log(1+(r/\alpha_R)^{-\beta_R})]^\eta + [\log(1+(m/\alpha_M)^{-\beta_M})]^\eta)^{1/\eta}} \end{aligned} \tag{7.7}$$

where $r, m \geq 0$, $\eta \geq 1$, (α_R, β_R), (α_M, β_M) are the hyper-parameters used in $F_{\mathcal{LL}}(r)$ and $F_{\mathcal{LL}}(m)$, respectively.

In this work, η in Eq(7.7) is estimated by Kendall tau correlation [KKPF13]; the values of (α_R, β_R), (α_M, β_M) are estimated by using MLE as mentioned in Section 7.4.1. We now show that the proposed Meta-Click distribution preserves the characteristics in the marginal distributions of each random variable:

Lemma 7.2: Marginals of Meta-Click are $\mathcal{LL}$

The marginals of Meta-Click are log-logistic distributions.

Proof. We take the limit of r to infinity:

$$\begin{aligned} & \lim_{r\to\infty} F_{Meta-Click}(r, m) \\ = \quad & F_M(m; \alpha_M, \beta_M) \\ = \quad & \frac{1}{1 + (m/\alpha_M)^{-\beta_M}} \end{aligned}$$

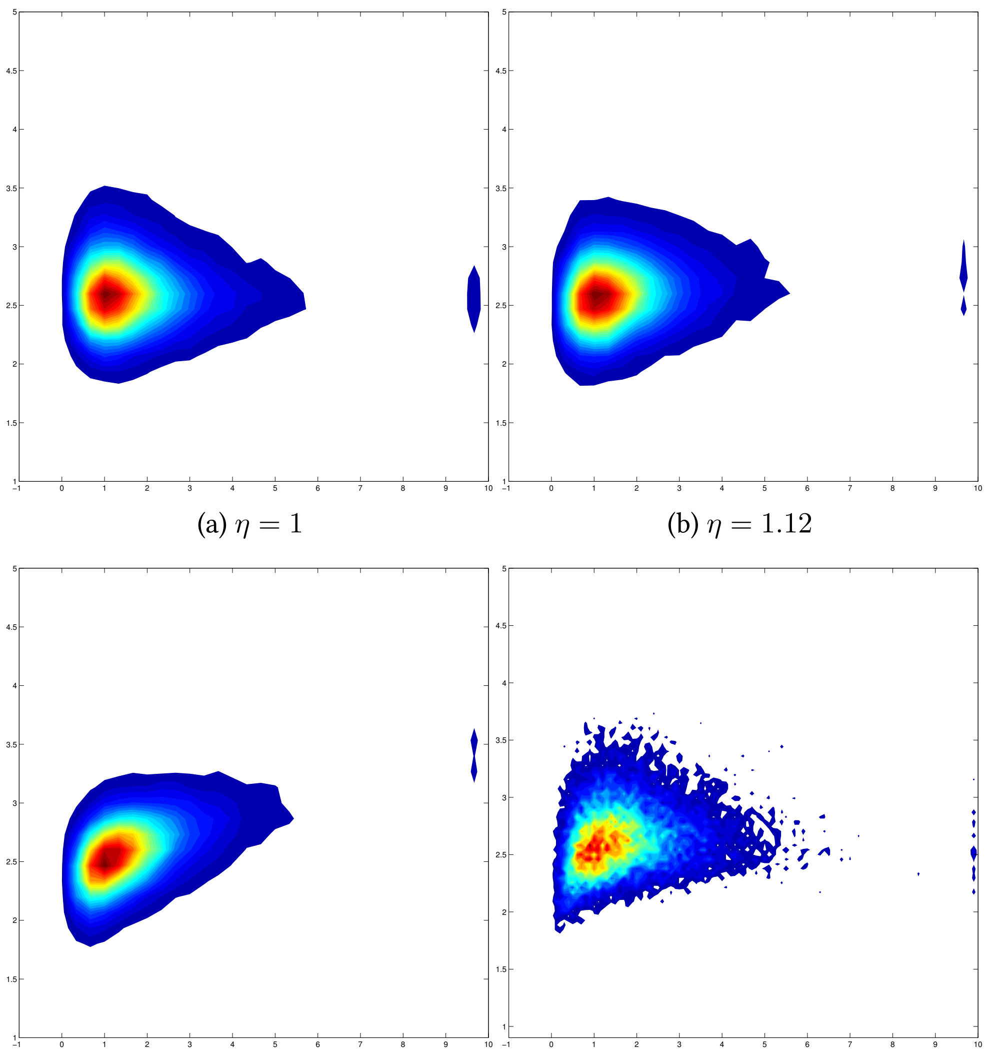
Therefore, $M \sim \mathcal{LL}\ (\alpha_M, \beta_M)$. We can show $R \sim \mathcal{LL}(\alpha_R, \beta_R)$ in a similar manner. ■

Figure 7.9(a)(b)(c) illustrate three contour plots of the proposed Meta-Click with setting η to various values, whereas Figure 7.9(d) provides the contour plot from the empirical data. The contour plot in (b) seems to match the empirical data qualitatively well.

7.5 M3A: Practitioners' Guide

We provide the step-by-step guide to apply the proposed M3A for behavioral modeling and anomaly detection:

- **Camel-Log at user level**: given a user's IAT, use Camel-Log to characterize their marginal IAT distribution with five parameters $(\theta, \alpha_{in}, \beta_{in}, \alpha_{off}, \beta_{off})$ in Eq(7.3).
- **Meta-Click at group level**: given each user's θ and α_{in} from the previous step, convert them into ratio R, log-median M and then use Meta-Click presented in Eq(7.7) to estimate the copula parameter η for the two-dimensional heavy-tail distribution.



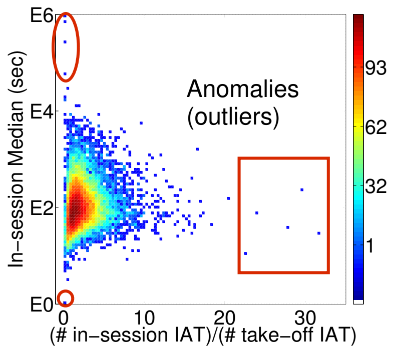
(a) $\eta = 1$ (b) $\eta = 1.12$

(c) $\eta = 1.3$ (d) Real data

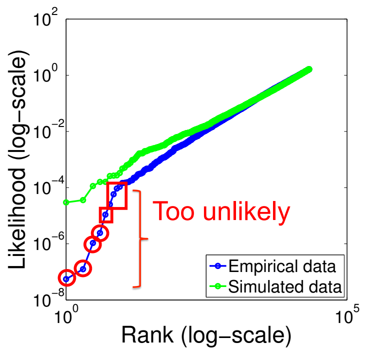
Figure 7.9: **Meta-Click matches real data**. (a)-(c): contour plots for Meta-Click (with various η). (d): real data. All plots are R v.s. M. In (b), $\eta = 1.12$, which is the value estimated from the real data. Notice how well (b) matches (d).

- **Anomaly detection**: given a user's R and M, calculate its likelihood by using Meta-Click.

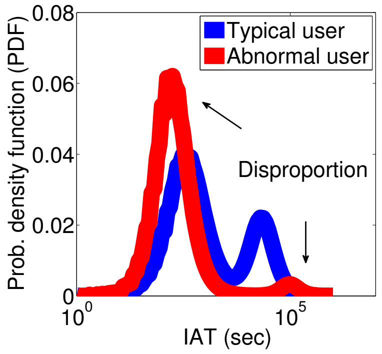
Figure 7.10 presents the anomalies detected by M3A. Figure 7.10(b) provides "rank-weirdness" plot: users are presented in a "least likely first" order, by using the likelihood of observing their R and M calculated by Meta-Click. All users fit on a line, except the first seven users who have tiny likelihoods. As a comparison, the green line shows a synthetic set of users by using Eq(7.5). Notice that none of the "green" users exhibits such tiny likelihoods; further notice that those seven users indeed correspond to outliers in (R, M) space, where we enclose them in a red box and two red ellipses for visual clarity in Figure 7.10(a). We examine the behaviors from



(a) Scatter between R and M



(b) Rank-weirdness plot



(c) Abnormally-active user

Figure 7.10: **M3A detects search anomalies**. In (a), each dot represents a user characterized by R and M extracted from the Camel-Log distribution. The anomalies spotted in (a) correspond to the few users (marked in red) with the lowest likelihoods in (b). Notice that, compared to the anomalies, the simulated samples with the corresponding ranks have much higher likelihoods (by two orders of magnitude). (c) illustrates the marginal PDF of IAT from an abnormal user detected by M3A. Notice the disproportion between in-session and take-off: about 30 queries per session, whereas typical users have 3 queries per session.

these users and find several of them constantly issuing identical queries, indicating a robotic, abnormal behavior.

Figure 7.10(c) further illustrates an abnormally-active user detected by M3A. Notice the disproportion between in-session and take-off (the ratio $R \approx 30$), which is ten times higher compared to a typical user's (around 3).

7.6 Related Work

Many prior papers have attempted to model the temporal, Internet-based activities of humans:

Internet-based, temporal data. Vaz de Melo et al. [dMFAL13, DAFL10] have proposed a self-feeding process to generate IAT following $\mathcal{LL}$ distributions for modeling the Internet-based communications of humans. Becchetti et al. [BCD+08] and Castillo et al. [CCD+08] have proposed novel graph-based algorithms for Web spam detection. Meiss et al. [MMV05] have demonstrated that client-server connections and traffic flows exhibit heavy-tailed probability distributions lacking any typical scale. Münz et al. [MLC07] have presented a flow-based anomaly detection scheme based on the K-mean clustering. Gupta et al. [GGAH14] provides a comprehensive survey on outlier detection for temporal data. Veca et al. [VMJS14] have proposed a time-based collective factorization for monitoring news. Xing et al. [XPYW11] have proposed to use local shapelets for early classification on time-series data. Ratanamahatana et al. [RLG+10] gives a high-level survey of time-series data mining tasks, with an emphasis on time series representations. Furthermore, point processes, time series and inter-arrival time analysis have attracted huge interests, with multiple textbooks (Keogh et al. [CSP+14]).

Metrics	Meiss et al. [MMV05]	Münz et al. [MLC07]	Vaz de Melo et al. [dMFAL13]	Liu et al. [LWD10]	**M3A**
Heavy tail	√		√	√	√
Bi-modal		√			√
IAT modeling			√		√
User-level & group-level modeling					√
Fits multiple datasets	√	√	√	√	√
Anomaly detection	√	√	√	√	√
Generative	√		√		√
Interpretable	√	√	√	√	√

Table 7.3: Metrics of temporal data-mining approaches: M3A possesses all desired properties

Human activities. Shie et al. [SPT13] has proposed a new algorithm (IM-Span) for mining user behavior patterns in mobile commerce environments. Saveski et al. [SG11] has adapted active learning to model the web services. Barabasi [Bar05] models and explains human dynamics with heavy-tail distributions. Liu et al. [LWD10] have provided a Weibull analysis of Web dwell time, to discover human browsing behaviors. Sarma et al. [DPS14] provides a fine tutorial on personalized search. Jiang et al. [JHA14] and Mehrotra et al. [MBY16] focus on analyzing task-based, online search behaviors, but not from the perspective of inter-arrival time.

Table 7.3 summarizes the comparison among several popular methods. As Table 7.3 shows, this is the only work focusing on the surprising pattern of web query IAT: in-session and take-off, and proposing a new framework M3A to (a) match and explain this pattern, and (b) detect anomaly. To the best of our knowledge, this is the first work to use log-logistic distributions and the Copulas (as a metamodel) to describe the IAT of web queries.

7.7 Conclusion

In this paper, we answer the motivational questions mentioned in the Introduction: 'Alice' is submitting one web search per five minutes, for three hours in a row—is it normal? How to detect abnormal search behaviors, among Alice and other users? Is there any distinct pattern in Alice's (or other users') search behavior?

We conclude this paper by bringing the answers to these questions:

- **A1: Pattern discovery and interpretation.** One key observation of IAT is provided: a bi-modal distribution with the interpretation of in-session and take-off behaviors.
- **A2: Behavioral modeling.** Specifically, we propose:

    - “Camel-Log” to parametrically characterize Alice’s (or any person’s) IAT by mixing two log-logistic distributions.
    - “Meta-Click” to describe the joint probability of two parameters of Camel-Log by using Gumbel Copula.
- **A3: Anomaly detection.** Camel-Log generates IAT with the same statistical properties as in the real data, and Meta-Click can detect abnormal users by examining their search behaviors.
- **A4: Generality**. We conduct analysis and modeling on AOL logs, and additionally generalize by showing similar characterization in Reddit posting behavior.

Finally, we provide a practitioners’ guide for M3A, and illustrate its power via “rank-weirdness” plot as in Figure 7.10(b). M3A exactly pin-points the outliers that a human expert would spot: the points in red circles/boxes, in Figure 7.10(a).

Based on content published in [Sha17].

Chapter 8

FLOCK: Astroturfing in Livestreaming Platforms

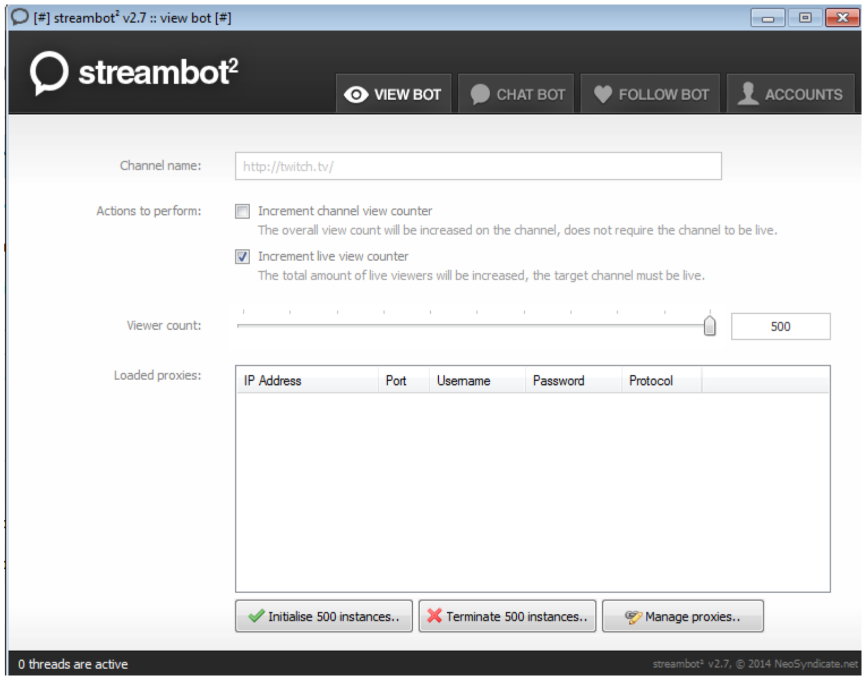
Exploring the astroturfing environment on livestreaming and detecting fake viewing behavior.

Livestreaming platforms have gained considerable popularity in recent years as a means of communicating user-generated content. They allow streamers to connect with viewers in a live fashion. This incentivizes streamers to artificially inflate their live viewership with fake viewers, called “viewbots.” In this chapter, we focus on the viewbot detection problem in livestreaming. We present the first known characterization of the problem in literature and propose FLOCK, an unsupervised approach which leverages temporal information about aggregate viewership to discern fraudulent from honest viewing behavior. Our FLOCK approach is scalable, demonstrates strong efficacy in practice and is used in production at a large, real-world livestreaming platform.

8.1 Introduction

In recent years, livestreaming platforms have risen to provide an unprecedented level of accessible and open video content to internet users. Livestreaming services such as Twitch, Youtube Live, and Ustream enable broadcasters to stream live video content of various types (often including electronic sports gameplay and other creative content) to an interested viewerbase who can both watch and interact with the broadcasters from their personal devices.

Given that livestreaming has become a popular social platform for many online communities, it has simultaneously become a target for fraud by means of *astroturfing*, or artificially inflating viewership and internet popularity. As viewership is a popular target metric for recommendation


[#] streambot² v2.7 :: view bot [#]
streambot²
VIEW BOT
CHAT BOT
FOLLOW BOT
ACCOUNTS
IP Address
Port
Username
Password
Protocol
Initialise 500 instances..
Terminate 500 instances..
Manage proxies..
streambot² v2.7, © 2014 NeoSyndicate.net

Channel name: http://twitch.tv/

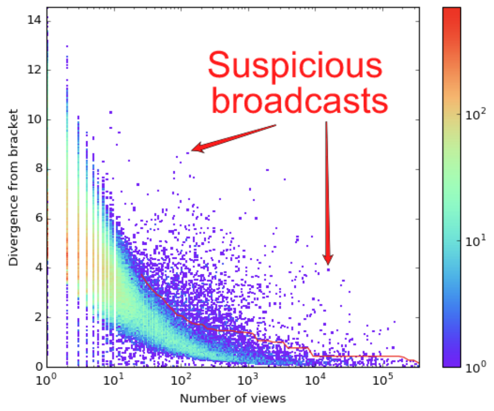
Actions to perform: [ ] Increment channel view counter
The overall view count will be increased on the channel, does not require the channel to be live.
[✓] Increment live view counter
The total amount of live viewers will be increased, the target channel must be live.

Viewer count: 500

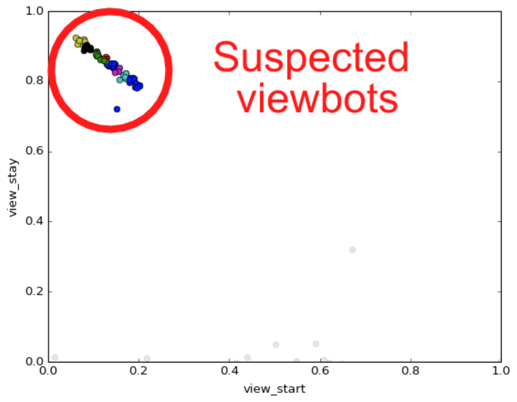
Loaded proxies:

0 threads are active

(a) Viewbotting software



(b) Viewcount-deviance plot



(c) Broadcast-behavior plot

Figure 8.1: **FLOCK finds botted broadcasts and their botted views.** (a) shows a popular *user-controlled* viewbotting tool used to manipulate broadcast viewcounts. Figure 8.1b shows our proposed FLOCK viewcount-deviance plot, which plots each broadcast's viewcount and model deviance – note the red decision boundary which isolates abnormal broadcasts. Figure 8.1c shows a FLOCK broadcast behavior plot for one such broadcast, which plots each view's relative start time and duration – notice the highly synchronized viewbot (colored, circled) behavior.

and a proxy for content quality, gaming this metric offers broadcasters numerous perceived benefits including improved recommendation rankings, directory listings, monetary partnership incentives, and hopes of a resulting larger authentic future audience. As such, it simultaneously hinders the experience for viewers who are suggested synthetically boosted content, as well as honest broadcasters who are overlooked in favor of dishonest ones. Numerous websites such as `viewbot.net` and `streambot.com` offer competitively priced *viewbots* which can be activated and deactivated on-demand to many customers (see Figure 8.1a) who wish to artificially inflate their “live” viewership.

For these very reasons, it is important for livestreaming service providers to tackle the major issue of discerning inauthentic from authentic views. This is exactly the problem we focus on in this work. While the astroturfing problem in the traditional social network (followership) setting has been explored in prior literature as an unsupervised dense-subgraph mining [SBGF14, BXG$^+$13, MWP$^+$14] or supervised classification [XFH15, Fre13] problem, the livestreaming (viewership) setting manifests numerous new challenges. Firstly, the impermanence of viewbots with primarily customer-controlled (rather than operator-controlled) parameters makes traditional dense-subgraph mining methods unsuitable – individual customers control how many viewbots they want to use at a given time as well as when they should start and stop, so attacks will generally not involve the same bots viewing the same channels at the same times. Secondly, the lack of sufficient ground-truth labels and indicative view-based features makes the classification task challenging. These differences contribute towards making the livestreaming astroturfing problem a distinct challenge from traditional settings.

In this work, to the best of our knowledge, we give the first known characterization, formulation and proposed solution to the livestream astroturfing problem. We begin by describing prior work on both livestreaming and astroturfing frontiers. We next describe the livestreaming astroturfing problem context and goals for practitioners looking to identify fraudulent behavior on livestreaming platforms both informally and formally. We further build intuition for, and propose FLOCK, an unsupervised, multi-step process for identifying viewbots in livestreaming settings which circumvents the aforementioned challenges. Our FLOCK approach works by (a) modeling broadcasts as aggregates of viewing behavior, (b) identifying anomalous (botted) broadcasts (Figure 8.1b), and (c) identifying anomalous (botted) views from within these broadcasts (Figure 8.1). We next evaluate FLOCK on a large-scale, industrial livestreaming workload and show experimental results demonstrating strong performance in practice as well as robustness against synthetic, adversarial attacks. Lastly, we discuss means for leveraging the results from FLOCK in practice and conclude. Summarily, our contributions are:

1. **Problem formulation:** We describe and formalize the problem of viewbot detection in livestreaming settings.
2. **Algorithm:** We propose the FLOCK algorithm, an unsupervised and scalable approach for finding viewbots.
3. **Practical efficacy:** We show that FLOCK achieves high precision/recall in practice and is robust to various attack patterns.

8.2 Related Work

As the astroturfing problem on livestreaming platforms has not previously been studied, we loosely categorize prior work into two main categories: livestreaming and astroturfing in social media.

Livestreaming: [VAM+02] empirically analyzes a livestreaming workload taken from internet-accessible cameras in Brazil, and models client arrivals as a Poisson process and interest profiles as a Zipf distribution for use in a synthetic workload generation toolkit. [SMZ04] conducts a larger-scale workload analysis using data from Akamai (a major content delivery network) and additionally presents results on client diversity, lifetime and stream popularity. [WLR09] studies performance of peer-to-peer (P2P) livestreaming systems using various queueing models. [LJL+06] presents the design of a scalable P2P livestreaming service built using inter-overlay optimization. [LZSJ+08, HLR07] discuss performance metrics and bounds for measuring network-quality in P2P livestreaming settings. [KSC+12] describes the prevalence of livestreaming of electronic sports (video games) and presents models to predict stream performance and popularity. [HGK14] studies the buildup and breakdown of individual and community behaviors on Twitch streams of varying popularity.

Astroturfing in social media: [CSYP12, YKGF06] propose random-walk based methods which aim to leverage abnormally sparse cuts between sybil and honest regions of undirected social networks to identify sybil nodes. [PSS+10, SBGF14, JCB+14b, JCB+14a] leverage spectral decompositions (eigendecomposition and SVD, as we discussed in Chapter 3) to catch users in social networks including Twitter and Weibo who form dense subgraphs or project abnormally to low-rank subspaces. [BXG+13, CYYP14] detail local clustering methods on Facebook page-likes and other user actions which aim to catch synchronized behavior in the form of temporally-coherent bipartite cores in graphs. [AMF10] finds anomalies in weighted graphs by demonstrating power-law patterns in features of graph egonets. [SBH+16, HSB+15] detail information theoretic and Bayesian approaches of identifying anomalous nodes and spammers in edge-attributed networks (as we will discuss in Chapter 9). [PCWF07] finds fraudulent sellers and reviewers on eBay using belief propagation when few node labels are known. [Fre13] uses a multinomial naive-Bayes classifier on n-grams of account names and e-mail addresses to find spammers on LinkedIn.

8.3 Background and Motivation

Livestreaming platforms connect *streamers*, or content broadcasters, with an audience of *viewers*. Each streamer broadcasts live video on their personal *channel* at various times for various durations, where one continuous stream is called a *broadcast*. Broadcasts are generally associated with video games, music, television, podcasts and other creative works. Viewers can enjoy live broadcasts by navigating to the streamer's channel during the times which he or she is *live*, and subsequently watching the broadcast.

While streaming content is just a hobby for most streamers, livestreaming platforms typically offer *partner* status exclusively to those who consistently stream and attain a large viewership. Partnership typically offers streamers numerous benefits including the ability to monetize from ad-revenue and paid viewer subscriptions for bonus channel content. Streamers are thus incentivized to inflate their live viewership statistics by using viewbots to both satisfy requirements for gaining partnership status and making money, as well as to improve their rankings in livestreaming directories which viewers can browse to find popular content. Sufficiently successful streamers can make living wages just from streaming as a full-time job – very popular streamers can make hundreds of thousands of dollars or more per year by means of subscriptions, donations and ad-revenue alone [Kar].

As views must be "live" when streamers are broadcasting, viewbot providers offer streamers full control with regards to (a) how many viewbots they would like to use (i.e choosing 47 from 0-100), and (b) how long the viewbots should continue to watch the stream. Upon streamer command, IPs under control of the viewbot provider emulate real viewers by sending HTTP requests for video to the livestreaming service until they are signaled to stop. Currently, a rate limit (only allowed k concurrent views per IP) is in place, meaning viewbot providers need access to $\frac{n}{k}$ IPs to successfully imitate n viewers. Given the size of the IP space from which inauthentic views can come from and transient, anonymous and streamer-controlled nature of viewbots, traditional unsupervised approaches to the astroturfing problem such as dense-subgraph detection and tensor decomposition are not suitable. Furthermore, given that ground-truth labels are difficult to obtain and most features gleanable from HTTP requests can be spoofed, supervised models for labeling individual views require constant monitoring due to their short-lived use.

Livestreaming industry practitioners are thus faced with the difficult problem of distinguishing authentic human viewers from inauthentic viewbots. Addressing this problem offers numerous benefits, most notably in preventing streamers who aim to cheat the system from becoming partnered, as well as limiting the inflation of perceived viewership and improving the authenticity of recommended popular content.

8.4 Proposed Method: FLOCK

We first give the intuition behind our FLOCK approach, and subsequently describe the individual steps in greater detail.

8.4.1 Intuition and Problem Formulation

As previously mentioned, labeling individual views as authentic or inauthentic from information contained in HTTP requests is difficult and adversarially error-prone. The intuition behind FLOCK is to take an unsupervised, offline approach which enables us to focus on viewing behaviors in *aggregate* and identify behaviors that stand out from some model of *normal* aggregate behavior. Specifically, we start by building a model of normal broadcast behavior, where broadcasts are considered aggregates of views. Next, we focus on a broadcast-level analysis instead of a view-level analysis, where we examine the aggregate behavior for all viewers who watch a given broadcast and determine whether or not the overall broadcast looks suspicious – that is, we formulate the problem of identifying botted broadcasts as an outlier detection problem. Broadcast-level analysis is useful, as it can account for multiple views simultaneously: while a single view which starts at 1pm and ends at 3pm on a given broadcast is not particularly suspicious, hundreds of such views are more suggestive of bot activity. We build from this intuition and restrict our analysis to this set of botted broadcasts and try to find groups of similar views that behave like bots by starting and stopping in lockstep. If removing such a group makes the broadcast look less suspicious in accordance with our model, then we classify those views as inauthentic.

This hierarchical paradigm of broadcast-level and subsequent view-level analysis offers interpretability and straightforward applications to both examining streamer broadcast history for viewbotting when they apply for partnership, as well as identifying IPs commonly used for viewbot traffic.

We formally construe our available data as sets of views $\mathcal{V}$ and broadcasts $\mathcal{B}$. For each view $v \in \mathcal{V}$, we have $\alpha(v)$, $\omega(v)$, and $\chi(v)$ which represent view start time, end time, and associated broadcast identifier. For each broadcast $b \in \mathcal{B}$, we have $\alpha(b), \omega(b)$, and $\rho(b)$ which represent broadcast start time, end time, and the set of constituent views $\{v \in V \parallel \chi(v) = b\}$.

With this notation, we define our problem as follows:

Problem 8.1: Viewbot identification

Given the set of views $\mathcal{V}$, broadcasts $\mathcal{B}$ and corresponding relations $\alpha(\cdot), \omega(\cdot), \chi(\cdot)$ and $\rho(\cdot)$, **find** the suspicious set of broadcasts

$$\mathcal{B}_{botted} \subseteq \mathcal{B}$$

and suspicious set of views

$$\mathcal{V}_{botted} \subseteq \bigcup_{b \in \mathcal{B}_{botted}} \rho(b)$$

This can be broken down into several steps, each with their own subproblem, which we describe the methodology for below: (a) modeling broadcast behavior, (b) identifying botted broadcasts and (c) identifying botted views. For clarity, please see Table 8.1 for reference of the recurrent symbols we use in the remainder of the section.

8.4.2 Modeling Broadcast Behavior

As a means towards the goal of identifying suspicious from normal broadcasts, we must build towards a model for what normal broadcast behavior looks like. Given the lack of ground-truth labeled data, we focus on an unsupervised model. To build such a model, we must first identify a set of relevant features. In this work, we avoid using descriptive (browser, country of origin, etc.) and engagement-based (chat activity, means of website navigation, etc.) features due to ease of manipulability and susceptibility to adversarial camouflage (engaging in actions that authentic viewers might do to appear human). Instead, we focus on modeling broadcasts by the temporal features of their constituent views.

8.4.2.1 Proposed Features

Specifically, for each view v in broadcast $\chi(v)$, we are interested in v's start and end times, $\alpha(v)$ and $\omega(v)$ respectively. Using view start time and duration is appealing, as these aspects of a view are difficult to spoof given the so-called "mission-constraints" of viewbot providers. Viewbots

Symbol	Definition
$\mathcal{V}, \mathcal{V}_{botted}$	set of views and botted views, resp.
$\mathcal{B}, \mathcal{B}_{botted}$	set of broadcasts and botted broadcasts, resp.
$\alpha(v), \alpha(b)$	view v and broadcast b's start time, resp.
$\omega(v), \omega(b)$	view v and broadcast b's end time, resp.
$\chi(v)$	broadcast which view v watched
$\rho(b)$	views that watched broadcast b
$\beta(b)$	bracket which broadcast b belongs to
$\zeta(t)$	set of broadcasts in bracket t
$\hat{b}$	probability distribution for broadcast b
$\hat{t}$	model probability distribution for bracket t
v_{start}, v_{stay}	start and stay feature representations for view v
$1_{X,Y}(v)$	indicator noting if v is in bin (X, Y)
H	discretization parameter for broadcast duration
T	discretization parameter for bracket duration
K	IQR multiplier for broadcast decision boundary
$D_{KL}(\hat{p} \parallel \hat{q})$	KL divergence of $\hat{q}$ from $\hat{p}$
$\mathcal{I}$	set of lockstep instances for a given broadcast b

Table 8.1: Frequently used symbols and definitions.

must necessarily persist for a streamer-desired duration by their very purpose – to boost the live concurrent viewers for an extended period of time.

Since a view must start during, and cannot persist longer than its corresponding broadcast $\chi(v)$ respectively, it is intuitive to consider these features as fractional values rather than raw timestamps. This is accomplished by defining the view start time and duration ("stay" time) features (v_{start} and v_{stay} respectively) as

$$v_{start} = \frac{\alpha(v) - \alpha(\chi(v))}{\omega(\chi(v)) - \alpha(\chi(v))}$$
$$v_{stay} = \frac{\omega(v) - \alpha(v)}{\omega(\chi(v)) - \alpha(\chi(v))}$$

For example, if a view begins halfway through a broadcast and lasts until three-quarters of the way through, it has a $v_{start} = 0.5$ and $v_{stay} = 0.25$. Note that this enforces the invariant $v_{start} + v_{stay} \leq 1$.

8.4.2.2 Proposed Model

Given that each broadcast b has an associated set of constituent views $\rho(b)$, we choose to model each b as a random variable drawn from a joint probability distribution reflecting the frequency of empirically observed views. To simplify representation and alleviate sparsity issues for broadcasts with small numbers of views, we discretize the otherwise continuous v_{start} and v_{stay} space over some number of H intervals each – H is tuned empirically. Thus, v_{start} and v_{stay}

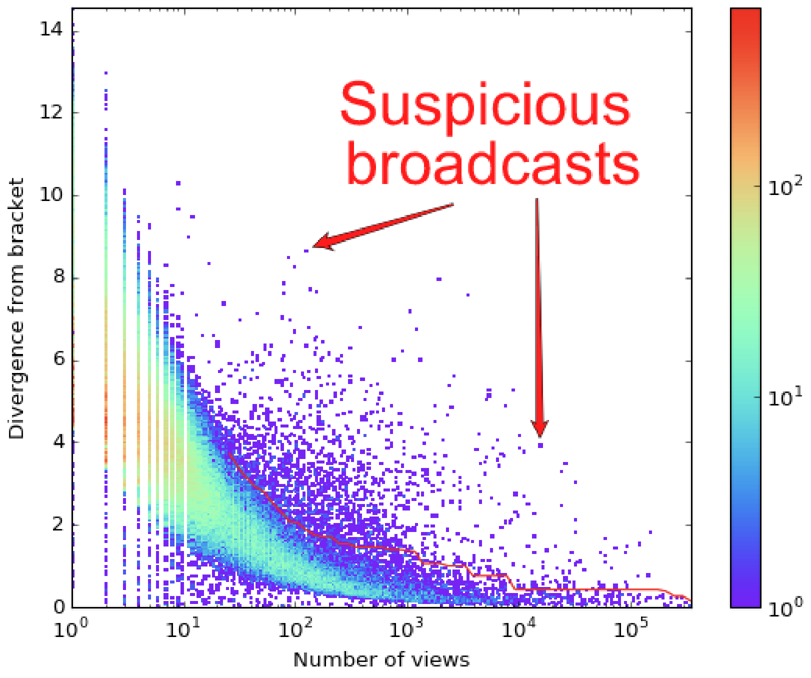


Figure 8.2: **Most broadcasts deviate from brackets predictably lowly, but outliers deviate abnormally highly**. The plot shows each broadcast as a point with viewcount on the x-axis and deviance between broadcast/bracket on the y-axis (area density denoted by color). The red line indicates our decision boundary.

can each take on values from $\{1 \dots H\}$, so the post-discretization sample space has a total of $\frac{H(H+1)}{2}$ outcomes (keeping in mind the invariant) – we will henceforth refer to v_{start} and v_{stay} as features defined on $\{1 \dots H\}$.

We can achieve this by modeling each b as a multinomial distribution $\hat{b}$ with probability masses defined by maximum-likelihood estimate (MLE) parameters. In this case, each view is treated as a realization of the associated random variable. Formally, we have

$$\hat{b}(v_{start} = X, v_{stay} = Y) = \sum_{v \in \rho(b)} \frac{1_{X,Y}(v)}{|\rho(b)|}$$

where $X, Y \in \{1 \dots H\}$ and $1_{X,Y}(v)$ is the indicator function which returns 1 if v belongs in bin (X, Y) and 0 otherwise. While we do not formally define the indicator here in interest of space, it follows naturally as a function of $\alpha(v)$, $\omega(v)$ and $\omega(b) - \alpha(b)$.

Given this approach for modeling behavior of a single broadcast, we now aim to build a model for what normal broadcast behavior looks like. In doing so, we build these models of normal behavior for broadcasts with different lengths separately, since viewers of different-length broadcasts are expected to behave differently. For example, views defined with $v_{start} = 0$ and $v_{end} = 1$ (indicating a full broadcast duration view) are likely far more common for broadcasts lasting 10 minutes than they are for broadcasts lasting 10 hours. This is due to a number of factors including time-constraints for the viewer, viewer endurance, etc. In order to group together similar-length broadcasts which we expect to be composed of similar patterns of viewer behavior, we discretize the the broadcast durations observed in our dataset into intervals spanning T minutes to alleviate sparsity issues – T is tuned empirically. We refer to these intervals as broadcast *brackets*, and introduce the functions $\beta(b)$ to denote b's bracket, and $\zeta(t)$ to denote the set of broadcasts in bracket t. Summarily, brackets are an abstraction which serve to help us separately consider and model the behaviors of different-length broadcasts.

To model the behavior for each broadcast bracket, we take a similar approach as for individual broadcasts: we treat each bracket as a multinomial distribution again using MLE parameters from the empirically observed views over all constituent broadcasts in the bracket. Formally, to compute the model distribution $\hat{t}$ for bracket t, we have

$$\hat{t}(v_{start} = X, v_{stay} = Y) = \sum_{b \in \zeta(t)} \left(\sum_{v \in \rho(b)} \frac{1_{XY}(v)}{\sum\limits_{b \in \zeta(t)} |\rho(b)|} \right)$$

Note that the usefulness of such a model in capturing normal broadcast behavior is inherently dependent on the broadcasts in our dataset. Our assumption is that most broadcasts are not viewbotted, and that there are far more authentic views than inauthentic views. Over a large number of views, we expect the bracket models will be sufficiently good estimates of authentic behavior despite some viewbotted activity. Furthermore, while individual broadcasts may have slightly different viewer behaviors due to factors such as streamer quality, directory positioning, etc., we expect that such differences will wash out over large numbers of views as well. The associated bracket distributions serve to broadly describe viewer behavior and the relative frequencies with which viewers start watching broadcasts of various lengths at different times and how long they typically watch for. Using the bracket distributions, we can answer questions such as: “how likely is a view which starts within seconds of a short broadcast to last for the whole duration?” and “how likely is the same phenomenon as broadcast duration increases?” Further discussion about the characteristics of these distributions is excluded for privacy and security reasons.

Now that we have a means for modeling both individual broadcasts and normal broadcast behavior (in different brackets), our next task is to discern which broadcasts are viewbotted.

8.4.3 Identifying Botted Broadcasts

We formulate the problem of differentiating authentic and viewbotted broadcasts as an outlier-detection task in which our interest is to find broadcasts that are unusually abnormal with respect to the associated bracket model. This intuition stems from the notion that if views in a given broadcast are distributed very differently from the views in most other similar-length broadcasts, they are likely botted as they they behave in an unusual and inhuman fashion. In order to accomplish our differentiation task, we must (a) identify a way to measure deviance between the broadcast distribution $\hat{b}$ and its associated bracket distribution $\widehat{\beta(b)}$, and (b) identify a classification threshold to set for the resulting deviance scores.

8.4.3.1 Measuring Deviance from the Model

There are a number of approaches for measuring statistical distance between two probability distributions, including variational distance, Hellinger distance, Kullback-Leibler (KL) divergence and others. Of these, we choose to use the KL divergence as our distance measure as it offers nice information theoretic properties and interpretability. The KL divergence between two distributions $\hat{p}$ and $\hat{q}$ is defined as

$$D_{KL}(\hat{p} \parallel \hat{q}) = \sum_i \left(\hat{p}(i) \cdot \log \frac{\hat{p}(i)}{\hat{q}(i)} \right)$$

for each outcome i. The divergence is defined only if $\hat{q}(i) = 0$ implies $\hat{p}(i) = 0$ in which case the summand is 0 in the limit, and is asymmetric in the sense that $D_{KL}(\hat{p} \parallel \hat{q})$ is generally not equal to $D_{KL}(\hat{q} \parallel \hat{p})$. The KL divergence of $\hat{q}$ from $\hat{p}$ (as written above), denotes the expected number of *extra* bits of information required to encode a sample from the empirically observed $\hat{p}$ distribution using a code optimized for the model $\hat{q}$ distribution rather than a code optimized for $\hat{p}$, and is thus non-negative. It can also be interpreted as the expected log likelihood ratio between $\hat{p}$ and $\hat{q}$ when $\hat{p}$ is the actual distribution of observed data (see [EC06] for further detail). Thus, the KL divergence of $\hat{q}$ from $\hat{p}$ is 0 when $\hat{p}$ and $\hat{q}$ are statistically indiscriminable, and large when $\hat{p}$ is highly unlike $\hat{q}$. In our usecase, we consider $D_{KL}(\hat{b} \parallel \hat{\beta(b)})$ to be the appropriate measure of deviance between a broadcast and its corresponding bracket distribution.

8.4.3.2 Thresholding for Classification

Given this means for measuring deviance, we next turn our attention to identifying the right threshold to set for discerning viewbotted from authentic broadcasts. While we expect that viewbotted broadcasts will have unexpectedly high deviance from the bracket distributions, choosing this threshold is important in practical implementations to limit false positives and negatives. In doing so, we must also keep in mind that the range of observed deviance scores will depend on the number of views for different broadcasts. When viewcount is small, the associated distribution $\hat{b}$ will likely not be able to express the nuances of the associated bracket distribution $\hat{\beta(b)}$ very well due to sparsity issues. Conversely, broadcasts with high viewcount should be better at approximating the associated bracket distribution because of sufficient numbers of samples. Thus, we expect high variance in the deviance scores for low viewcount broadcasts, which should decline for higher viewcount broadcasts. Figure 8.2 demonstrates this phenomenon – the heatmap shows a single point for each broadcast in our dataset, reflecting the viewcount and the corresponding deviance (KL divergence in bits) of the bracket from the broadcast (colors indicate density as per the colorbar). Note that most broadcasts with a certain viewcount commonly deviate from the associated bracket distributions in a predictably shrinking range of high-density, but several broadcasts have uncharacteristically high deviance as depicted in the sparse cloud of points violating the expected trend.

These broadcasts with abnormally high deviance are exactly those which we suspect are viewbotted. In order to automatically identify these broadcasts, we compute a moving threshold of K multiples of the inter-quartile range (IQR) above the 75th percentile (3rd quartile) of samples as the outlier "fence" or decision boundary. While $K = 1.5$ is typically used for normally distributed data [HIT86], we observe that samples in each moving bin are not normally distributed

– thus, K is tuned empirically in practice. Although a $K\sigma$ boundary above the mean could also be used to similar effect, we use the 3rd quartile and IQR here as they are less sensitive to outliers. In practice, we can also impose a threshold U on the minimum viewcount we enforce a broadcast to have before we even suspect it of being viewbotted in order to avoid penalizing broadcasts with abnormally high deviance due to sparsity issues – U is tuned empirically. These broadcasts are also far less likely to be viewbotted due to their low viewcount. Figure 1 shows a superimposed red line indicating such an outlier fence.

Equipped with a means of discerning viewbotted from authentic broadcasts, we next turn our attention to identifying botted views.

8.4.4 Identifying Botted Views

Broadcasts which have been viewbotted are likely composed of both authentic and botted views. Distinguishing between these is a crucial task for identifying abusive clients as well as gaining better estimates for the amount of authentic traffic.

In order to discern between botted and authentic views, we rely on the intuition that viewbot activity results in unusually large deviance between the broadcast and bracket distributions, whereas authentic viewing behavior is more or less distributed according to (or with small deviance from) the bracket distributions. This follows naturally from the assumption that most viewing behavior is authentic. Thus, it stands to reason that views which result in increased deviance between the broadcast and bracket distributions are prime suspects for being botted. As previously discussed in Section 8.4.1, marking individual views as botted or authentic is a highly error-prone approach – as such, labeling individual views which result in higher deviance is a risky approach. However, again we leverage the intuition that labeling an aggregate gives higher confidence than labeling the individual – essentially, if a similar group of views contributes to increased deviance, we can consider the constituent views in the group to be botted. We define a "similar" group of views as a group whose members have temporal coherence and occur in *lockstep*, or close synchrony. A number of previous works demonstrate that lockstep behavior is a strong signal for bot activities [BXG+13, CYYP14, PSS+10] (also see Chapters 3 and 6).

Leveraging this intuition, our approach for identifying botted views is to first identify instances of lockstep behavior (groups of similar views in a given broadcast), and mark these instances as botted if pruning their constituent views from the broadcast results in a smaller deviance between the pruned/altered broadcast and the associated bracket distribution. Formally, we aim to solve the following optimization problem for each outlier broadcast b:

$$\min_{b' \in 2^{\mathcal{I}}} D_{KL}(\hat{b'} \parallel \hat{\beta(b)})$$

where each of the $|\mathcal{I}|$ instances corresponds to an exclusive subset of views $\rho(b)$ such that the intersection between any two instances is the emptyset, $2^{\mathcal{I}}$ denotes the powerset of $\mathcal{I}$ (such that each element corresponds to a subset of instances), and b' and $\hat{b'}$ are the pruned broadcast and resulting empirical probability distribution formed by the views in the corresponding subset of instances. Intuitively, $\mathcal{I}$ gives a hard partitioning of $\rho(b)$ into $|\mathcal{I}|$ instances, and we are after the most seemingly authentic combination of instances, where the objective function for authenticity is the KL divergence between the pruned broadcast and associated bracket distribution. This problem has multiple associated challenges: (a) firstly, we must identify a means of partitioning the set of views $\rho(b)$ into some number of $|\mathcal{I}|$ instances, and (b) secondly, as finding the exact solution once $\mathcal{I}$ is fixed is still a combinatorial minimization task, we must find an efficient and scalable means for tackling the problem.

8.4.4.1 Partitioning Views

For the first task, we can construe instances as clusters of views in the 2D space spanned by v_{start} and v_{stay}. Clustering of similar data is a common data mining task, and we can select from a number of available clustering algorithms including hierarchical clustering, DBSCAN, K-means and more (see [Ber06] for an overview). Of these, K-means is commonly chosen due to its scalability and well-understood squared loss criterion. However, one notable caveat of K-means is the requirement of a user-specified number of clusters. Knowing this parameter a priori is difficult in our usecase and many others. Thus, we instead use an algorithm similar to X-means [PMO00], which automatically infers a suitable number of clusters by aiming to minimize the Bayesian Information Criterion (BIC), a function of log-likelihood penalized by model complexity. The approach starts with all data points in a single cluster, and iteratively splits clusters into smaller clusters if the split reduces the BIC. While the original X-means uses traditional K-means in the inner-loop, we instead use a much faster minibatch variant proposed in [Scu10] which is shown to give only marginally worse solutions than traditional K-means. This approach outputs a hard-partitioning of the views into a suitable number of instances. While the clustering approach is heuristic, an intuitively good clustering can generally be achieved over a few initializations.

8.4.4.2 Solving the Optimization Problem

The next challenge is to actually choose the subset of $\mathcal{I}$ which minimizes the above objective. As the task is to find the subset which has a minimum divergence over all the subsets, the problem is clearly combinatorial and would require checking $2^{|\mathcal{I}|}$ subsets for a broadcast partitioned into $|\mathcal{I}|$ instances. As this becomes computationally expensive quickly even for small values of $|\mathcal{I}|$ and given that our objective is non-submodular, we resort to heuristics to approximate the solution. Below, we propose several greedy heuristics for pruning the botted instances.

- **`PRUNE-TOPMOST`** – Rank each instance $\mathcal{I}_i$ in decreasing order according to the resulting reduction in deviance if it is pruned from the original broadcast b. Prune out the single instance which results in maximal reduction. If no such instance exists, stop.
- **`PRUNE-ITERATIVE`** – Rank each instance $\mathcal{I}_i$ in decreasing order according to the resulting reduction in deviance if it is pruned from the original broadcast b. Try to prune each of the instances in that order, removing those which result in reduction from the current best. A full pass over instances is considered 1 iteration. Repeat until convergence, starting with the current best broadcast from the previous iteration. If no instance is removed between the start of successive iterations, stop.
- **`PRUNE-STEPWISE`** – Rank each instance $\mathcal{I}_i$ in decreasing order according to the resulting reduction in deviance if it is pruned from the original broadcast b. Prune out the single instance which results in maximal reduction. Repeat until convergence, starting with the

current best broadcast from the previous step. If no instance is removed between the start of successive steps, stop.

Though the heuristics each offer different means of pruning botted views, they all start by considering the full set of instances $\mathcal{I}$ as the initial "best" solution, and greedily removing individual instances which result in an improvement in the objective function. Specifically, at different points in each heuristic, the test $D_{KL}(\hat{b}_{current} \parallel \hat{\beta(b)}) - D_{KL}(\hat{b}_{proposed} \parallel \hat{\beta(b)})$ is conducted in the scenario that some instance is removed from the current best broadcast $b_{current}$ resulting in the candidate broadcast $b_{proposed}$, and if the result is ≥ 0, the instance is removed from $b_{current}$ and the instances forming $b_{proposed}$ becomes the new best solution.

`PRUNE-TOPMOST` involves ranking the instances only once, and pruning the single instance which improves the objective the most. This objective is motivated by the notion that most bot behavior might be naive and viewbot attacks may typically only span one instance per broadcast. `PRUNE-ITERATIVE` and `PRUNE-STEPWISE` both involve pruning multiple instances if doing so improves the objective, but vary in the extent of greediness. `PRUNE-ITERATIVE` ranks the set of remaining instances at the start of each iteration based on the current best broadcast at the start of that iteration. However, in reality, as soon as an instance is pruned during an iteration, the true ranking for the next best candidate instances to prune may change. It is furthermore possible that the removal of an instance in one iteration may induce or preclude the removal of another instance in a future iteration. The `PRUNE-ITERATIVE` heuristic does not dynamically update the ranking upon each successful instance pruning but rather on an iteration-level, based on the notion that this iteration-level ranking may be sufficient in practice. `PRUNE-STEPWISE` conversely *does* dynamically update the ranking after each step, or successful pruning of an instance. This way, the best candidate instance for pruning is chosen at each step at the cost of more frequent ranking computations compared to `PRUNE-ITERATIVE` if multiple instances are to be pruned. These heuristics have various expected trade-offs in computational efficiency and objective minimization efficacy. We examine their practical performance in Section 8.5.

In conjunction with previous tasks, this gives us a means of identifying both viewbotted broadcasts and their botted views. See Algorithm 8.1 for pseudocode of our proposed FLOCK approach.

8.5 Experimental Results

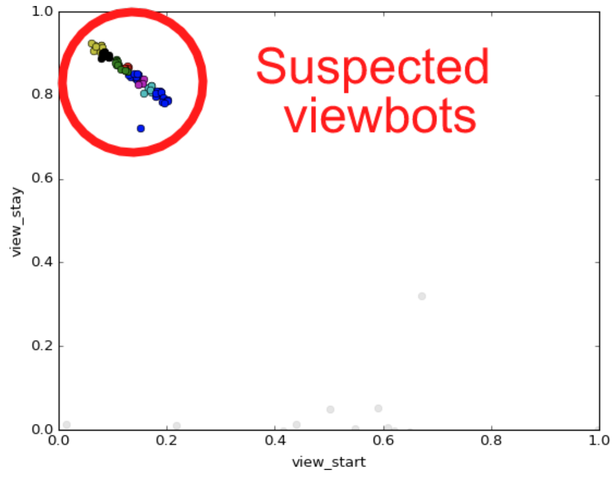
In this section, we empirically evaluate our proposed FLOCK approach's effectiveness in detecting viewbots. Firstly, we briefly describe the dataset used in the evaluation. Next, we describe several experiments in order to (a) measure success in identifying viewbotted broadcasts, (b) evaluate comparative performance of our view pruning heuristics, (c) demonstrate effectiveness in reliably discerning authentic from botted views, and (d) consider implications in the presence of a well-informed adversary. These are detailed in the below subsections.

8.5.1 Data Description

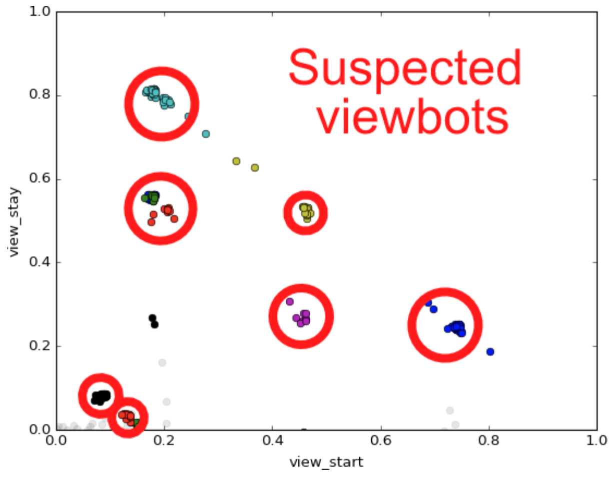
For the experiments below, we use data from a large, undisclosed livestreaming corporation spanning an 8 hour timeframe from early May 2016. Our dataset consists of 92,044 broadcasts and

Algorithm 8.1: FLOCK

```
1: **Model Broadcast Behavior**: Aggregate per-broadcast and per-bracket views and
   transform into $v_{start}, v_{stay}$ space. Build viewership models $\hat{b}$ and $\hat{t}$ for each broadcast $b$ and
   bracket $t$ according to empirically observed viewing behavior.
2: **Identify Botted Broadcasts**: Compute deviance of each broadcast's bracket distribution
   $\hat{\beta(b)}$ from the broadcast distribution $\hat{b}$, and generate viewcount-deviance plot as in Figure
   8.2. Generate decision boundary by binning viewcount logarithmically and computing a
   $Q_3 + K \cdot IQR$ fence per bin. Mark broadcasts with excessively high deviance as botted.
3: **Identify Botted Views**: For each broadcast $b$, partition $b$ into a set of instances/clusters $\mathcal{I}$.
   Attempt to solve the minimization problem $D_{KL}(\hat{b'} \parallel \hat{\beta(b)})$ where $b'$ is chosen from the
   powerset of instances $2^{\mathcal{I}}$. Use the most suitable heuristic from PRUNE-TOPMOST,
   PRUNE-ITERATIVE and PRUNE-STEPWISE.
```



(a) 76/89 broadcast views began soon after the start and persisted for the full duration. Note that the other 13 views are much sparser and shorter.



(b) 201/239 broadcast views are manifested in dense clusters indicating bots in lockstep. The teal/blue instances on the diagonal contain 50 views each.

Figure 8.3: **Viewbots creates dense clusters of activity in the $(\mathbf{v_{start}}, \mathbf{v_{stay}})$ space.** Two fraudulent broadcast behavior plots are shown with suspected botted instances and their constituent views plotted in opaque color (also circled) and unsuspected authentic views in translucent grey.

an associated 16,280,308 views. For each broadcast, we have information about the broadcasting channel and the start and end times of the broadcast. For each view, we have information about the client IP, start and end times and the destination channel. From these, we can associate views with broadcasts.

8.5.2 Broadcast Classification

The first important consideration for evaluating effectiveness is determining whether our intuition that outlier broadcasts (the sparse cloud in Figure 8.2) are in fact viewbotted is correct.

	Original D_{KL} (avg. bits)	**Pruned D_{KL} (avg. bits)**	**MAD (bits)**	**MAPD (% bits)**	**Runtime (sec)**
`BASELINE`	3.28	3.28	0	0	0
`PRUNE-TOPMOST`	3.28	2.87	0.408	12.26	1277.65
`PRUNE-ITERATIVE`	3.28	2.25	1.035	31.27	2655.51
`PRUNE-STEPWISE`	3.28	2.23	1.049	31.86	5119.04

Table 8.2: Quantitative evaluation of view pruning heuristics versus baseline (no pruning).

Unfortunately, calculating precision is difficult given the lack of ground truth. In lieu of existing labels, we resort to manual labeling. We randomly sample 100 broadcasts each from the outlier and non-outlier regions according to the decision boundary, intermix them, and manually label each broadcast using criteria based on our best judgment with respect to whether many temporally similar views come from a limited subset of IP subnets or ASNs. In practice, this often looks like 10-1000 views starting and stopping roughly at the same times, all coming from the same subnet, (i.e 46.152.x.x.)

Several such examples of suspected viewbotted broadcasts with the associated suspected views are depicted in Figure 8.3. In both cases, large fractions of broadcast views are concentrated in tightly-knit clusters which upon further inspection were from a few subnets under poorly known international ASNs. Figure 8.3a shows a rather simplistic type of viewbotting, in which bots join incrementally and quickly soon after a broadcast begins, and persist until the end of the broadcast. Conversely, Figure 8.3b shows a more complex type of viewbotting in which the streamer is tuning the bot viewership over the duration of the broadcast as he or she desires. Intuition suggests this might occur in three cases: (a) the streamer is experimenting with a viewbot provider's tool, (b) the streamer feels they have activated too many bots or too few bots initially, but massages the number over time, (c) malfunctions on the viewbot provider's side. Retroactive analysis of the live concurrent viewers counter for this broadcast indicates sharp spikes and drops (of 20 viewers at a time) as expected given the clustered behavior in the plot.

The results of our labeling experiment indicate 98% positive and 99% negative precision. That is, 98% of outlier broadcasts and 99% of non-outlier broadcasts were labeled viewbotted and non-viewbotted respectively. Recall numbers can unfortunately not be calculated due to the unbounded number of false negatives. These figures suggest that our proposed decision boundary and unsupervised classification approach is able to make a highly accurate distinction between viewbotted and non-viewbotted broadcasts. As in most threshold-based classification settings, the K (IQR multiplier used for computation of the decision boundary) can be increased to further limit false positives while catching fewer true positives, or decreased with an inverted effect if desired.

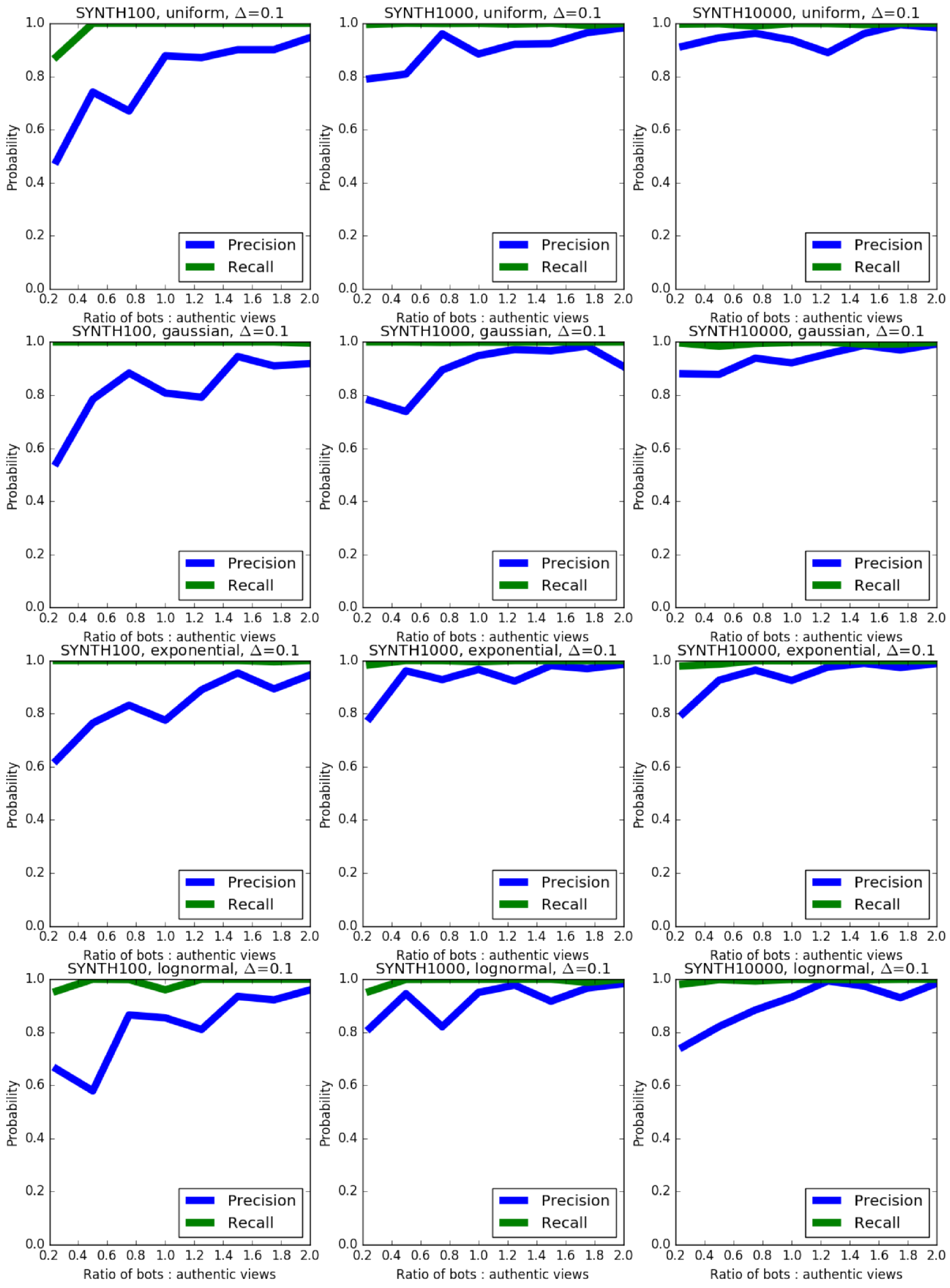


Figure 8.4: **FLOCK is accurate.** The subplots show that our approach achieves high precision/recall in discerning synthetically botted views from authentic views in a variety of settings.

8.5.3 View Pruning Heuristics

As previously discussed, the per-broadcast view pruning minimization task proposed in Section 8.4.3 is combinatorial, so we propose the use of several greedy heuristics with varying trade-offs between presumed optimization performance and scalability. In this section, we evaluate the heuristics in these terms. PRUNE−TOPMOST costs $\mathcal{O}(H^2 \cdot (|\mathcal{I}| + 1))$ for ranking the $|\mathcal{I}|$ instances and computing the KL divergence each time, and pruning a single instance. PRUNE−ITERATIVE costs $\mathcal{O}(H^2 \cdot 2|\mathcal{I}| \cdot p)$ for p pruning iterations where each iteration involves ranking and trying to prune each of the $|\mathcal{I}|$ instances (we find p is generally very small in practice, ≤ 5). PRUNE−STEPWISE costs $\mathcal{O}(H^2 \cdot (\mathcal{I} + 1) \cdot s)$ for s pruning steps, each of which involves running PRUNE−TOPMOST. Generally, PRUNE−STEPWISE is slower that PRUNE−ITERATIVE, which is slower than PRUNE−TOPMOST. However, the heuristics have inverse performance on the minimization task.

Table 8.2 gives empirical results which demonstrate these findings in comparison to a null baseline, in which no pruning is performed. For each heuristic, we prune each outlier broadcast in our dataset and measure the initial average deviance (D_{KL}, in bits) from the broadcasts and their associated bracket distributions. We next prune each of the broadcasts using the various heuristics and measure the post-pruning average deviance per broadcast, mean absolute (percentage) deviation (MAD and MAPD) and total runtime over all broadcasts (over 5 initializations). Note that here, higher is better for MAD and MAPD, indicating a larger drop in the deviance D_{KL}. Note that given D_{KL}'s interpretation as a log likelihood ratio, an average improvement of even 1 bit corresponds to doubling agreement between distributions. While minimization performance is inversely related to the computational performance of the heuristics, the benefit of PRUNE−ITERATIVE over PRUNE−TOPMOST is quite large (200%) compared to PRUNE−STEPWISE over iterative (3%). Given the much worse scaling of PRUNE−STEPWISE in situations where many instances are pruned, we choose to use PRUNE−ITERATIVE in practical implementation as it minimizes the objective almost equally well.

8.5.4 View Classification

As getting ground truth labels on a per-view basis is even more difficult to obtain than on a broadcast-level, we turn to synthetic experiments to evaluate FLOCK's performance in discerning botted views from authentic ones. To do so, we simulate botted broadcasts by varying numbers of authentic views and botted views by sampling the two types from various distributions. Specifically, we sample authentic views from an empirical bracket distribution and jitter them with Gaussian noise. Then, for each broadcast, we sample the first botted view's v_{start} and v_{stay} time uniformly over the feasible space, and subsequent botted view interarrival times (IAT) and intertermination times (ITT) from a number of distributions which viewbot providers might use to "space out" real attacks. As Figure 8.3 indicates that botted views are almost in lockstep and delivered over a short amount of time, we artificially deliver and terminate these botted views in accordance over 10% of broadcast duration respectively – we refer to this attack scale parameter as Δ. Intuitively, $\Delta = 0.1$ indicates that botted views were delivered over a 10% timeframe, and then all terminated in a different 10% timeframe. This is a realistic setting from our observations. We found results were not sensitive to Δ, indicating that delivering the bots slowly in a non tightly-knit fashion still resulted in them being caught by our approach.

Figure 8.4 summarizes precision and recall results from experiments on 96 combinations of the following 3 parameters: authentic viewcount (chosen from 100, 1000 and 10000), proportion of botted views (chosen from $0.25, 0.5 \ldots 2.0$ – functions as a multiplier on the number of authentic views), and viewbot attack distribution (IAT and ITT chosen from uniform, Gaussian, exponential and lognormal distributions). Precision and recall figures were averaged over 5 runs of each experiment. The results show consistently high (≥ 0.95) recall in almost all cases, and high (≥ 0.9) precision for broadcasts with especially high levels of botted activity. The precision tends to increase with higher numbers of botted views as the clustering process is better able to separate the botted views from authentic ones and successfully prune them. Precision also tends to be better for broadcasts with higher number of authentic views as the relative fraction of authentic views captured in an instance of mostly botted views (false positive rate) is lower. These results are especially promising for practical scenarios as in Figure 8.3 in which we observe that a large fraction of total views are botted.

8.5.5 Adversarial Implications

In previous experiments, we operated from the assumption that botted views are delivered in lockstep based on practical observations. But, what are the implications of our approach in a setting with an adversary who has complete knowledge of the bracket distributions he must emulate to appear normal? Note that this is a very strict, worst case setting – in order to have the correct bracket distribution to sample from, the adversary must a priori know (a) broadcast duration (b) internal records on viewerbase behavior. We do not expect this level of sophistication from an attacker, as this information is not exposed to the public. However, it is a useful thought experiment to consider the effectiveness of FLOCK in the face of an informed adversary who can reliably blend in with normal viewer behavior.

We conduct an experiment in which the adversary acts to minimize risk of being caught by sampling their views directly from the target bracket distribution (provably optimal, as D_{KL} between broadcast and bracket is 0 in the limit). Since adversaries are IP constrained, we aim to evaluate the IP cost implications upon adopting this strategy. Specifically, we calculate how many more IPs are required to reach the same live concurrent viewer count when using FLOCK with a rate limit, compared to a standalone rate limit as described in Section 8.3. We estimate this figure by computing the expected value of a random variable with outcomes corresponding to relative overhead in maximum number of IPs required for each approach on each bracket, and probability corresponding to the empirical frequency of broadcasts in that bracket.

Our results indicate that using FLOCK with a rate limit incurs an expected 40% overhead in IP addresses required to viewbot compared to naive rate limiting – intuitively, viewbot providers need 40% more IPs to add “noise” resembling real views rather than simply activating bots in lockstep like in Figure 8.3. Thus, we conclude that even against a knowledgeable adversary, our approach significantly increases the cost of viewbotting.

8.5.6 Scalability

FLOCK’s analysis phase begins by clustering each of the outlier broadcasts in $\mathcal{B}_{botted}$. As X-means has been shown to scale better than traditional K-means, we consider the time complexity for

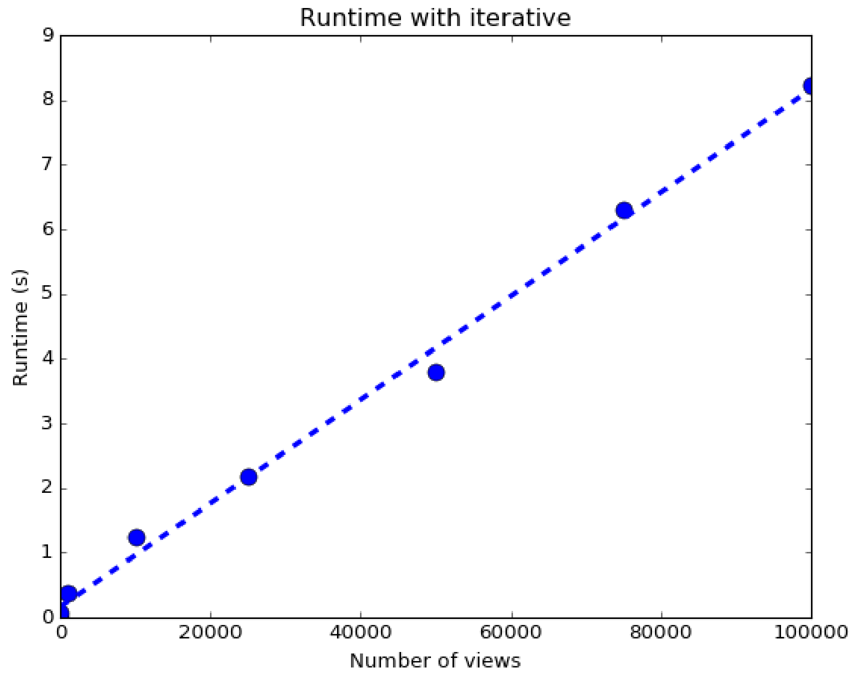


Figure 8.5: **FLOCK scales linearly in the number of views.**

clustering the views of each broadcast b to be at worst $\mathcal{O}(|\rho(b)| \cdot |\mathcal{I}| \cdot d \cdot c)$, for $|\rho(b)|$ views, $|\mathcal{I}|$ instances, d dimensions (2 in our case) and c clustering iterations. Upon clustering, we use `PRUNE-ITERATIVE` to prune suspected botted instances and views from each broadcast – see Section 8.5.3 for complexity analysis of the heuristics. Figure 8.5 shows the linear scaling of FLOCK on a number of synthetically generated broadcasts with varying viewcount.

8.6 Discussion

FLOCK is an unsupervised, offline (post hoc) approach for detecting botted broadcasts and views in livestreaming. In practice, the results can be leveraged in a number of ways. Firstly, FLOCK can be used daily on newly collected view and broadcast data. Bracket models can be composed over a longer time rather than a single day to account for changes in global viewer behavior over time due to improved QoS, usability, etc. The results of these daily runs can provide history of a streamer's broadcasts as well as history of each seen IP's previous views along with FLOCK's associated ruling of (in)authenticity. The former can be used to aid in making partnership decisions by filtering fraudulent applicants with a history of botted broadcasts. The latter can be used to better estimate true past viewership, and adjust future live viewcounts using IP-based penalties reflecting the prevalence of previously botted activity from the IP. It can also be used to identify viewbot browser, service provider and environment signatures decipherable from the received HTTP requests and proxy ground "truth" for a supervised scheme.

8.7 Conclusion

Online livestreaming has become a prevalent means for individuals to broadcast creative content to the masses. Streamers have high incentive to gain viewership in order to promote their brand, gain status and earn a living through ad revenue, donations and subscriptions. Our work is the first to focus on the problem of astroturfed, or artificially inflated viewership on livestreaming platforms. We begin by characterizing the livestreaming context and formalizing the problem setting for identifying fake views. To this end, we propose FLOCK, a principled and scalable method for identifying botted broadcasts and views in an unsupervised fashion. FLOCK works by first discerning broadcasts with highly abnormal and seemingly inhuman viewership behavior, and next tries to prune out lockstep views from these broadcasts which make the broadcast appear more genuine. Our approach achieves over *98% precision* in identifying botted broadcasts and over *90% precision/recall* in identifying views in large viewbot attacks. Furthermore, we find that even against an intelligent adversary aware capable of mimicking normal viewing behavior, FLOCK forces viewbot providers to expend *40%* more IP addresses to successfully viewbot compared to existing rate-limiting schemes.

Part III

Mining Rich Graphs

Based on content published in [SBH+16].

Chapter 9

EdgeCentric: Anomaly Ranking in Edge-attributed Graphs

Evaluating abnormality of nodes by modeling their edge-attributes with a compression paradigm.

How can we automatically identify anomalous behavior given a network with attributed edges? Networks with edge attributes are commonplace in the real world in a variety of settings, such as online product ratings, user friendships and other interaction networks. The attributes on edges in such networks capture information about how adjacent nodes interact with one another – for example, the rating that a user gives a product, or the time at which the rating is given. In this chapter, we propose the EdgeCentric approach for ranking suspiciousness of nodes in a network by leveraging edge attributes in a completely unsupervised fashion, which lends well to the scarcity of ground-truth labels in practical anomaly detection scenarios. Our approach is grounded in information theoretic principles, is scalable and demonstrates practical efficacy in identifying fraudulent and otherwise anomalous behavior in real e-commerce networks.

9.1 Introduction

Given a graph with attributed edges, what can we say about the behavior of the nodes? For example, in a user-product graph with a rating attribute (1-5 stars) on edges, how can we discern which users rate (and which products are rated) normally or abnormally? Furthermore, between two users with varying edge behavior, can we say which is more suspicious? These are exactly the questions we address in this paper – more specifically, we focus on the problem of leveraging edge-attributes in social and information graphs for anomaly detection and user behavior modeling purposes. For practitioners, learning about their data in an unsupervised fashion when ground-truth is scarce or unavailable is an important setting, particularly in fraud and anomaly detection usecases. Furthermore, answers to these questions are invaluable for routing attention to the most anomalous behaviors in given data. Informally, our problem is as follows:

Problem 9.1: (Informal) Edge-attributed Anomaly Detection

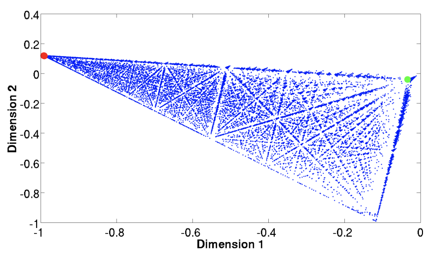
Given a static graph with possibly multiple numerical or categorical edge attributes, **identify** the nodes with most irregular (adjacent) edge behavior in a **scalable** fashion.

This problem has numerous applications – graphs with edge attributes are ubiquitous in the real-world. Typically, these attributes take the form of numerical or categorical features which describe details about the interactions between two connected nodes. For example, edges in unipartite social graphs (e.g. Facebook, Twitter) may be attributed with temporal information indicating the beginning of a friend or follower relationship. Similarly, in a who-calls-whom phone-call network (e.g. Sprint, Verizon), each caller-callee edge can be attributed with a timestamp and duration indicating when the call was made and how long it lasted. Edge attributes allow for richer representations of interactions in heterogeneous (multi-partite, multi-relational) graphs as well – for example, number-of-star ratings in user-product graphs (e.g. Amazon, Yelp) or play counts in user-media networks (e.g. Youtube, Spotify).

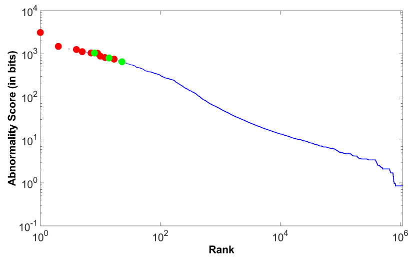
In this work, we propose EdgeCentric, an effective information theoretic approach for general node-based anomaly detection in edge-attributed graphs. Specifically, our method leverages MDL (Minimum Description Length) to rank abnormality of nodes based on patterns of edge-attribute behavior in an unsupervised fashion. Figure 9.1 shows one application of EdgeCentric on the Flipkart e-commerce network, where it is able to spot fraudulent users giving too many atypical rating values. Figure 9.1a shows a collapsed 2-dimensional subspace of users produced from the original 5-dimensional rating space (users rate products from 1-5 stars) which spectral algorithms or practitioners may examine in an effort to identify anomalous behavior. In this space, we do not find any apparent, suspicious microclusters of abnormal users. However, Figure 9.1b shows that our EdgeCentric approach successfully identifies (amongst others) highly abnormal behaviors of users who give many ratings of only 5 stars (red) or 1 stars (green). These behaviors deviate substantially from global user behavior, shown as the blue J-shape in Figure 9.1c.

The main contributions of our work are as follows:

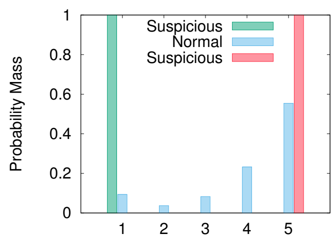
1. **Formulation:** We formalize the problem of anomaly detection on edge-attributed graphs using an information-theoretic approach.
2. **Methodology:** We develop EdgeCentric, an effective and scalable algorithm for the same.
3. **Practicality:** We experiment with our EdgeCentric on multiple large, real-world graphs and demonstrate its effectiveness and generality.



(a) Two clusters (red and green) of hard-to-discern fraudsters shown in a collapsed 2D subspace, reduced from the original 5D subspace over user rating values (1-5).



(b) Our approach, EdgeCentric, identifies the users at the red and green clusters as highly abnormal.



(c) We find that the abnormal users in the red cluster give only 5 star ratings, whereas users in the green cluster give only 1 star ratings.

Figure 9.1: **EdgeCentric spots abnormal users on real graphs.** Applied on a dataset of 3 million Flipkart user-product ratings, EdgeCentric finds users who greatly deviate from typical behavior – the red and green clusters contain single-minded "enthusiastic" and "disgusted" users who only give 5 star or 1 star reviews respectively, compared to the global (J-shape) behavior shown in blue.

Reproducibility: Our code for EdgeCentric is open-sourced at `www.cs.cmu.edu/~neilshah/code/edgecentric.tar`.

9.2 Related Work

Prior work loosely falls into three categories: (a) mining plain graphs, in which analysis is conducted using only connectivity information, (b) mining graphs with node attributes, and (c) mining graphs with edge attributes. We describe the relevant work from each category next. Table 9.1 gives a comparative analysis of existing methods, showing that none of the existing works satisfies all the relevant criteria for our problem setting.

9.2.1 Mining unattributed graphs

Akoglu et al. [AMF10] identify power-law patterns in egonets and report deviating nodes as anomalous. Tong et al. [TL11] present a non-negative residual matrix factorization method to improve graph anomaly detection in low-rank subspaces. [SBGF14, JCB$^+$14b, JCB$^+$14a] propose spectral methods to spot fraudulent behavior in low-rank subspaces of the Twitter and Weibo social network graphs, as seen in Chapter 3. [GVK$^+$12] proposes a modified PageRank measure which penalizes fraudsters based on social linking promiscuity and collusion. [PCWF07] and [ACF13] use belief propagation to spot fraudsters, on eBay, and on product-review sites, respectively. [YA15] proposes the *network footprint score* to spot opinion spammers, exploiting self-similarity and neighborhood diversity.

Dense subgraph discovery is also relevant to the anomaly detection task. Numerous methods exist for graph partitioning, including the seminal METIS algorithm [KK95] and spectral methods [WS05, PSS$^+$10] – more background on such methods is given in Chapter 5. Several

	Prior-free	Heterogeneous	Independent edge-attributes	Identifies anomalies
NFS [YA15]	✔	✔	✘	✔
FBOX [SBGF14]	✔	✔	✘	✔
CATCHSYNC [JCB+14a]	✔	✘	✘	✔
COPYCATCH [BXG+13]	✔	✔	?	✔
FRAUDEAGLE [ACF13]	✘	✔	?	✔
NETPROBE [PCWF07]	✘	✔	✘	✔
ODDBALL [AMF10]	✔	✔	✘	✔
CROSSSPOT [JBC+15]	✔	✔	?	✔
BIRDNEST [HSB+15]	✘	✔	?	✔
MIMAG [BGHS12], RMICS [BGHS13]	✔	✘	✔	✘
METIS [KK95], GRACLUS [DGK05], EIGENSPOKES [PSS+10]	✔	✘	✘	✘
EDGECENTRIC	✔	✔	✔	✔

Table 9.1: Feature-based comparison of EDGECENTRIC with alternative approaches (? indicates limited support).

information-theoretic approaches additionally automate choosing the number of partitions, including automatic cross-associations [CPMF04] and VoG [KKVF14b] for static and TIMECRUNCH [SKZ+15] for dynamic graphs (introduced in Chapter 6).

9.2.2 Mining graphs with node attributes

[GLF+10] unifies structural and attribute similarity and infers community and outliers using hidden Markov random fields. [PAIM14] introduces a "focused" clustering approach which identifies clusters and cluster outliers given a set of seed nodes from which user interests are learned. [NC03] proposes an MDL formulation for identifying common graph substructures.

[HZZ+02] proposes a variant of hierarchical linkage clustering for coupled analysis of attributed gene expression and biological networks. [LZWY06] uses spectral clustering to group various types of homogeneous node-attributed relational data. [GFB+10] introduces a pruning-based algorithm to identify subspace clusters which also exhibit strong graph connectivity. [ATMF12] proposes an MDL formulation for jointly reordering connectivity and feature matrices to identify attributed clusters. [ZCY09] proposes the use of a unified distance measure to weight the contributions of graph structure and node attribute similarity in clustering tasks.

9.2.3 Mining graphs with edge attributes

Significant work has been done on the topic of mining general edge-attributed graphs. In some cases, an edge attribute is construed as a weight, which can be used by some cut-based [AWO10] and spectral clustering [LZWY06] approaches. In our problem setting, we rather consider each edge as an interaction and each attribute as a *feature* describing the interaction, which can be categorical or numerical and need not conform to the notion of edge weight.

A considerable number of works which fall into this category have stemmed from the analysis of heterogeneous information networks (HINs), which are networks with multiple node types and edge relations. [SHZ$^+$09] was one of the first works in this area, which integrates ranking and clustering of objects in HINs using a mixed-membership model, by which nodes have unique ranks within each cluster. [SHY$^+$11] further formalizes the definition of a HIN, and also introduces the notable meta-path concept for capturing inter-entity relationships (for example, *author-cites-paper-cited by-author*) in such networks. The authors propose a function for computing inter-node similarity by the prevalence of such meta-paths between the nodes. [YCSH12] proposes an influence propagation algorithm for predicting links between objects in HINs. [SHAC12] extends the typical link-prediction problem by adding a temporality factor, and aiming to forecast when a certain edge will form based on statistical models. Many of these works more generally consider the problem of mining graphs which have multiple object types and relations, but do not necessarily focus on the edge-attributes, as this work does. [SH12, SLZ$^+$17] cover a greater scope of research in HIN mining.

The recommendation systems community has also focused on learning models of graphs with ratings [KBV09], and in some cases these models have been used to find outliers [BMFS14]. A related line of research involves mining online reviews, which can be considered as textual edge attributes [HL04]. Further work has focused on linguistic indicators of fraud in online reviews [JL08]. [DAFL10] introduces a log-logistic model for call duration in phone-call networks. [BXG$^+$13] uses local graph search on the Facebook user-likes-page graph with temporal edge features to find fraudulent subgraphs within fixed timespans. [BGHS12, BGHS13] propose methods for mining dense subgraphs with similar attribute subspaces. [JBC$^+$15] formulates a related metric of suspiciousness but is based on Poisson distributions and thus limited to simple count data. [HSB$^+$15] also approaches the problem of modeling the distribution of ratings and interarrival times from a Bayesian perspective. Our work differs in that it takes a frequentist approach based on MDL and is designed to handle any set of edge-attributes on complex heterogeneous graphs.

Overall, none of the existing works matches all the desirable features in Table 9.1. Our proposed EdgeCentric approach (a) is unsupervised, and needs no priors or existing node labels, (b) extends naturally to heterogeneous networks with multiple object and relation types, (c) supports multiple, independent edge-attributes and (d) can identify and rank anomalies.

9.3 Problem Formulation

In this section, we outline the first core contribution of our work: specifically, we formalize the problem of detecting anomalous nodes using edge attributes by leveraging a compression paradigm based on MDL. For clarity, see Table 9.2 for an overview of the recurrent symbols used in future discourse.

9.3.1 Preliminaries

The Minimum Description Length (MDL) principle states that given a family of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ for some observed data $\mathcal{D}$ is the one which minimizes the sum $L(M) + L(\mathcal{D}|M)$, where $L(M)$ is the description length in bits used to describe the model M, and $L(\mathcal{D}|M)$ is the description length in bits used to describe the data $\mathcal{D}$ encoded using the given model M. MDL enforces lossless encoding to fairly evaluate various models. In this paper, rather than using MDL to *find the best model* for our given data, we instead use it to answer the question of *how well the data fits* a given model. The intuition behind our approach is that data which fits the model well enjoys high compression, while data which is ill-represented is more costly to encode.

In our problem setting, we are given a static directed or undirected multigraph $G(\mathcal{V}, \mathcal{E}, m)$ in which nodes are connected by (possibly multiple) edges (see Chapter 2 for details). As a refresher, technically, $m : \mathcal{E} \rightarrow \{\{u, v\} \mid u, v \in \mathcal{V}\}$ assigns each edge $e \in \mathcal{E}$ to a pair of nodes. Furthermore, we have object type and relation/edge type mapping functions $\Phi : \mathcal{V} \rightarrow \mathcal{B}$ and $\Psi : \mathcal{E} \rightarrow \mathcal{R}$, where each node $v \in \mathcal{V}$ is characterized by an object type $\Phi(v) \in \mathcal{B}$ and edge $e \in \mathcal{E}$ is characterized by a relation type $\Psi(e) \in \mathcal{R}$. Here, we define an object type to reflect a node "role," – for example, a user or product. A relation type reflects the relationship between two objects – for example, user-rates-product. R. When $|\mathcal{B}| = 1$ and $|\mathcal{R}| = 1$, the graph is *homogeneous*; otherwise, it is *heterogeneous.* Furthermore, edges of each relation $r \in \mathcal{R}$ are labeled with values corresponding to the same finite subset of numerical or categorical attributes chosen from attribute set $\mathcal{A}$, given by the mapping $\Omega : \mathcal{R} \rightarrow 2^{\mathcal{A}}$, where $2^{\mathcal{A}}$ denotes the power set of $\mathcal{A}$. In other words, the graphs we consider can have numerous relation types, and edges of each relation type are characterized by a fixed number of the same attributes (at least 1). In the remainder of the problem formulation, let us consider a simple, undirected user-product graph, in which $|\mathcal{B}| = 2$ (user objects, product objects) and $|\mathcal{R}| = 1$ (user-rates-product, or product-rated-by-user relation) for ease of explanation. Let us also assume that we have only one attribute on the edges: say, rating of the product in terms of number of stars (1-5).

Then, our problem definition is as follows:

Problem 9.2: Edge-attributed Abnormality Ranking

Given a static multigraph $G(\mathcal{V}, \mathcal{E}, m)$ with ≥ 1 numerical or categorical edge attributes chosen from $\mathcal{A}$, **devise** an abnormality function $\delta(\cdot)$ to score each node $v \in \mathcal{V}$ based on its (adjacent) edge attribute behavior, and **identify** the most irregular nodes in a **scalable** fashion.

Symbol	**Definition**
G	static input graph
$\mathcal{V}, \lvert\mathcal{V}\rvert$	node-set, # of nodes of G resp.
$\mathcal{E}, \lvert\mathcal{E}\rvert$	edge-set, # of edges of G resp.
$m(\cdot)$	function to realize the multi-graph
$\mathcal{A}, \lvert\mathcal{A}\rvert$	attribute-set, # of total attributes across edges in $\mathcal{E}$ resp.
$\mathcal{B}, \mathcal{R}$	set of object types and relation types resp.
$\Psi(\cdot)$	maps nodes in $\mathcal{V}$ to object types in $\mathcal{B}$
$\Phi(\cdot)$	maps edges in $\mathcal{E}$ to relation types in $\mathcal{R}$
$\Omega(\cdot)$	maps relation types in $\mathcal{R}$ to attribute sets in $2^{\mathcal{A}}$
$\delta(\cdot)$	unified abnormality function, defined on nodes in $\mathcal{V}$
U, P	user, product resp.
$C, C(i)$	global (model) dist. C, prob. mass of ith element resp.
$C_{u,r,w,j}, C_{p,r,w,k}$	jth (kth) U (P) model dist. on attr. w and rel. r resp.
$\rho_{u,r,w,j}, \rho_{p,r,w,k}$	jth (kth) U (P) cluster prop. on attr. w and rel. r resp.
$\hat{U}, \hat{P}$	discrete prob. dist. (of ratings) for U and P resp.
$f_{u,r}, f_{p,r}$	rating vectors for U and P on relation r resp.
$H(\cdot)$	Shannon entropy in bits, defined on discrete prob. dist.
$KL(\cdot \parallel \cdot)$	KL divergence in bits, defined on two discrete prob. dists.
M	data model M
$L(U, M)$	# of bits used to encode M and U's behavior given M
$L(M)$	# of bits to encode M

Table 9.2: Frequently used symbols and definitions

9.3.2 Intuition

In order to see how we can leverage an information theoretic perspective and use MDL to inspire the formulation of δ, we must first consider our model and data representations. In this regard, to encode each user node, we must store information about the user's associated interactions through edges. In our running example, because each edge simply contains information about a single categorical attribute value (1-5), we must encode the attribute value to losslessly reconstruct the vector which describes the user's rating behavior. Thus, for each user node, we will encode a vector of rating values, e.g. $[5, 5, 1, 2, 5, 3, \ldots]$. Likewise, to encode a product node, we store information about the product's associated interactions through edges. Thus, we store the ratings that the product was given by various users: say, $[1, 2, 1, 3, 2, 2, \ldots]$.

To encode these individual user and product rating vectors, we first build a general model of rating behavior over *all* users and products, respectively. Note that this can be construed as an elementary user/product behavior model (we will relax the assumptions for a single model of behavior later in the section). For example, presume that the general pattern of rating behavior over all users follows the distribution $[0.15, 0.1, 0.05, 0.3, 0.4]$ (total proportions of 1s, 2s, 3s, 4s and 5s, respectively). Then, we can describe this model distribution C as a general trend that we expect a given user U's edge-attribute (rating) value vector f_u to obey, and describe the vector of

$|f_u|$ rating values with respect to this model in our formulation. In doing so, our total encoding length in bits for each user U with distribution $\hat{U}$ is as follows:

$$L(U, M) = L(M) \ + \ L(U|M)$$

where

$$L(U|M) = |f_u| \cdot \big(H(\hat{U}) + KL(\hat{U} \parallel C)\big)$$

A similar cost could be written for each product. Following the earlier description of MDL, $L(M)$ is the cost to encode the overall model (in our case C), and $L(U|M)$ is the cost to encode a fixed user's data given the model. Here the cost for encoding the user based on the model includes the Shannon entropy H and the Kullback-Leibler divergence KL. While the Shannon entropy reflects the inherent information content of the distribution $\hat{U}$, the KL divergence captures the extra information content between the user distribution $\hat{U}$ and the model distribution C:

$$KL(U \parallel C) = \sum_i U(i) \log_2 \frac{U(i)}{C(i)}$$

Here both U and C are distributions over a discrete set of outcomes, and $U(i)$ and $C(i)$ denote the probability mass associated with outcome i. Given that $H(\hat{U}) + KL(\hat{U} \parallel C)$ describes the cost of encoding a single sample from the user distribution $\hat{U}$ according to the model distribution C, we multiply by $|f_u|$ to denote the cost of $|f_u|$ total samples from the user rating vector f_u.

Although the general construction described above is required for fully encoding and reconstructing the rating vector given by a single user U (product P) according to MDL, our goal is to be able evaluate and compare the abnormality δ of two users (products) U_1 and U_2 according to our data model, rather than evaluate the model itself. In this regard, the last components of the above description, $|f_{u_1}| \cdot KL(\hat{U_1} \parallel C)$ and $|f_{u_2}| \cdot KL(\hat{U_2} \parallel C)$, are especially useful. Intuitively, these terms measure the total number of *extra* bits required to encode the attribute behavior of users U_1 and U_2 using a code optimized for the global model distribution C respectively. Note that our interest in abnormality comparison neither necessitates the use of the model cost $L(M)$, nor the entropy term. This is because the former is a fixed constant, and the latter is a cost associated with inherent information content rather the data model. As a result, by excluding these terms, we are not measuring the *total* information content for a node, but rather the more desirable information content *with respect to the model.* Though other distributional distance measures such as the KS statistic or variational distance could also be used to similar effect, we use KL divergence due to its principled information theoretic properties and interpretability. Hence, we define our initial formulation δ_{base} as follows:

Definition 9.1: Base Score

Given a single edge-attribute with model distribution C, the base abnormality scoring function δ_{base} for node $v \in \mathcal{V}$ is defined as

$$\delta_{base}(v) = |f_v| \cdot KL(\hat{v} \parallel C)$$

where $|f_v|$ gives the cardinality of the edge-attribute value vector f_v produced from v's neighboring (outgoing) edges, $\hat{v}$ gives the discrete probability distribution associated with node v over the chosen attribute and C gives the global discrete probability distribution of the chosen attribute over all edges.

This formulation admits two especially desirable properties:

Observation 9.1: Scale Sensitivity

Given two users U_1 and U_2 where $KL(\hat{U}_1 \parallel C) = KL(\hat{U}_2 \parallel C)$ and $KL(\hat{U}_2 \parallel C) > 0$, if $|f_{u_1}| > |f_{u_2}| > 0$, then $\delta_{base}(U_1) > \delta_{base}(U_2)$.

Observation 9.1 formalizes the intuition that given equal distributional deviation from the model, the user with more actions is more surprising.

Observation 9.2: Contrast Sensitivity

Given two users U_1 and U_2 such that $KL(\hat{U}_1 \parallel C) > KL(\hat{U}_2 \parallel C)) > 0$ and $|f_{u_1}| = |f_{u_2}|$ and $|f_{u_2}| > 0$, then $\delta_{base}(U_1) > \delta_{base}(U_2)$.

Observation 9.2 formalizes the intuition that given an equal number of ratings, the user whose distribution is more unlike the model is more surprising.

Note that Definition 9.1 gives a base formulation δ_{base}, for the elementary case in which we have a relation with a single, global model distribution C for just a single edge-attribute. We next relax these assumptions and discuss how to extend this formulation to more complex scenarios. We first discuss extensions to scoring a multifaceted model in which we consider multiple model distributions for a single attribute, and next broach the topic of building a joint scoring function which can additionally incorporate multiple attributes. Finally, we touch upon expanding these definitions to a unified scoring scheme which can handle more complex, heterogeneous graph structures with multiple relation types. Our end goal is to devise a formulation of δ which accounts for all of these factors in ranking abnormality.

9.3.3 Handling multifaceted edge behavior

It is often the case that patterns in user behavior are more granular than singular, global trends. For example, different users may rate products in different ways. Given that some users will be less critical and more easily satisfied than others, we may expect that some fraction of users give generally positive ratings (4s and 5s) but very few negative or neutral ratings. Conversely, some users will be very difficult to please and will give mostly negative and neutral ratings. One can consider that many such latent user (generally, node) behaviors may exist as a result of distinct preferences, response bias and a number of other factors. For example, some fraudsters may slander a product by giving it all 1-star ratings, whereas other fraudsters may bolster the ratings of a product by giving it all 5-star ratings. It can be useful to model these behaviors separately, as the single global distribution may actually be a mixture of behaviors of varying prevalence that nodes exhibit. As a motivating example, consider a global distribution of 50% ratings as 1-star and 50% as 5-star. This can actually be comprised of 3 latent user behaviors: only 1-star raters, only 5-star raters, and a small fraction of combined 1 and 5-star raters. The base model will penalize the latter group the least as they perfectly match the global distribution, but in reality this might be the most abnormal (rarest) group of users.

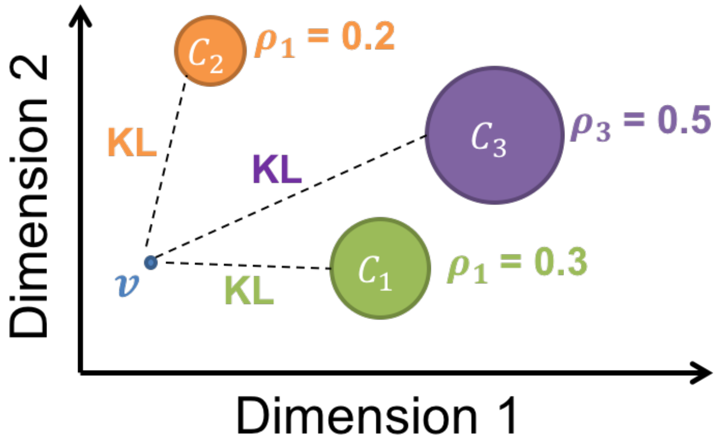


Figure 9.2: **EdgeCentric can handle multifaceted edge behaviors.** Depicted is a toy example in which a user v's multifaceted weighted abnormality score is computed with respect to clusters $C_1 \dots C_3$ and their respective, varying proportions $\rho_1 \dots \rho_3$. Notation is simplified for illustration purposes given the "user" context.

In fact, δ_{base} can be extended to incorporate such multifaceted behaviors without much complication. The base formulation assumes the existence of a single, global model C which describes the attribute distribution over all edges. In the user-rates-product scenario, we can consider C to be a discrete probability distribution defined over the 1-5 rating values. To capture the notion of multiple models of rating (and without loss of generality, attribute) behavior, we introduce the notation $C_{u,j}$ and $C_{p,k}$ to denote the jth model distribution for user ratings and the kth model distribution for product ratings[1], where $j \in \{1 \dots s\}$ and $k \in \{1 \dots t\}$ given s total user rating distributions and t total product rating distributions. We can consider these as clusters which describe various modes of rating behavior. In addition to the cluster distributions, we also define their proportions $\rho_{u,j}$ and $\rho_{p,k}$ as the fraction of user and product nodes which belong to the jth and kth clusters respectively – here, we consider that a user U belongs to a cluster j if the L^2 distance from $\hat{U}$ is smaller to the distribution $C_{u,j}$ than for all other clusters $\{1 \dots s\} \setminus \{j\}$. The analogous definition applies to a product P and cluster k. Note that with the introduction of such a multifaceted model, our model distribution C is defined *separately* for user and product

[1]In general, we have for each object type $b \in B$ corresponding cluster distributions $C_{b,i}$.

ratings – this is in contrast to the definition when we considered a single, global model. The distinguishing factor is that in considering multiple clusters, the patterns in how users rate and how products are rated can actually differ depending on the specific edge structure of G.

In this case, we face the problem of identifying abnormality as a function of multiple clusters rather than just a single one. The abnormality of a node should also reflect to what extent its behavior fits with these various cluster distributions – for instance, even if there are two clusters of user rating behavior, if one cluster is more widespread and characteristic of general user rating behavior than the other, this factor should be intuitively accounted for in the scoring. To account for this concept, we introduce the following definition of the multifaceted abnormality scoring function δ_{mf}:

Definition 9.2: Multifaceted Score

Given a single edge attribute and h cluster distributions of type $b \in \mathcal{B}$ indicated by $C_{b,g}$ where $g \in \{1 \ldots h\}$, the multifaceted abnormality scoring function δ_{mf} for a node $v \in \mathcal{V}$ with $\Psi(v) = b$ is defined as

$$\delta_{mf}(v) = |f_v| \cdot \sum_{g=1}^{h} \big(\rho_{b,g} \cdot KL(\hat{v} \parallel C_{b,g})\big)$$

where $|f_v|$ gives the cardinality of the edge-attribute value vector f_v produced from v's neighboring (outgoing) edges, $\hat{v}$ gives the discrete probability distribution associated with node v over the chosen attribute, and $C_{b,g}$ and $\rho_{b,g}$ give the gth model distribution and proportion of the gth cluster respectively.

This scoring function intuitively gives the *expected* number of extra bits required to encode the behavior of v on a single edge attribute with respect to multiple cluster distributions. To see this, observe that δ_{mf} is in fact the expectation over a discrete random variable X with probability mass function defined by the cluster proportions $\rho_{b,g}$ for $g \in \{1 \ldots h\}$, and outcomes defined by $\delta_{base}(v)$ for $v \in V$ and cluster distribution $C_{b,g}$ – a visual depiction is given in Figure 9.2. This extension to the base formulation admits yet another desirable property:

Observation 9.3: Prevalence Sensitivity

Given two cluster distributions $C_{u,1}$ and $C_{u,2}$ with proportions such that $\rho_{u,1} > \rho_{u,2}$ and users U_1 and U_2 such that $\hat{U_1} = C_{u,1}$ and $\hat{U_2} = C_{u,2}$, if $KL(\hat{U_1} \parallel C_{u,2}) = KL(\hat{U_2} \parallel C_{u,1})$ and $KL(\hat{U_2} \parallel C_{u,1}) > 0$ and $|f_{u,1}| = |f_{u,2}|$ and $|f_{u,2}| > 0$, then $\delta_{mf}(U_1) < \delta_{mf}(U_2)$.

Observation 9.3 formalizes the intuition that if two users have no deviation from their own cluster distributions and equal deviations from the other cluster's distribution, and otherwise give an equal number of ratings, then the user who belongs to the smaller cluster is more surprising.

Note that by incorporating multiple patterns of edge behavior in this way, the multifaceted model inherently allow for the possibility of capturing abnormal behavior as part of the model itself. In fact, we may find groups of users who form their own clusters based on abnormal rating patterns as a result of fraud or suspicious activity. However, by computing the expectation over clusters using the cluster proportions as probabilities, we can still robustly identify abnormal users assuming they make up a small fraction of all users, given that they will deviate substantially from the rest of the data. The intuition is because although they may cost few bits to encode with respect to their own abnormal cluster distribution, they will still cost many bits to store with respect to the other cluster distributions, which are weighted much more substantially due to their larger constituency in the data.

9.3.4 Handling multiple edge-attributes

We now broach the topic of building a joint abnormality function which incorporates the presence of multiple edge attributes in addition to multifaceted models on each of the individual attributes. This is particularly useful in practical applications, where service providers collect a variety of information about each interaction. For example, in the user-rates-product scenario, practitioners may also collect auxiliary information about the rating interaction included timestamp, rating/review text, or verification information (indicating whether a purchase occurred or not). Each of these attributes collects information about a different aspect of the interaction which may indicate fraudulent, suspicious or otherwise anomalous but simply interesting behavior. For example, consider a user whose given rating distribution was not itself atypical, but had a consistent inter-arrival time (IAT) of 5 seconds between ratings, meaning that each subsequent rating was given 5 seconds after the previous – it is apparent in such a case that this reviewer's abnormality would not be well-indicated on the rating attribute, but would appear strongly on the temporal attribute

There are a number of strategies we could employ for incorporating multiple attributes into the ranking context. One strategy is to consider ranking in a subspace formulation, where we consider abnormality with respect to various subspaces of edge attributes. However, this approach introduces an intractable number of subspaces along with sparsity issues, especially for high-dimensional data.

A second strategy is to consider abnormality additively over each of the attributes, assuming independence. In this approach, we compute the δ_{mf} score for each user over each attribute and simply sum the scores together. We find that this approach offers numerous comparative advantages over the previously mentioned joint subspace method. Firstly, instead of focusing on the combinatorial number of underlying subspaces, we focus on just a single space. This gives us a single abnormality ranking in which the top-ranking users are those who score highly in abnormality on many or all attributes. Furthermore, defining an additive measure of abnormality offers an attractive interpretation from the compression perspective – it can be interpreted as the expected number of extra bits to encode a node's actions with respect to a joint (but independent) model over all edge attributes. values for that given attribute, the sum represents the expected number of extra bits to encode a user's values with respect to a joint model (though formulated independently) over all edge attributes. The summed abnormality scores are thus naturally weighted by the deviation in terms of information content (in bits) from their respective attribute models.

We slightly modify our existing notation from the multifaceted (multiple clusters per attribute) model to distinguish cluster distributions between attributes $w \in \{1 \dots y\}$ on a single relation. Now, instead of $C_{u,j}$ and $C_{p,k}$ to denote the jth cluster distribution for user ratings and kth cluster distribution for product ratings, we write $C_{u,w,j}$ and $C_{p,w,k}$ to denote the jth user cluster distribution and kth product cluster distribution for attribute w, respectively. Similarly, we write proportions as $\rho_{u,w,j}$ and $\rho_{p,w,k}$ for the proportion of the jth user cluster and kth product cluster for the wth attribute, respectively. Additionally, each attribute w may have a different number of user and product clusters so we write $j \in \{1 \dots s_w\}$ and $k \in \{1 \dots t_w\}$ where s_w and t_w denote the total number of user and product cluster distributions for the wth attribute, respectively. Thus, we define δ_{ma} as follows:

Definition 9.3: Multi-attribute Score

Given multiple edge attributes $w \in \Omega(r)$ defined on a single relation $r \in \mathcal{R}$, with h_w cluster distributions of type $b \in \mathcal{B}$ respectively indicated by $C_{b,w,g}$ where $g \in \{1 \dots h_w\}$, the multi-attribute abnormality scoring function δ_{ma} for node $v \in \mathcal{V}$ with $\Psi(v) = b$ is defined as

$$\delta_{ma}(v) = |f_v| \cdot \sum_{w \in \Omega(r)} \Big(\sum_{g=1}^{h_w} \big(\rho_{b,w,g} \cdot KL(\hat{v_w} \parallel C_{b,w,g}) \big) \Big)$$

where $|f_v|$ gives the cardinality of the edge-attribute value vector f_v produced from v's neighboring (outgoing) edges, $\hat{v_w}$ gives the discrete probability distribution associated with node v over attribute w, and $C_{b,w,g}$ and $\rho_{b,w,g}$ give the gth model distribution and proportion of the gth cluster on the wth attribute respectively.

9.3.5 Handling multi-relational graphs

Thus far, we have built up δ_{ma} as an abnormality scoring function which handles multiple edge attributes with multifaceted models indicating various clusters of node behavior. Now, we briefly discuss how to extend this scoring function to more complex heterogeneous schemas with multiple relation types ($|\mathcal{R}| > 1$). Handling multiple relation types is yet another factor which can enable richer anomaly detection. For example, consider that in our running user-rates-product scenario, we additionally incorporate a new object type of seller, and introduce a new relation user-rates-seller. Now, one can envision a similarly motivated scenario for the multi-attribute formulation – however, instead of a user giving typical rating values with atypical IATs, the user could now give typical rating values even with typical IATs for products but not for sellers. Thus, considering only the user-rates-product relation for the user, we might not be able to identify a user as abnormal using the δ_{ma} score. However, incorporating the secondary user-rates-seller relation, we are able to appropriately penalize the user's atypical behavior.

Fortunately, extending the formulation to handle multiple relations per object follows a very similar argument to the multi-attribute scenario where we consider handling multiple attributes per relation. We now define a joint model on the object type which incorporates multiple relations per object, and multiple attributes per relation. Given such a model, users who behave atypically on multiple types of interactions will be considered the most abnormal. We can again devise an additive formulation with a minor modification to notation – given that a user may have rated a different number of products than sellers, we use the notation $f_{u,r}$ for user U's vector for relation r, and $|f_{u,r}|$ for the size of the attribute vector. Similarly, we write $f_{p,r}$ and $|f_{p,r}|$ for product P's vector and the associated size for relation type r. Then, we define the unified heterogeneous, multi-attribute and multifaceted abnormality scoring function δ as follows:

Definition 9.4: Unified Score

Given multiple edge attributes $w \in \Omega(r)$ defined on multiple relations $r \in \mathcal{R}$, with h_w cluster distributions of type $b \in \mathcal{B}$ respectively indicated by $C_{b,r,w,g}$ where $g \in \{1 \ldots h_w\}$, the unified abnormality scoring function δ for a node $v \in \mathcal{V}$ with $\Psi(v) = b$ is defined as

$$\delta(v) = \sum_{r \in R} \left(\sum_{w \in \Omega(r)} \left(|f_{v,r}| \cdot \sum_{g=1}^{h_w} \left(\rho_{b,r,w,g} \cdot KL(\hat{v_w} \parallel C_{b,r,w,g}) \right) \right) \right)$$

where $|f_{v,r}|$ gives the cardinality of the edge-attribute value vector $f_{v,r}$ produced from v's neighboring (outgoing) edges of type r. Formally, $f_{v,r} = \{e \in E \mid v \in m(e) \wedge \Psi(e) = r\}$. Furthermore, $\hat{v_w}$ gives the discrete probability distribution associated with v over attribute w, and $C_{b,r,w,g}$ and $\rho_{b,r,w,g}$ give the gth model distribution and proportion of the gth cluster on the rth relation type respectively.

Note that the definition of δ given in Definition 9.4 is the final formulation of the abnormality scoring function. From a compression perspective, it gives the expected number of extra bits required to encode a given node's edge-attribute vectors with respect to a joint model over multiple relations, multiple attributes and multiple per-attribute clusters. In the user-product-seller scenario, we can consider that for each user, we compute deviation with respect to a joint model over the user-rates-product and user-rates-seller relations, each of which has multiple attributes (rating value, IAT, etc.) and various clusters representing patterns of behavior. The definition is general, and extends to various node types with various numbers of relations and attributes.

Algorithm 9.1: EdgeCentric

```
Input: graph $G$
Output: sorted abnormality score vector for each node type in $G$
1: For each node in $G$, aggregate attribute values from outgoing edges per-relation-type.
2: Based on attribute type and range of values, discretize the space categorically for
   categorical attributes, and linearly or logarithmically for numerical attributes. Bin the
   per-node aggregated attribute values accordingly and normalize to construct probability
   mass functions.
3: For each node-type and attribute, cluster the vectors describing the per-attribute probability
   mass functions associated with each relation.
4: For each node-type, compute the abnormality score $\delta$ for all nodes over associated relations
   and attribute clusters.
5: For each node-type, sort (descending) the resulting abnormality scores and return with
   node indices.
```

9.4 Proposed Method: EdgeCentric

Thus far, we have built up both intuition and formalization for the use of δ as an abnormality score for nodes in edge-attributed graphs. We next describe our EdgeCentric algorithm, which draws the attention of the analyst/practitioner to the nodes with the most surprising behavior in the given network. The pseudocode for EdgeCentric is given in Algorithm 9.1 – we describe the five associated key steps below.

Step 1 – Aggregation: For each node-type in G, we aggregate the attribute values over the outgoing edges from each node for each associated relation-type. In our user-rates-products scenario, we have two node-types (users and products) connected by a relation with two attribute types: ratings (categorical) and timestamps (numerical). Since our relation is undirected, for each node we aggregate the attribute values for the adjacent edges, thereby collecting a vector of rating values for the user (product) as well as a vector of associated timestamps.

Step 2 – Discretization: Given the attribute types and ranges, we discretize the value space of each attribute in a principled manner. Categorical data is by definition discrete and thus does not need further processing. For numerical attributes, the discretization process requires more sophistication. We propose an adaptive binning approach as follows: if the maximum value of the attribute is an order of magnitude larger than the minimum, we space the bin markers logarithmically into d bins ($d = 20$ in our experiments). Otherwise, we space the bins linearly. Logarithmic binning addresses issues associated with sparsity and scale insensitivity in large-ranged data. In smaller ranges, linear binning tends to be sufficient. For temporal data, instead of binning timestamps, we bin the inter-arrival times (IATs) instead to reflect time between actions.

Step 3 – Clustering: After binning the per-node attribute values and normalizing to construct the appropriate probability mass functions, we cluster the vectors describing the probability masses as a number of d-dimensional points. Though any clustering algorithm could be used

Graph	Nodes	Edges
`Flipkart` [fli]	1.1M users, 545K products	3.3M ratings
`SWM` [ACF13]	964K users, 15K applications	1.1M ratings
`AmazonHPC` [MPL15]	1.8M users, 252K products	3.0M ratings

Table 9.3: Datasets used for empirical analysis

for this purpose, we use X-means [PMO00], as it automatically chooses the number of clusters in a principled manner by optimizing Bayesian Information Criterion (BIC). The centers of the resulting clusters are d-dimensional probability mass functions themselves, which we use as the cluster distributions. We can then compute cluster proportions by empirically assigning the input points to clusters by smallest ℓ_2 distance.

Step 4 – Scoring: Given the cluster distributions across all attributes and node-types over the respective relations, we now compute the abnormality score $\delta(v)$ for each node $v \in \mathcal{V}$ according to Definition 9.4. For each node-type, and over each of the attributes on associated relations, we *additively* compute the abnormality score in terms of the expected cost in extra bits with respect to the attribute cluster distributions.

Step 5 – Ranking: Finally, we sort the scores for each node-type in a descending fashion and return the ranking with associated node indices to the practitioner. This effectively routes practitioner attention to the most abnormal nodes for each of the node-types in the graph (users, products, etc.), with respect to encoding cost over a joint model composed of independent edge-attribute models. This information can then be leveraged for further investigation.

Note that while EdgeCentric is motivated by the utilization of edge-attributes in interaction networks, it can also be applied in non-network settings given attribute distributions for each object in an object set are known.

9.5 Experimental Analysis

In this section, we evaluate EdgeCentric and aim to answer the following questions: what kinds of edge-attribute behavior do we observe in real graphs? Is EdgeCentric effective in finding abnormally-behaving nodes by leveraging this information? Finally, is EdgeCentric scalable?

9.5.1 Datasets and Experimental Setup

For our evaluation, we apply EdgeCentric to 3 real-world graphs with various edge-attributes. The datasets are summarized in Table 9.3 and described in further detail below.

Flipkart: The `Flipkart` dataset contains information about reviews and ratings in the Flipkart e-commerce network which provides a platform for sellers to market products to customers. It contains roughly 3.3 million ratings given by 1.1 million users to 545 thousand products from Aug. 2011 to Jan. 2015.

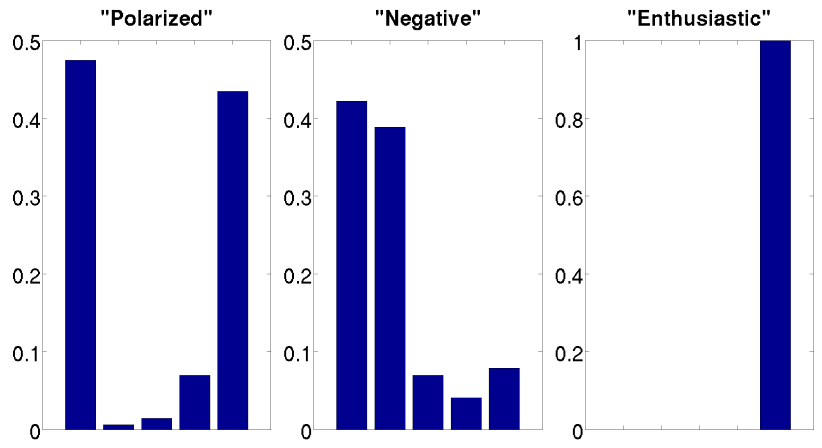


Figure 9.3: **Discovered popular user-rating patterns.** Here, we show several cluster distributions and associated probability masses for user ratings on the `Flipkart` dataset – bins correspond to 1-5 stars.

Software Marketplace: The `SWM` dataset contains information about purchases in an online marketplace which allows customers to purchase software applications. The data for this marketplace was originally collected in [ACF13]. It contains over 1.1 million ratings given by 964 thousand users to 15 thousand applications over the timespan of Apr. 2008 to June 2012.

Amazon Health and Personal Care: The `AmazonHPC` dataset is publicly available. It contains information about online purchases of health products on the Amazon e-commerce network. It contains roughly 3.0 million ratings given by 1.8 million users to 252 thousand products over the timespan of May 1996 to July 2014.

Our code for EdgeCentric is primarily implemented in Python. For experiments, we use a machine with 32 Intel Xeon 8837 CPU cores @ 2.67GHz and 1TB RAM.

9.5.2 Findings on `Flipkart`

In our analysis on the `Flipkart` dataset, we constructed a single relation (user-rates-product) on which we had one categorical attribute (rating from 1-5) and one temporal numerical attribute (UNIX timestamp). Thus, we ranked abnormality of users with respect to their rating and IAT behavior.

Figures 9.3 and 9.4 show the probability mass functions corresponding to several of the prevalent distributions that we found as a result of clustering the user edge-attribute data. Figure 9.3 shows several interesting rating patterns we discovered from the process: *polarized*, *negative* and *enthusiastic* users. We characterize these patterns as follows:

- *Polarized* users give mostly 1 star and 5 star ratings, with very few middle-ground ratings – this can correspond to the natural tendency to either love or hate a product, or result from fraudulent users who aim to popularize all the products of a single seller, and defame the competitors.

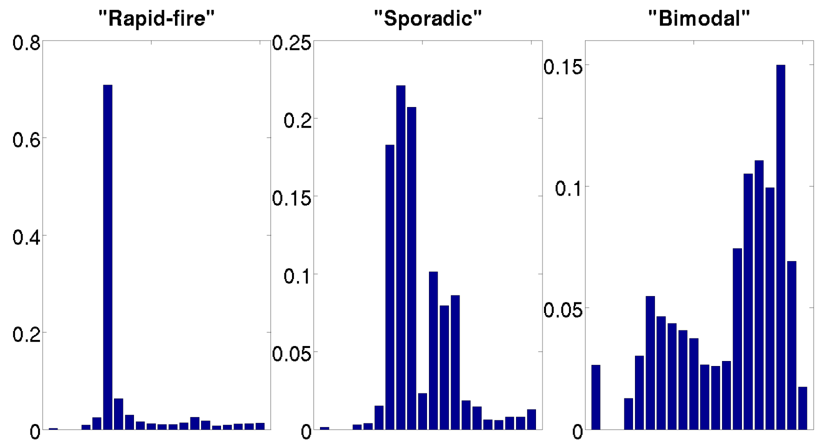


Figure 9.4: **Discovered popular rating frequency (IAT) patterns.** Here, we show several cluster distributions and associated probability masses for IATs between ratings on the `Flipkart` dataset – bins correspond to logarithmically discretized interarrival times (the first few bins span IATs from several seconds to just a few minutes).

- *Negative* users give mostly 1 or 2 star ratings – we conjecture this is mostly a consequence of response bias, where users are sharing their opinions only because they are especially displeased with a product.
- *Enthusiastic* users give only 5 star ratings and none others – this is suggestive of strong response bias or blatantly fraudulent behavior (especially when the user gives many such ratings).

We additionally find isolated clusters for users who give only ratings of a single star outcome (1-5) – these *single-minded* individuals are particularly prevalent in the data, given the large number of low-activity users who rated only one or a few products since inception. The presence of these behaviors in various proportions of the data then informs the computation of the abnormality scores and EdgeCentric rankings for individual users.

Figure 9.4 shows several IAT patterns (indicating rating frequency), selected from clusters produced from the X-means process. The bins are discretized logarithmically, so that the span of the first few bins corresponds to IATs between 0 seconds to roughly 10 minutes, whereas the latter bins span from 10 minutes to several years (normally the case for users who rate only a few products in total, with a large gap between subsequent uses of the Flipkart platform). We found several interesting IAT patterns, including those of *rapid-fire*, *sporadic* and *bimodal* users. We characterize these as follows:

- *Rapid-fire* users are the most blatantly suspicious – these users almost exclusively give ratings with just a few seconds or minutes between subsequent ones. This type of behavior is almost guaranteed to be fraudulent and does not correspond with any intuition of real human behavior.
- *Sporadic* users' behavior is far more in-line with human intuition. These users mostly give ratings several days, weeks and months apart. Few ratings are given with very short IATs,

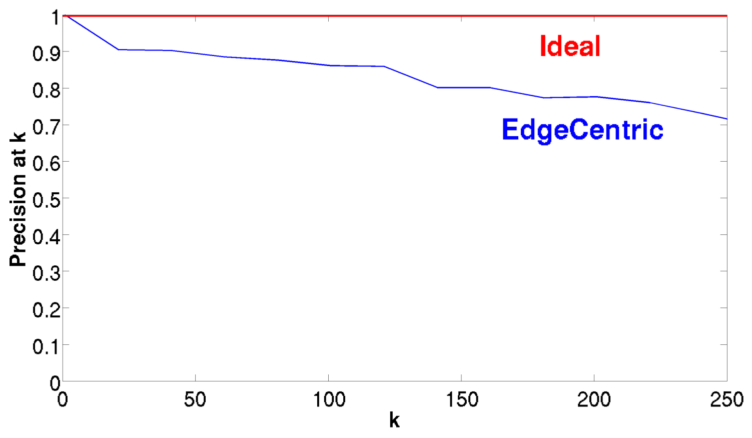


Figure 9.5: **EdgeCentric finds fraudulent accounts on Flipkart with high precision.** Here, we show the precision@k for various values of k ranging from 1 to 250, based on hand-labeled data from domain experts at Flipkart.

indicating that the users do not rate many items in a single purchase, and purchase only sporadically (grocery products, birthday presents, holiday gifts, etc.)

- *Bimodal* users occasionally spend weeks to months without rating a product, but often have periods of frequent activity on the order of multiple ratings (purchases) in days to weeks. Notice that the probability mass for the users in this cluster is distributed across almost all orders of IAT, with most of the mass concentrated on the orders of days to weeks, suggesting that the users are engaged with the Flipkart service and give ratings frequently (presumably because they also purchase products frequently). However, a non-trivial amount of the mass is distributed between shorter timeframes of seconds to minutes, indicating that the users rate multiple products in a single sitting (likely due to the purchase and resulting receipt of several products at the same time).

Upon applying EdgeCentric to this dataset, we provided a list of the 250 most abnormal accounts to domain-experts at Flipkart who investigated and labeled these users individually according to various criteria involving the user's review-text, rating distributions and frequencies. Figure 9.5 shows the precision at k (P@k) for a spread of k values over this range of 250 users, indicating positive results of 90% precision over the top 50 users, and over 70% precision over the top 250 users – recall results are incalculable given unbounded false negatives and lack of ground-truth labels. These are substantial findings for Flipkart – given previously unsophisticated fraud detection approaches, most fraudsters did not have to resort to distributed attacks (the fraudulent users had each committed hundreds to thousands of actions). One common pattern found by domain-experts was that most fraudsters either spammed 4/-5 star ratings to multiple products from a single seller (boosting seller ratings), or spammed 1/2-star ratings to products from another seller (defaming competition). We further found that the most abnormal user had given *3692 5-star ratings* with an average IAT of just a *few seconds.*

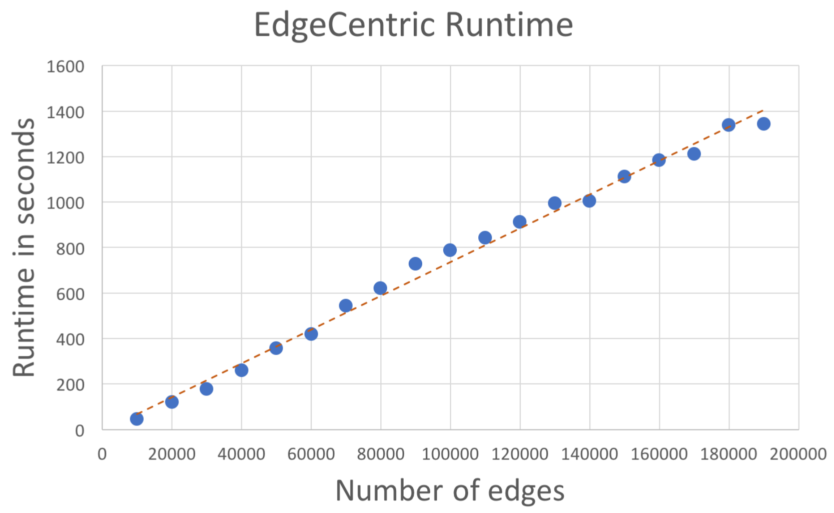


Figure 9.6: **EdgeCentric scales linearly.** Here, we show EdgeCentric's runtime on induced subgraphs of the `Flipkart` dataset.

9.5.3 Findings on `SWM`

On the `SWM` dataset, we constructed a single relation (user-rates-application) on which we had one categorical attribute (rating from 1-5). Thus, we ranked abnormality of users with respect to their rating behavior. We do not show the clustered rating behavior in interest of space, but note that similar behaviors can be observed in this dataset in terms of polarized raters, "single-minded" raters, etc. as in Figure 9.3.

We find that the users with the highest scores according to our EdgeCentric approach have spammy behavior. The most abnormal user in this dataset had given *186 5-star ratings* to a single application. The accompanying reviews had very high textual similarity and included quotes like

> *"Awesome!!!,Get this app now and earn points for a $10 gift card."*
> *"Try the app today and you will be amazed of how much money you can make with it......"*
> *"Awesome App!!!! FREE money ,The app is great to earn points for FREE money. Get it today!"*

Another of the highly-ranked users had given 107 5 star ratings to a single application, spamming the following review:

> *"Great app,Just great! Enter code: [redacted] To win even more points!!!!!!!!!!!!!!!!!!!!"*

In fact, the top-ranked 20 users according to EdgeCentric often posted repetitive, spammy text in addition to highly skewed ratings. Usually, the review text promoted the application, included personalized codes which the reviewers claimed would give customers free money/points, or were generally characteristic of information-free content. We additionally found correspondences between the codes reviewers asked customers to use and the reviewer's own usernames, suggesting that the code gave the reviewer an associated perk rather than the customer. It stands to reason that the associated applications incentivized existing customers to attract more potentials. Unfortunately, we are unable to check for ground-truth with service providers.

9.5.4 Findings on `AmazonHPC`

For the `AmazonHPC` dataset, we constructed a single relation (user-rates-product) on which we have one categorical attribute (rating in stars from 1-5) and one temporal numerical attribute (UNIX timestamp of rating). Thus, we ranked abnormality of users with respect to their rating and IAT behavior. Clustered rating behavior is not shown in interest of space limitations, but we find similar overarching patterns as in Figures 9.3 and 9.4.

In the `AmazonHPC` dataset, we observe that the top 5 users with the highest scores are in fact reviewers with high-status badges who review products which they receive free of charge in return for positive rating and review. Good evaluations from these reviewers are seen as a status symbol for the product.

The most abnormal user according to EdgeCentric had rated 348 products with *317 of the ratings being 4/5-stars.* A quick look at his Amazon review profile shows many similar reviews including the statement

> *"Note: sample unit provided for reviewing purposes."*

Another abnormal user ended almost all of her reviews with

> *"[Seller] provided [Product] for evaluation and review."*

At the time of data collection, the user had rated 285 products with *262 4/5 star reviews*, of which almost *80% were 5 stars.* Interestingly, reviewers who receive products for evaluation tend to give abnormally many more positive ratings than others and appear disingenuous.

Several of the other top reviewers have similar badges, indicators, and review styles – 2 of the top 5 have *Vine Voice* badges, indicating that the reviewer is a member of the invitation-only *Amazon Vine Voice* program which gives customers advance access to unreleased products for the purpose of writing reviews. [NPR] provides further details from one such top-reviewer who details his experiences in having received thousands of dollars of free products from Amazon sellers over the years for evaluation purposes.

It is comparatively difficult to make claims about the legitimacy of products. This is both because products have inherent differences in quality and frequency of purchase, and also spammers rating the product very highly or poorly do not necessarily imply the illegitimacy of a product or its seller. Interestingly, the most abnormal product was a digital weighing scale from the *EatSmart* product line, with 19,593 reviews of which over *80% are 5-star.* This is an unusually strongly rated product in comparison with others in the dataset, and also rated with very high frequency (short IAT). Several of the other highly abnormal products are fitness products including the *BlenderBottle* and *FitBit*, which have *91% and 85% 4/5 star ratings* over 10 thousand reviews each. The distributional deviance is unusually high and done in large-scale, suggesting suspicious activity.

9.5.5 Scalability

Finally, we show that EdgeCentric is scalable on real-world graphs. Figure 9.6 shows the linear runtime in seconds on various induced subgraphs from `Flipkart`. The time-complexity of EdgeCentric is roughly $O(|\mathcal{E}|d + |\mathcal{V}|kdi)$ for a single attribute, where $|\mathcal{V}|$ and $|\mathcal{E}|$ are the number of nodes and edges in G, d is the attribute dimensionality, k is the number of clusters and i the number of clustering iterations. The former term reflects the cost of binning the attributes, whereas the latter term reflects the cost of clustering points using naïve Lloyd's k-means. The time complexity of X-means is ill-defined given its inherent dependence on the number of underlying clusters in the data, but is empirically shown to scale better than k-means due to the diminishing size of the clustering problem in each iteration.

9.6 Conclusion

In this work, we broach the issue of detecting anomalies in large, edge-attributed real-world graphs, which are commonplace in modern e-commerce platforms, social networks and other web services. Specifically, we first formalize the problem of detecting anomalous nodes in graphs as an unsupervised ranking problem, in which we aim to score nodes based on the abnormality of their edge behavior. To this end, we first build up the intuition of using information theoretic principles to quantify deviation from typical behavior in a data-driven fashion, and extend this formulation in the presence of multiple user behaviors, multiple edge-attributes and complex heterogeneous graphs. We then introduce the EdgeCentric approach to leverage this formulation. Finally, we show substantiating results including high precision (*87%* over the top 100 users) on the Flipkart e-commerce platform, practical scalability and interesting observations on typical and atypical user behaviors gleaned from applying our method to several large, real-world networks.

Based on content published in [SLBF17].

Chapter 10

The Many Faces of Link Fraud

Characterizing the multimodality of link-fraud using honeypots.

Most prior work on social network link fraud detection tries to separate genuine users from fraudsters, implicitly assuming that there is a single "pattern" of fraudulent behavior. But is this assumption true? And, in either case, what are the characteristics of such fraudulent behaviors? Characterizing and understanding fraudulent behavior and motives is crucial to building algorithms resistant to evolving fraudsters. In this chapter, we focus on exactly this. We set up *honeypots*, or "dummy" social network accounts on which we solicit fraudulent links, or followers. We report the signs of such behaviors, including oddities in honeypot local network connectivity, account attributes, and similarities and differences between various fraud providers. We detail our careful approach for data collection, discover and study multimodal fraud patterns, and give surprising insights into their behaviors and strategies. Finally, we propose a new class of features based on attribute entropy over first-order account followers, which we show gives near-ideal classification performance.

10.1 Introduction

What are the characteristics of fraudulent accounts in online social networks? Understanding the behavior and actions of fraudsters is paramount to building effective anti-fraud algorithms. While previous works in social network fraud detection have primarily focused on leveraging signature properties of fraudsters including temporally synchronized behavior [BXG+13], excessively dense [PSS+10] and oddly distributed [SBGF14] graph connectivity, uncommon account names [Fre13] and spammy links [GTPZ10], our work focuses on establishing the veracity and applicability of these assumptions. In doing so, we ask: do all fraudsters share the same signature behavior, or are there multiple signatures? Since fraud detection is an adversarial setting in which fraudsters are constantly adapting to in-place detection mechanisms, it is important to constantly monitor and evaluate the strategies that fraudsters are employing to profitably perform ingenuine actions to better inform future detection mechanisms.

We focus on one particular setting of social network fraud called *link fraud* which involves the use of fake, *sockpuppet* accounts to create links, or graph connections, which represent followership or support of target, customer entities. Fake links artificially inflate the follower count of customer accounts, making them appear more popular than they actually are. These fake links are deceptive to authentic users and hinder the performance of machine learning algorithms which rely on authentic user input to recommend relevant and useful content to their userbase.

To study the behavior of these fake follower accounts, we employ the use of *honeypots*, or dummy accounts on which we solicit fake Twitter followers sourced from various fraud service providers. Honeypots enable us to have an clear signal of fake follower activity which is not tainted by follows from real accounts. Upon setting up the honeypot accounts and purchasing fake followers, we instrument a number of carefully engineered tracking scripts which poll Twitter API to store details including account relationships and attributes over a period of time. This allows us to collect a rich representation of the fraudster ecosystem which we subsequently analyze.

In this work, we make and explore the following key observation:

Observation 10.1: Fraud Multimodality

There are multiple types of link fraud which exhibit notably different network structures and patterns in account attribute settings.

Specifically, we focus on studying and characterizing the network connectivity properties and attribute distributions which are exhibited by fake followers involved in these different types of fraud. We detail a number of further observations on how these types of behavior induce different, odd network structures and suspicious patterns in account attributes. Figure 10.1 shows the contrast in follower connectivity of a genuine account versus two distinct types of fraudsters. Through our analysis, we additionally engineer strong features which enable us to discriminate these fraudulent users from genuine ones using novel (first-order) follower entropy features.

Summarily, our work offers the following notable contributions:

- **Instrumentation**: We detail our experimental setup and data scraping tools which gather a wealth of Twitter user information while respecting API rate limits.
- **Observations on Fraud Multimodality**: We discover that link fraud is not unimodal and instead has multiple types, and identify and characterize two such types: *freemium* and *premium*, with the possibility of more.

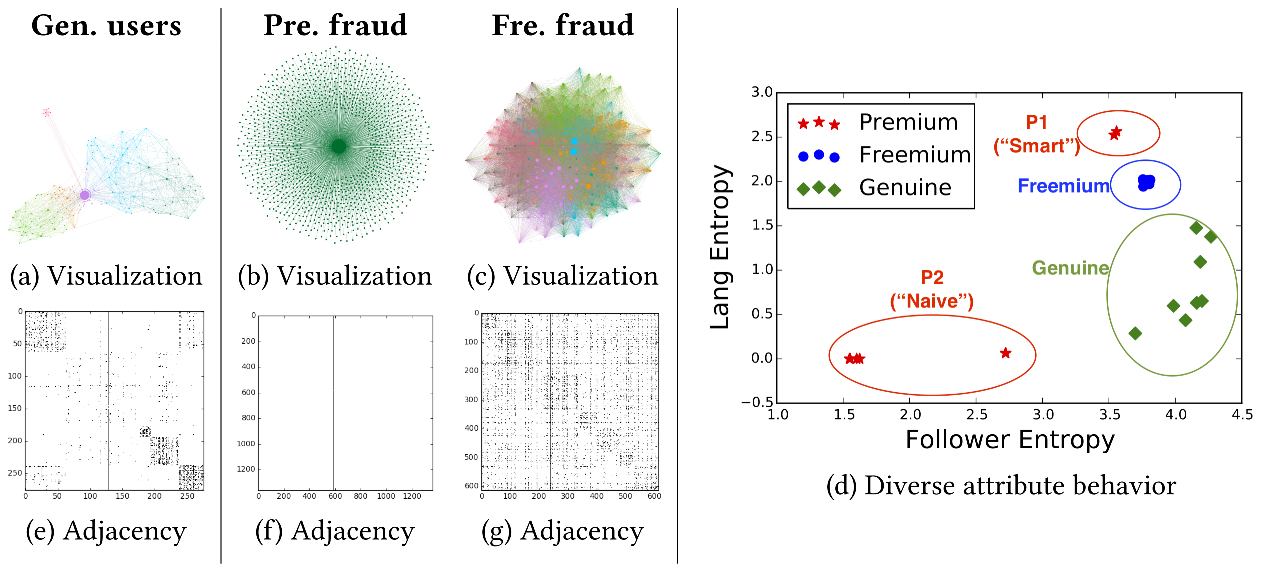


Figure 10.1: **Freemium (Fre) and premium (Pre) fraud types have different local network structure and account attributes compared to genuine behavior.** Nodes are colored by modularity class, and sized proportional to in-degree in (a)-(c). The associated, reordered adjacency matrices are shown in (e)-(g) – the vertical line in each spyplot indicates the the central node. Notice the block community structure in genuine followers compared to the star structure for premium and near-clique structure for freemium followers. (d) shows differences in attribute (language and follower) entropy over the various behaviors, showing how fraud patterns skew attribute distributions away from genuine ones.

- **Features**: Based on the above observations, we carefully engineer novel, entropy-based features which allow us to accurately discern fraudsters from genuine users in our ground-truth Twitter dataset with near-perfect F1-score.

10.2 Related Work

We categorize related work into two categories: underground market studies and fraud detection approaches.

Underground Markets: Prior works have shown the use of fake accounts for followers in social media [TMG+13], phone-verified email accounts [TIB+14], Facebook likes [BXG+13], etc. These accounts are often used to spread spam [GTPZ10, GHW+10] and misinformation [GLK13, GLKJ13]. [Per13] estimates that the fake follower market produces $360 million per year. Recently, several works have studied the existence of underground online markets where these fraudulent actions can be purchased – [WWZ+12, MLK+10] explore underground markets providing fake content, reviews and solutions to security mechanisms. [TMG+13] studies several fraud providers over time and describes trends in pricing, account names and IP diversity. [SWE+13] compares growth rates of accounts with legitimate and fraudulent followers. [AK15] observes the varying retention and reliability of various fraud providers. Comparatively, our

Service	Type	Cost	Followers bought	Followers delivered	Followers remaining
fastfollowerz	Premium	$19	1000	1060	1059
				1060	1059
intertwitter	Premium	$14	1000	1099	977
				1102	974
devumi	Premium	$19	1000	1360	1358
				1354	1354
twitterboost	Premium	$12	1000	1361	1361
				1350	1350
plusfollower	Freemium	£9.99	1000	1094	748
				1078	737
hitfollow	Freemium	£9.99	1000	926	623
				937	638
newfollow	Freemium	£9.99	1000	884	600
				883	589
bigfolo	Freemium	£9.99	1000	872	594
				865	577

Table 10.1: Honeypot account summaries (two honeypots per service).

work is the first to identify major social graph differences between fraud types and across providers, and propose novel entropy-based features for capturing these behaviors.

Fraud Detection: [BMRA10, LCW10] use profile features to detect spammers on Twitter. [SKV10] passively analyzes accounts with promiscuous following behavior and builds a classifier using profile and messaging features. [CJ12, YKGF06] aim to find fake accounts in social networks via a generative stochastic model and a random-walk based method respectively – both assume small cuts between fake and genuine nodes. [BXG+13, CYYP14] use graph-traversal based methods to find users with temporally synchronized actions on Facebook. [SBGF14, JCB+14a, PSS+10] propose spectral methods which identify dense or odd graph structures indicative of fraud.

10.3 Know Thy Enemy: Characterizing Link Fraud

In this section, we discuss some preliminaries about instrumentation, data collection and relevant metrics, and next illustrate numerous insights about network connectivity and account attributes of link fraudsters.

10.3.1 Setup and Data Collection

We first discuss how we identified and purchased followers from target fraud service providers, and next detail the scraping task, followed by preliminaries.

10.3.1.1 Purchasing Fake Followers

There are a number of different fraud service providers easily accessible and available on the web. We begin by identifying these services so we can purchase fake followers from them. To identify these services, we used Google search and queried using keywords such as "buy Twitter followers." Combining the search results, we obtained a list of websites which claim to provide these services.

From surveying the websites on this list, we notice there are several prevalent models of service – we categorize these into two frameworks: *premium* and *freemium*. Premium services offer customers multiple tiers of follower counts (1K, 5K, 10K, etc.) for various amounts of money and ask only for the customer's Twitter username and a form of payment. Freemium services offer both a paid option as in premium services, but additionally offer a free option which does not ask the user for money, but instead requires the user to provide their Twitter login details to the service. In return for these details, the services promise to direct a small number of followers to the account.

We next setup a pool of honeypot accounts by repeating the Twitter account creation process a number of times using monikers from online screenname generators. We found that to create a sizeable pool of honeypots, we needed to distribute the account creation over several IPs in order to avoid phone verification prompts. Upon setting up the pool of honeypots, we purchased basic follower packages from several premium and paid freemium services, avoiding rarely used ones with low Alexa rank. Summarily, we bought 1K followers from 8 different services (4 freemium, 4 premium) to 2 honeypot accounts per service. We chose to purchase 2 honeypot accounts per service instead of only 1 in order to examine the overlap dynamics of fake links to multiple customers. The final list of the services we used, service types, costs and their follower counts are summarized Table 10.1. Honeypots were created on the same day, and follower purchases were all done at the same time. Furthermore, the honeypots attracted no followers by themselves prior to the purchases. As a result, we posit that all followers of the honeypots are fake.

10.3.1.2 Instrumentation Details

Reproducibility: Code available at `https://goo.gl/qMBWim`.

We use the REST API to scrape data relevant to our operation from Twitter. As the API heavily rate-limits various data resource types, it is only feasible to extract a limited amount of information as an end-user. Prior to purchasing fake followers, we start a number of Python scripts which poll the API and insert data into a Postgres database:

Honeypot account details: Every hour, we collect public details for each honeypot Twitter account including number of friends and followees, number of favorites, number of Tweets, language, etc.

Honeypot account follower IDs: Every 12 hours, we collect the list of follower IDs for each honeypot. Since the honeypots were created with empty profiles, we can safely assume that all followers to these accounts were fraudulent and purchased.

Honeypot account follower details: Every day, we extract public details for each of the accounts in the honeypot follower list.

Honeypot account followers' friends/followers IDs: Every day, we collect the list of friend and follower IDs of the honeypot followers to examine their other connectivity.

Honeypot account followers' friends/followers details: Every 3 days, we extract public details for each of the friends and followers of the honeypot followers to gain more information about them.

Account details requests are limited to 15 requests per 15 minute window, and each request returns details for up to 100 accounts. Similarly, ID list requests are limited to 180 requests per 15 minute window, and each request returns up to 5000 account IDs. Hence, it is relatively easy to scrape the first-order honeypot account follower IDs and details without exceeding the rate limit, but collecting details for the second-order followers is a bottleneck. Since the number of nodes to collect information for can explode substantially even at the second-order, we limit collection to <100K friends and followers for each of the given follower of the honeypot account. We determine periodicity values empirically using back-of-the-envelope calculations. While this data could be collected slowly using a single Twitter API key, we speed up the process by using multiple keys and cycling keys upon resource exhaustion.

10.3.1.3 Preliminaries

In the remainder of our work, we conduct analysis on two types of networks: the *ego network* and *boomerang network*.

Ego network: An ego network (or egonet) traditionally consists of a central node called the ego, as well as the neighboring nodes and the relationships (edges) between them. Egonets can essentially be considered as a local graphical representation of a node within the context of the broader, global graph and depict how the surrounding nodes are connected. For our purposes, we examine *per-service* egonets, where we consider the union of the individual egonets of both honeypot accounts per service. Thus, in our case, each per-service egonet is actually comprised by 2 egos (the honeypot accounts), the union of both honeypots' neighboring nodes (the purchased, fake followers) and the relationships between them. The per-service egonet representation allows us to both individually study the *per-honeypot* egonets as well as any interactions between them. That is, if the two honeypots for each service have distinct sets of neighboring nodes, then their per-honeypot egonets will also be distinct. Conversely, if any nodes are neighbors of both honeypots, the associated per-honeypot egonets will be conjoined. Various levels of overlap suggest differences with regards to how services reuse accounts to deliver fake links.

Boomerang network: Drawing conclusions from per-service egonet analysis can be deceiving in the sense that while it does give insights into the *internal* relationships between the fake followers and honeypots, it does not consider the *external* relationships formed by the fake followers. As such, it is unable to give us a full perspective on the utilization of these fake followers. In order to gain the requisite perspective, we conduct analysis of the proposed boomerang network. We define the per-service boomerang network to be comprised of the per-service egonet *in addition to* the out-links of the follower nodes – the structure is reminiscent of a boomerang, in that it is comprised of the nodes "1 step back and 1 step forward" with respect to the honeypot account. Thus, the per-service boomerang network gives us an additional layer of information on top of the per-service egonet: connections to the other accounts followed by the honeypot's fake followers.

We further use the *density*, *bipartite density*, *transitivity* and *reciprocity* metrics to summarize and describe network structure, and *overlap coefficient* and *multiple systems estimation* (MSE) to characterize network overlap.

Density: We define density as

$$\frac{\#\text{edges}}{\#\text{nodes} \ \cdot \ (\#\text{nodes} \ - \ 1)}$$

Density represents the fraction of existing to possible total edges, with density 1 indicating a complete graph.

Bipartite density: We define bip. density between sets $\mathcal{A}$ and $\mathcal{B}$ as

$$\frac{\#\text{edges between } \mathcal{A} \text{ and } \mathcal{B}}{(\#\text{nodes in } \mathcal{A}) \ \cdot \ (\#\text{nodes in } \mathcal{B})}$$

Bipartite density captures the fraction of existing to possible edges between two sets of nodes, with bipartite density 1 indicating a complete bipartite graph.

Transitivity: We define transitivity as

$$\frac{3 \ \cdot \ \#\text{triangles}}{\#\text{connected triples}}$$

Transitivity denotes the degree of triadic closure, with transitivity 1 indicating that all connected triples of nodes are also triangles.

Reciprocity: We define reciprocity as

$$\frac{\#\text{bidirectional edges}}{\#\text{edges}}$$

Reciprocity conveys the relative frequency of bidirectional edges, with reciprocity 1 indicating that all edges are bidirectional.

Overlap coefficient: We define overlap coef. between $\mathcal{A}$ and $\mathcal{B}$ as

$$\frac{|\mathcal{A} \ \cap \ \mathcal{B}|}{min(|\mathcal{A}|, |\mathcal{B}|)}$$

Overlap coefficient indicates the proportion of members that overlap between sets, with overlap coefficient 1 indicating that $\mathcal{A} \subseteq \mathcal{B}$ or $\mathcal{B} \subseteq \mathcal{A}$ and 0 indicating $\mathcal{A} \cap \mathcal{B} = \emptyset$.

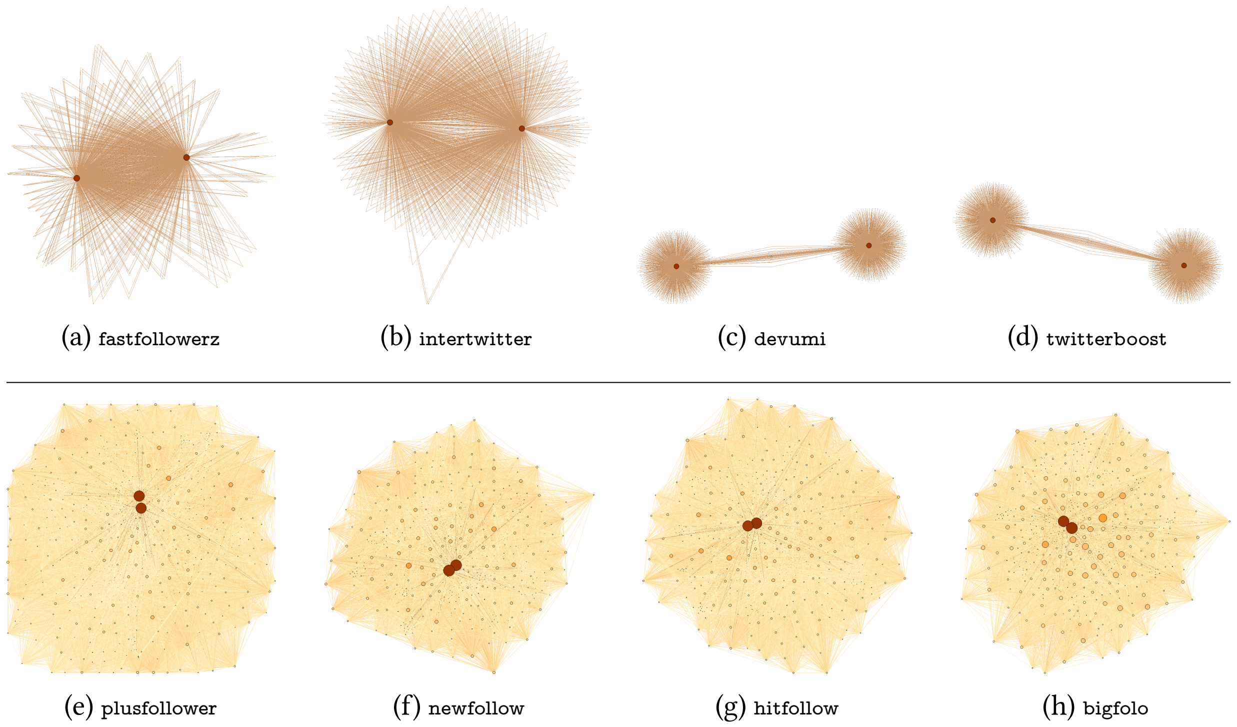


Figure 10.2: **Premium fraudsters (top) form overlapping stars whereas freemium ones (bottom) form dense, near-cliques.** Subplots show per-service egonets with honeypots in dark-red – darker color and larger size indicates higher in-degree.

Multiple systems estimation: We use MSE to estimate population size from two randomly sampled sets $\mathcal{A}$ and $\mathcal{B}$ as

$$\frac{|\mathcal{A}| \cdot |\mathcal{B}|}{|\mathcal{A} \cap \mathcal{B}|}$$

Intuitively, if $\mathcal{A}$ and $\mathcal{B}$ have low overlap, the total population size is much larger than if they have high overlap.

Upon shifting our discussion to account attributes distributions, we use *entropy* as a means to capture distributional skew.

Entropy: We define entropy for a distribution X with n outcomes $(x_1 \dots x_n)$ as

$$-\sum_{i=1}^{n} P(x_i) \cdot \log_2 P(x_i)$$

Entropy measures the unpredictability of a distribution in bits of information, with entropy of 0 bits indicating concentration of 100% probability on a single outcome, and entropy of $\log_2 n$ bits indicating uniform distribution of probability between n outcomes.

10.3.2 Network Observations

We first focus on studying the local network properties of fraudulent accounts. Targeting oddities in network connectivity is a central theme in many link fraud detection approaches, as the mission constraints of delivering fake links to customers necessarily affects graph structure. But what are these changes? In this section, we leverage social network analysis tools to characterize effects of fraud on the surrounding network structure, and show the similarities and differences between premium and freemium fraud. We detail analyses on two types of induced subgraphs: the ego network and more expansive boomerang network.

10.3.2.1 Ego Network Patterns

Figure 10.2 shows the per-service egonets for each of the 8 providers, with increased node size and darkness corresponding to higher in-degree. The honeypots (egos) are the two large and dark orange colored nodes in each subfigure. Cursory analysis reveals a notable difference in egonet network structure between freemium and premium providers. We see that the premium egonets (first row) have a star/bipartite structure: each honeypot node is the hub of a star, and the satellite nodes overlap and are disconnected. Conversely, freemium egonets have denser, near-clique type structure which suggests denser connectivity between the neighboring nodes.

The statistics for premium service egonets in Table 10.2 (top) further lend credence to the visual differences we observe from Figure 10.2, giving us the following insight:

Observation 10.2: Egonet Sparsity

Premium fake followers rarely follow each other, resulting in sparse egonet structure. Freemium fake followers have dense egonet structure.

This is substantiated by the low density and node to edge ratios across premium providers. Of these, `fastfollowerz` and `intertwitter` have an order of magnitude greater density than `devumi` and `twitterboost`. This is substantiated by the 1:2 node to edge ratio in the former 2 providers as compared to the near 1:1 ratios of the latter 2. `fastfollowerz` and `intertwitter` also have marginally higher transitivity values compared to the 0 transitivity of `devumi` and `twitterboost`, indicating that the former 2 have few triangles between the fake follower nodes whereas the latter 2 have none. We also observe no reciprocal links in these providers, indicating only one-way relationships.

Conversely, the freemium statistics in Table 10.2 (bottom) support that freemium fake followers have dense egonet structures. Freemium providers are an order of magnitude denser than the densest premium egonets – all 4 providers have 6-7% density. While not shown in interest of space, the per-honeypot egonets were each found to have an even higher 11-14% density individually. The 1:50 node to edge ratios substantiate this high density. We also notice that transitivity values are much higher for freemium providers, suggesting that an unusually high 28-30% of wedges are also triangles. Given that density and transitivity are equal in random

	Service	# Nodes	# Edges	Density	Transitivity	Reciprocity
Premium	fastfollowerz	1,066	2,289	.002	.001	.000
	intertwitter	1,051	2,003	.002	.00006	.000
	devumi	2,681	2,712	.0003	.000	.000
	twitterboost	2,680	2,711	.0004	.000	.000
Freemium	plusfollower	920	51,868	.061	.288	.411
	newfollow	755	37,052	.065	.294	.408
	hitfollow	782	41,879	.068	.305	.416
	bigfolo	749	36,043	.064	.294	.413

Table 10.2: Egonet summary statistics.

graphs, the freemium egonets do not appear to be random, but are likely composed of dense subregions which are themselves sparsely connected. The link structure reflects how freemium providers trade follows between accounts (random partitions, biased selection, account similarity, etc.) Furthermore, all 4 providers have similar, high reciprocity of 40-42% suggesting frequent "follow-back" behavior.

Rationale: The freemium services accumulates a pool of free accounts, and hence trading follows enables each free user to gain some followers. As a result, such behavior creates a denser subgraph, but are also used by providers to deliver the follower demands of paid customers and turn a profit. Comparatively, premium providers are unable to use free users' accounts and must create fake accounts.

These insights pose an interesting question: as we expect fraudsters to act in a manner that maximizes profit, *what motivates the differences in structure between freemium and premium providers?* We propose an answer: If we consider that each account has a budget of edges it can create without being suspended, it seems that premium providers greatly underutilize accounts compared to freemium ones. This is because for fraudsters, delivering more links while avoiding suspension is strictly better as it means that they can either serve more customers or artificially inflate their own popularity.

10.3.2.2 Boomerang Network Patterns

Figure 10.3 shows 2 boomerang networks, one for bigfolo and twitterboost, each representative of a different fraud strategy. Again, honeypot accounts are amongst the large, dark nodes with high in-degree, and the lighter, smaller nodes are fake followers or their friends. Note that the layout clusters nodes based on similar linkage, so groups of nodes visually close share connectivity properties. As with egonets, we again see a stark contrast in the boomerang structure of these two providers. Figure 10.3a shows the dense internal connectivity of bigfolo's fake followers (as we saw in Figure 10.2h), in conjunction with the sparser and less compact external connectivity to friends. Conversely, Figure 10.2d shows sparse internal connectivity between twitterboost's fake followers on the left, but dense near-bipartite external connectivity to the customers (including honeypots) on the right.

	Service	# Nodes	# Edges	Bip. Density
Premium	fastfollowerz	40,486	491,458	.012
	intertwitter	176,921	2,383,251	.013
	devumi	67,893	2,495,586	.014
	twitterboost	68,297	2,474,759	.014
Freemium	plusfollower	646,901	1,352,253	.002
	newfollow	616,824	1,221,574	.003
	hitfollow	558,100	1,172,248	.003
	bigfolo	574,823	1,157,672	.003

Table 10.3: Boomerang network summary statistics.

Table 10.3 (top) gives summary statistics about premium boomerang networks, which substantiate the following:

Observation 10.3: Boomerang Density

Premium fake followers are frequently reused to follow customers, resulting in dense external connectivity in the boomerang network. Freemium fake followers are less reused to follow customers, and hence have sparse external connectivity.

Interestingly, we see that the relative values of these statistics are inverted for the boomerang networks from the egonets – unlike for egonets where the density metric was an order of magnitude higher for freemium providers, the bipartite density in boomerang networks is instead an order of magnitude higher for the premium providers.Note that the premium providers' bipartite density indicates that nearly 1-2% (a huge amount) of all possible edges between the fake followers and their combined set of friends exists. The node to edge ratios are also much higher for premium providers – fastfollowerz and intertwitter are 1:14, and devumi and twitterboost are roughly 1:37 compared to only 1:2 for the freemium providers.

The freemium boomerang network statistics in Table 10.3 (bottom) again establishes the second part of the insight. This is further substantiated by the observation that freemium providers have an order of magnitude lower bipartite density than premium ones. We also observe that freemium boomerang networks have higher number of nodes than the premium counterpart. This is intuitive as freemium followers are otherwise genuine accounts, they have an expansive set of true friends, whereas premium fake followers are all synthetic accounts.

10.3.2.3 Network Overlap Patterns

In our analysis thus far, we noticed that various providers have different levels of evident overlap in the fake followers they deliver between their 2 honeypots. How extensive is this overlap? Do these providers reuse accounts in the same ways? Furthermore, is there any overlap between the followers across providers? Here, we shed light on these questions.

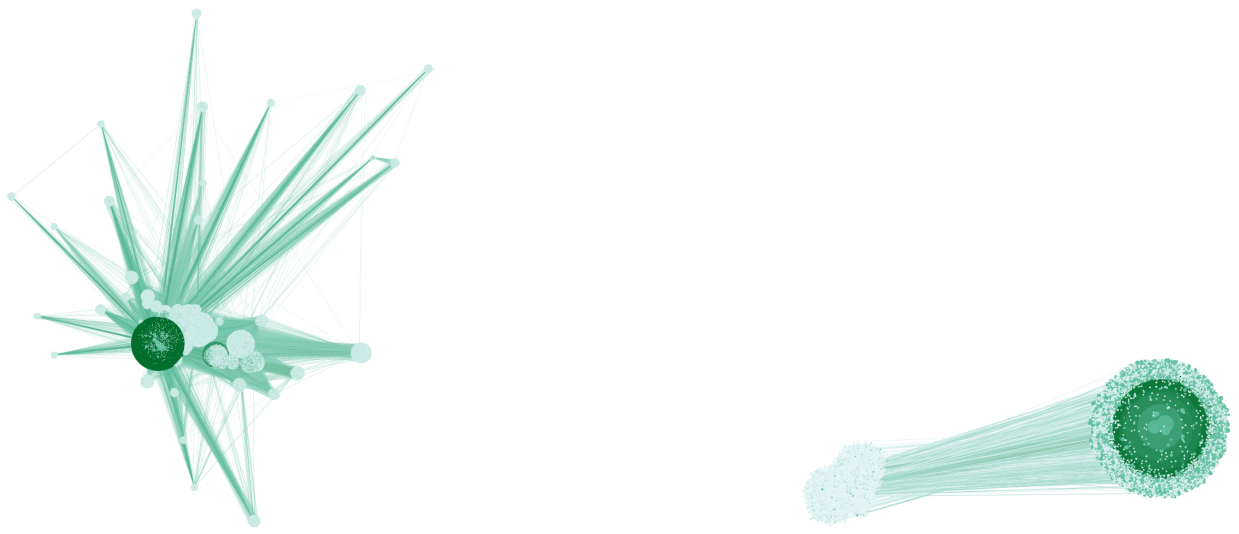

(a) bigfolo (fre.) (b) twitterboost (pre.)

Figure 10.3: **Freemium followers have dense internal and sparse external connectivity (left), and vice versa for premium followers (right).** Subplots show boomerang networks, with darker node color and larger size indicating higher in-degree.

	Service	# Nodes	Overlap	Est. Pool # Nodes
Premium	fastfollowerz	1,064	.996	1,064
	intertwitter	1,049	.953	1,051
	devumi	2,679	.024	55,719
	twitterboost	26,78	.024	55,677
Freemium	plusfollower	918	.815	954
	newfollow	753	.765	798
	hitfollow	780	.802	814
	bigfolo	747	.774	791

Table 10.4: Fraud providers have varying account reuse habits.

Intra-Network Patterns First, we study *intra-network overlap*, describing overlap between the fake follower nodes within each service. Table 10.4 shows the overlap coefficients between the honeypot followers for each service. Assuming the followers for each honeypot are randomly sampled from the service's account pool, we additionally compute the estimated total number of fake accounts currently in the fraud provider's hands using MSE.

The various degrees of overlap and commensurate estimates of pool size suggest the following insight:

Observation 10.4: Varying Delivery Structure

Service providers have varying methods for account reuse in efforts to to distribute suspicion across their account pools.

We observe that the freemium providers tend to have a high, 0.8 overlap which results in an estimated pool size slightly larger than either of the two sets of honeypot followers. However, the premium providers have an interesting split which reveals that `fastfollowerz` and `intertwitter` have very high, near 1.0 overlap, resulting in the pool size being roughly equal to each set of followers. This indicates that the pool is reused almost exactly for multiple customers. Conversely, `devumi` and `twitterboost` have near 0 overlap. As a result, we estimate that the pool size is quite large, containing over 55K total fake accounts.

While we cannot be certain without further investigation, these providers likely have different means of selecting and shifting the pool of active fake followers. For example, the pools used in `fastfollowerz` and `intertwitter` may cycle between a number of different "sub-pools" based on time, customer account features, or random choice. Conversely, the evidently much larger estimated pool size for `devumi` and `twitterboost` suggests that they may each have a single, large fixed pool of usable accounts from which followers are sampled regardless of other factors.

Inter-Network Patterns Thus far, we have established that providers reuse multiple follower accounts across customers in order to turn a better profit. But how far does this reuse go? Are any accounts responsible for delivering fake links to customers from different providers? To answer these questions, we study the pairwise *inter-network overlap* of followers between providers.

Table 10.5 shows an 8×8 matrix with the pairwise overlap coefficients. Given the number of nonzero entries, we draw the following surprising insight:

Observation 10.5: Collusion

Service providers seem to collaborate with and draw from each other to commit fraudulent actions.

We notice that there is substantial overlap within the freemium and premium providers. While `fastfollowerz` and `intertwitter` share no accounts with the other premium providers, `devumi` and `twitterboost` have a .07 overlap. Comparatively, all 4 freemium providers have a large 0.6-0.7 overlap, indicating that most of their fake accounts are *the same*. Furthermore, the set of followers for freemium and premium providers have 0 overlap, substantiating that followers in freemium providers are otherwise real accounts whereas those in premium providers are synthetic.

		fastfollowerz	intertwitter	devumi	twitterboost	plusfollower	newfollow	hitfollow	bigfolo
Premium	fastfollowerz	1.0	0	0	0	0	0	0	0
	intertwitter	0	1.0	0	0	0	0	0	0
	devumi	0	0	1.0	.07	0	0	0	0
	twitterboost	0	0	.07	1.0	0	0	0	0
Freemium	plusfollower	0	0	0	0	1.0	.65	.69	.64
	newfollow	0	0	0	0	.65	1.0	.64	.63
	hitfollow	0	0	0	0	.69	.64	1.0	.63
	bigfolo	0	0	0	0	.64	.64	.63	1.0

Table 10.5: Fraud providers share follower accounts.

Nonzero overlap between providers is an interesting finding – it is indicative of either a willingness to share follower accounts between fraud providers, or commonality in leaked or hijacked accounts. Upon further inspection, we notice a number of suggestive findings:

- Overlapping providers shared domain WHOIS protectors.
- Overlapping premium providers use the same Yoast SEO plugin and stylesheets.
- All freemium providers have two-column sites, advertised up to 30K followers, and priced from £9.99.
- All freemium providers contained the line: *"[service] is Not Affiliated With OR Endorsed By Twitter.com."*

10.3.3 Attribute Observations

In this section, we study the similarities and differences in account attributes of fake followers. Table 10.6 shows per-service, per-attribute entropy in bits for a variety of user attributes. The account attributes include creation year, default profile and profile image booleans, favorites count, followers count, friends count, lists count, statuses count, geolocation enabled boolean, language identifier, protected statuses boolean, UTC timezone, and a Twitter verification boolean which corresponds to high-profile, "famous" accounts. These attributes have varying outcome spaces. Creation date has 11 possible years (2006-2016), since Twitter was founded in 2006. Booleans have 2 possible outcomes (T,F). We encountered 35 different language identifiers and 39 UTC timezone settings. For count features, we logarithmically discretized the space into 32 bins from 1 to 1M to capture the wide range of activity levels. For each service, we aggregate attribute values and compute the entropy over the outcomes. The table shows the actual sample entropy in addition to the maximum possible (uniform) entropy. As previously mentioned, lower entropy indicates high synchronicity between followers. Note that a difference in entropy of 1 bit corresponds to twice the predictability.

The most striking insight from Table 10.6 is as follows:

	Service	Created (year)	Def. Prof.	Def. Prof. Image	# Favorites	# Followers	# Friends	# Lists	# Statuses	Geolocation	Lang.	Protected	UTC	Verified
Premium	fastfollowerz	1.37	.63	.01	3.65	2.73	2.73	2.99	3.8	.00	.06	.00	1.04	.00
	intertwitter	2.99	.82	.94	4.04	3.54	2.63	2.53	4.31	.67	2.55	.56	1.97	.18
	devumi	1.13	.97	.02	1.05	1.54	1.17	2.49	1.18	.00	.00	.00	1.42	.00
	twitterboost	1.13	.97	.03	1.05	1.56	1.16	2.51	1.15	.00	.00	.00	1.41	.00
Freemium	plusfollower	1.82	.93	.73	4.18	3.76	3.38	2.73	4.40	.54	2.04	.30	1.70	.00
	newfollow	1.68	.90	.75	4.20	3.70	3.32	2.64	4.37	.55	1.99	.28	1.62	.00
	hitfollow	1.78	.93	.73	4.14	3.76	3.32	2.72	4.37	.52	2.01	.30	1.70	.00
	bigfolo	1.88	.92	.75	4.20	3.74	3.34	2.72	4.40	.56	2.05	.32	1.71	.00
	Max Entropy:	3.46	1.00	1.00	5.00	5.00	5.00	5.00	5.00	1.00	5.13	1.00	5.29	1.00

Table 10.6: Per-service entropy (in bits) over account attribute distributions.

Observation 10.6: Entropy Gap

Premium service providers deliver followers with low entropy, high regularity attributes, whereas freemium service providers have more attribute disparity.

We notice that the premium providers have substantially lower entropy values in many attributes versus freemium providers, and even near 0 entropy in other attributes like geolocation. We elaborate on the specific differences next.

10.3.3.1 Account Creation

devumi, twitterboost and fastfollowerz have very low creation year entropy compared to freemium providers. While both freemium and premium accounts tend to be created more recently (perhaps because of higher suspension rate in older accounts), premium providers have a heavy bias towards recently created accounts (>2014).

10.3.3.2 Profile Defaults

fastfollowerz has a much lower entropy than other providers in terms of default profile – we found that >84% of these accounts did not have a default profile, whereas default profiles are actually *more common* than not in freemium accounts. Surprisingly, fastfollowerz, devumi and twitterboost also have near 0 entropy for profile image compared to the much higher entropy for freemium providers. We find that premium followers almost always set a custom image, suggesting that the information was fabricated or stolen from real users. Conversely, default profile images are common for freemium service accounts – this is intuitive, most real users do not fully customize their profiles.

10.3.3.3 Action Counts

devumi and twitterboost have much lower entropy for action counts (favorites, followers, friends, lists and statuses) compared to freemium providers. fastfollowerz also exhibits lower entropy. As Figure 10.1d shows, there is even more variation between premium providers. Figure 10.1d shows that intertwitter (P1 "smart") follower counts are disparate and closer to genuine users' entropy, unlike other premium fraudsters (P2 "naïve") who behave robotically. Comparatively, freemium followers have lower follower count entropy compared to genuine ones, which is intuitive as while the freemium follows are real accounts, their follower counts are not independent from each other due to the follows traded between themselves. Figure 10.4 shows the rank-frequency plots for follower counts for various follower types. The plots substantiate our observations on entropy, and also show that different user types exhibit differences with regards to power-law fit, which is expected for skewed distributions on social networks. While entropy values in this paper are computed empirically using the samples from Table 10.2, accounts on real networks have varying follower counts, leading to different entropy estimates even when drawn from the same distribution. Fortunately, we can intimately relate sample size and entropy of a power-law distribution in a closed form using the Euler-Maclaurin formula as below:

Lemma 10.1: Power-Law Entropy

The entropy H of a size $|V|$ sample from a PL distribution $P(r) = C \cdot r^{-a}$ is given by:

$$H \approx -\frac{C \cdot \log_2(C) \cdot (|V|^{1-\alpha} - 1)}{1-\alpha} + \frac{\alpha \cdot C \cdot (-|V|^{1-\alpha} \cdot ((\alpha-1) \cdot \ln(|V|) + 1) + 1)}{(\alpha-1)^2 \cdot \ln(2)}$$

Proof.

$$\begin{aligned}
H &= -\sum_{r=1}^{|V|} Cr^{-\alpha} \cdot \log_2 C \cdot r^{-\alpha} \\
&\approx -\int_1^V C \cdot r^{-\alpha} \cdot \log_2 C dr + \alpha \cdot C \int_1^V r^{-\alpha} \cdot \log_2 r dr \\
&\approx -\frac{C \cdot \log_2 C \cdot r^{1-\alpha}}{1-\alpha}\Big|_1^V + \alpha \cdot C \left(\frac{-r^{1-\alpha}((\alpha-1) \cdot \ln(r) + 1)}{(\alpha-1)^2 \cdot \ln(2)}\Big|_1^V \right) \\
&\approx -\frac{C \cdot \log_2(C) \cdot (|V|^{1-\alpha} - 1)}{1-\alpha} + \\
&\quad \frac{\alpha \cdot C \cdot (-|V|^{1-\alpha} \cdot ((\alpha-1) \cdot \ln(|V|) + 1) + 1)}{(\alpha-1)^2}
\end{aligned} \qquad \blacksquare$$

where $C = 1/H_{|V|,\alpha}$ (inverse of the V^{th} harmonic number of order α).

This estimate enables us to gauge how close varying-sized samples are to the original power-law. This is especially useful for practitioners aiming to gauge what the entropy of an account's followers' attributes theoretically *should be* according to the number of followers assuming a given power-law fit, versus the empirical estimate. If these are not close, one can deduce that the account's followers *do not* obey the expected power-law fit and therefore may be suspicious. This procedure is computationally more efficient and likely more accurate than fitting a separate power-law for each of the attributes across followers of each account.

We noticed similar patterns in entropy for status and favorite counts as well. The lower entropy of action counts characteristic of premium providers stems from the variety of options premium providers have for Twitter engagement – in addition to fake followers, the premium providers also offer fake retweets and favorites services. Thus, premium providers are incentivized to reuse accounts for multiple types of fraud, and when done naïvely result in high synchrony in "serviceable" attributes.

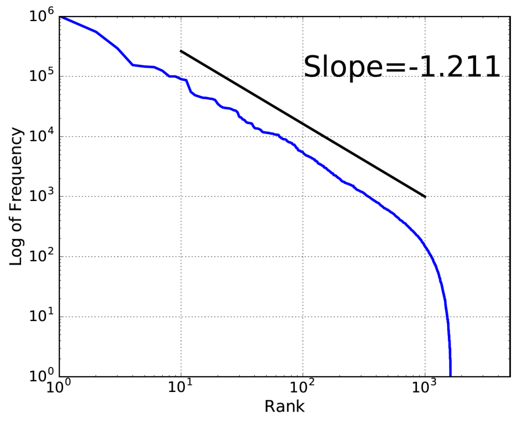
10.3.3.4 User Settings

`fastfollowerz`, `devumi` and `twitterboost` all have near 0 geolocation, language, and tweet protection entropy. Of these, all `devumi` and `twitterboost` accounts use the US English language setting, have geolocation disabled and do not protect tweets. `fastfollowerz` has a slightly higher language entropy of .06, but we found that all `fastfollowerz` accounts were either using US or GB English, suggesting a heavy premium bias for English accounts. We also found that premium followers almost entirely have USA timezones. "Smart" `intertwitter` followers' high language entropy from Figure 10.1d suggests an aim to better camouflage user attributes compard to the "naïve" providers. Given that `intertwitter` also has some verified accounts, we hypothesize that the accounts may be hijacked ones. This is in contrast with freemium providers, which have much higher frequency of enabled geolocation, variance in language and protected tweets. Figure 10.1d also shows that freemium followers tend to appear similar to genuine ones as they are otherwise real user accounts. However, we find that freemium followers have higher language entropy than genuine ones, as freemium followers are spread over many languages whereas genuine followers tend to disproportionately speak their followee's language (i.e. if a user speaks Spanish, most of his followers speak Spanish).

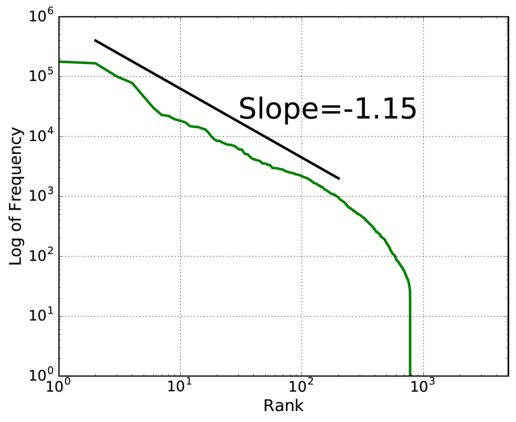
Furthermore, all 4 freemium providers and `twitterboost`/`devumi` have extremely similar attribute entropy over their fake followers respectively, further substantiating Insight 10.5.

In addition to the attributes reported in Table 10.6, we also studied the 160-character user description field. The description field essentially contains the high-level summary of what the user aims to appear as to other Twitter users, and is thus interesting to analyze. We ask: what, if any, are the differences between freemium and premium follower descriptions?

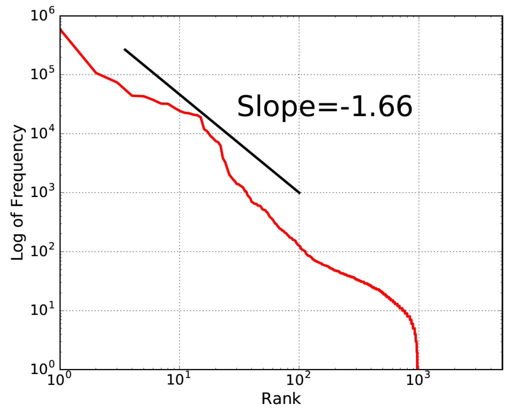
Figure 10.5 shows two wordclouds, aggregated over description text across all premium and freemium followers respectively. Font size corresponds to relative frequency in the text. For clarity, we remove common stopwords. We arrive at the following insight:



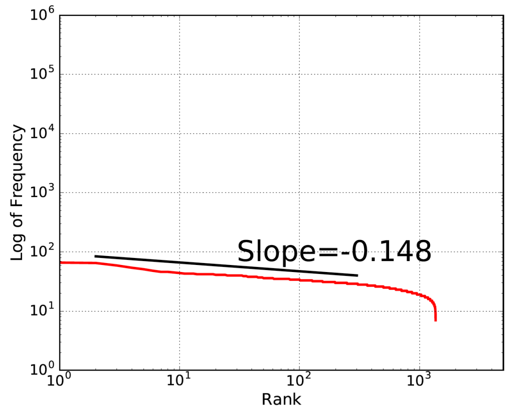
(a) Gen. follow count



(b) Fre. follow count



(c) P1 (“smart”) follow count



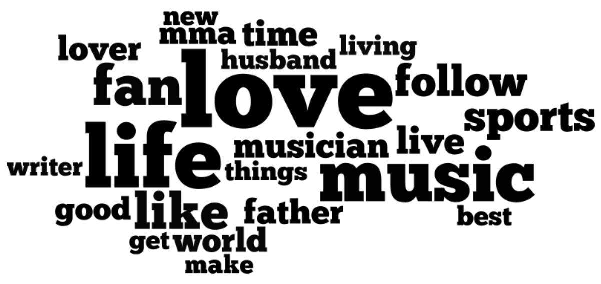
(d) P2 (“naïve”) follow count

Figure 10.4: **Rank-frequency plots reveal different patterns in follower counts of various follower types.** Note that genuine follower counts in (a) reflect traditional power-law behavior with a common exponent (~ 1.2) and are linear in log-log scale. Freemium counts in (b) fit similarly, despite with a slightly lower exponent (~ 1.15). Comparatively, “smart” premium counts in (c) fit a power law but with much higher exponents (~ 1.66). Interestingly, we find that “naïve” premium followers do fit a power law, but have unnaturally low exponents ($\sim .148$) due to their low entropy and highly concentrated, robotic behavior.

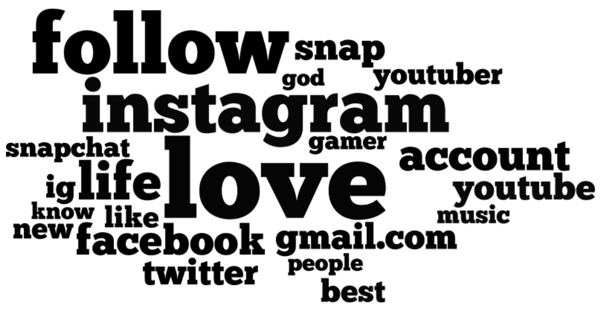
Observation 10.7: Clout vs. About

Freemium followers tend to have descriptions focusing on social media clout, whereas premium followers tend to talk about themselves.

Figure 10.5a (premium), has words like “musician,” “lover,” “writer” and “sports”, corresponding to descriptive personal details – these are likely copied from genuine users. Conversely, Figure 10.5b



(a) Premium



(b) Freemium

Figure 10.5: **Freemium followers have social media (Facebook, Instagram, Snapchat) focused descriptions (right), whereas premium followers have wordy descriptions (left).**

(freemium) has terms like "snapchat," "youtube," and "instagram", as these users try to increase clout by advertising their other, real social media pages, i.e., "*follow me on snapchat.*"

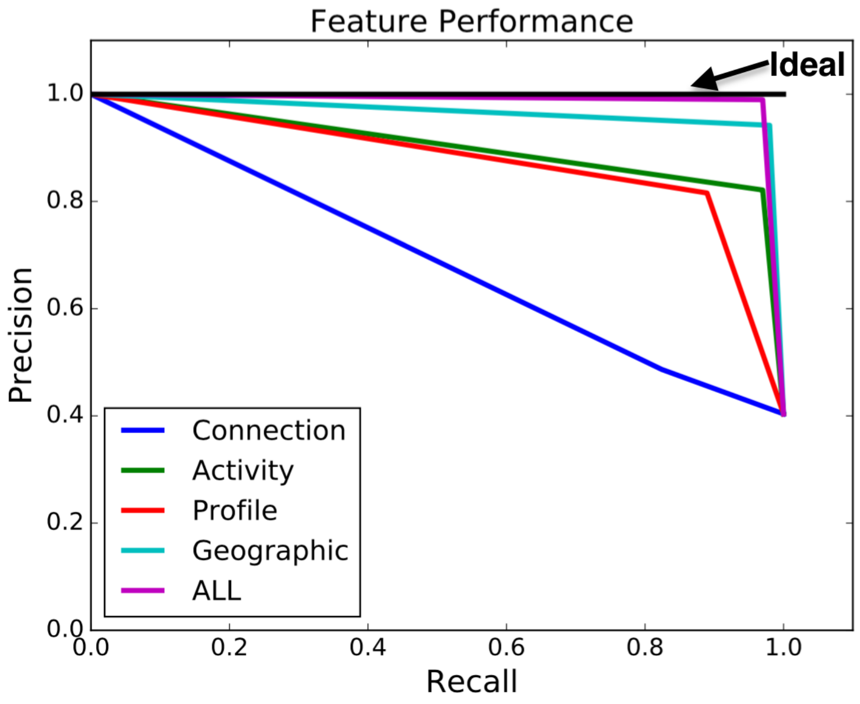


Figure 10.6: **Leveraging all features together gives the best detection performance.**

10.4 Assessing Discriminative Power of Entropy Features

Thus far, we have highlighted a number of distributional differences between fraudulent and genuine users. Can we leverage these differences to discriminate user behaviors? In this section, we evaluate a number of attribute features on their discriminative power in a supervised setting.

We classified the engineered entropy features from Table 10.6 into the following groups based on feature type:

- *Connection*: # Followers, # Friends
- *Activity*: # Statuses, # Lists, # Favorites
- *Profile*: Default Profile (and Image), Verified, Created
- *Geography*: Language, UTC
- *All*: the union of all above features

Note that while we nominally refer to these features as above, they refer to the *entropy of the feature over account followers*, rather than raw values of the account itself.

We evaluate these features using binary classification (genuine vs. fraudulent) as is traditionally done in practice. We use a Support Vector Machine (SVM) with radial basis function (RBF) kernel and 10-fold cross validation as the classifier of choice, but any out-of-box classification method could be used. Our carefully assembled ground-truth dataset consists of 307 fraudulent users and 200 genuine users, whose features are computed over their followers. The fraudulent accounts are a combination of premium and freemium honeypots as well as accounts whose profiles have been listed on freemium providers' websites as users of the service. We define our fraudulent set over this multitude of account types with various properties in order to demonstrate generality. The genuine accounts belong to well-known academics in machine learning and data mining. We avoid using randomly sampled Twitter users, as previous works have shown a non-trivial amount of fake accounts on Twitter which may excessively corrupt our ground-truth genuine set. In practice, getting additional ground-truth labels is a very costly endeavor and requires careful manual inspection for each individual case.

Figure 10.6 shows the relative performance of our feature groups in terms of overall precision and recall. We notice that *Connection* features perform comparatively poorly, *Profile* and *Activity* features perform better, *Geography* performs even better, and the combination *All* performs near-ideal with .98 precision and .95 recall (much higher recall than supervised approaches which use raw account features for Twitter spam classification [MC11]). Thus, we conclude that our proposed entropy features are highly reliable in discerning genuine from fraudulent users. The added benefit of using the entropy-based features is that it is much harder to control for from the fraudster's perspective – this is because while the fraudster has significant control over his own account's properties, he has limited ability to influence who follows him.

10.5 Discussion

The analysis in this work has a number of important implications on fraud detection in practice. We detail these below.

Multimodal Detection: Using individual signatures to find one type of fraud tends to be at the expense of finding other types. For example, clique detection primarily focuses on freemium fraud, whereas bipartite core detection focuses on premium fraud. Using complementary methods is a promising strategy.

Importance of Time: Varying account reuse policies makes temporal granularity an important consideration in graph-based fraud detection. While analysis on a low granularity graph can reveal dense fraudulent structure in frequent reuse regimes, it may never do so for low reuse regimes. Higher granularity can be useful in these cases.

Deceptive Account Attributes: Using individual account attributes to label fraudsters is of limited use. Our work suggests that most freemium fraudsters are actually real users with real profile attributes – they may be resistant to such detection schemes. Conversely, leveraging an account's follower's attributes shows promise in bridging this gap.

Total vs. Partial Fraud: Different types of fraud may call for different penalties. While the implication "has one fake link $\rightarrow$ has all fake links" seems true for premium fraudsters, it is not for freemium ones. Removing fake links vs. suspending fake accounts is a promising way to penalize such fraudsters and minimize false positives.

The need for multimodal anti-fraud mechanisms suggests a shift in the detection paradigm from drawing a two-class boundary between genuine and "one-hat-fits-all" fraudulent users, to a more complex multiclass boundary between genuine, premium fraudulent, freemium fraudulent, and other fraud types which may be discovered in the future.

10.6 Conclusion

In this work, we aimed to study the nature of modern link fraud regimes. To this end, we setup honeypot accounts on Twitter, purchased fake followers for them from a variety of fraud-providing services, and carefully instrumented a data scraping process to capture their behaviors. Specifically, we studied the local network connectivity of fake followers via the egonet and proposed boomerang networks, as well as attribute distributions over profile features and account actions. Our analyses showed that there are multiple types of link fraud (we discover at least two: *freemium* and *premium*) with varying behaviors regarding internal and external network connectivity, disparity in attribute homogeneity across followers, and differences in descriptive word-usage in Twitter bios. Furthermore, we found fascinating evidence that service providers have varying types of account-reuse policies and seem to collude with each other on a number of fronts. Furthermore, we proposed the use of first-order entropy features taken across account followers' attributes to discern fraudulent from genuine accounts, and showed that these features were able to attain near-perfect F1 score on our ground-truth dataset. Holistically, our work offers several implications for practical fraud detection including multimodality of fraud behaviors, the importance of temporally sensitive algorithms, usefulness of follower rather than raw account features, and disadvantages of account-based versus link-based fraud targeting.

Part IV

Conclusions and Future Directions

Chapter 11

Conclusions

In this thesis, we have taken a graph-based approach to anomaly detection in large online social networks. The constituent works target both algorithmic and application-driven problems encountered in practical scenarios. Together, these works advance the state of anomaly detection by building models for, and identifying outliers in interconnectivity patterns, temporal behavior and rich, attributed information. Below, we recapitulate the contributions and impacts of this body of work.

11.1 Contributions

11.1.1 Plain Graphs

In Part I, we focus on discerning abnormal behaviors in node interconnectivity and graph structure.

- **Detecting Stealthy Link-fraud Attacks:** Chapter 3 outlines the theoretical limits of traditional factorization-based link-fraud detection methods, and introduces fBox, a complementary algorithm which detects fraudsters that prior work cannot. We demonstrate effectiveness on Twitter, where fBox identifies *hundreds of thousands* of suspicious users.
- **Cross-Graph Blame Attribution:** Chapter 4 proposes DeltaCon-Attr, a scalable and effective method for pinpointing the culprits responsible for change in connectivity between two snapshots of a graph with known node correspondence. DeltaCon-Attr obeys principles aligning with human intuition and finds qualitatively interesting findings on the ENRON e-mail network.
- **Improved Summarization for Large Graphs:** Chapter 5 improves upon the prior state-of-the-art with ConDeNSe, a graph summarization approach which generates more compact and interpretable summaries which have only *10%* as many structures as other methods and up to *50%* better compression rate.

11.1.2 Dynamic Graphs

In Part II, we further leverage information about time and temporal habits of users to identify anomalous behavior in individuals and groups.

- **Interpretable Dynamic Graph Summarization:** Chapter 6 builds upon similar principles as in Chapter 5, and proposes TimeCrunch, the first general dynamic graph summarization approach capable of finding and labeling a variety of temporal structures. We demonstrate TimeCrunch's effectiveness in pattern extraction on *5* real-world dynamic graphs.
- **Modeling Interarrival Times in Web Searches:** Chapter 7 shows that Poisson assumptions for query times hardly hold in real-world data – in fact, user interarrival times (IATs) are *bimodal*. The chapter proposes the Camel-Log distribution which better models IATs of *78%* of users over the next competitor in a large AOL dataset, and the associated Meta-Click model which is used to find search bots and spammers.
- **Catching Fake Views in Livestreaming Platforms:** Chapter 8 is the first to characterize the problem of fake viewership in livestreaming platforms in literature, and proposes the FLOCK algorithm for combating fake views. We show that FLOCK achieves *98%* precision in detecting astroturfed broadcasts and over *90%* precision and recall in detecting fake views.

11.1.3 Rich Graphs

In Part III, we expand our focus to incorporating rich, attributed information and identifying abnormality in multiple feature spaces.

- **Ranking Anomalies in Edge-Attributed Graphs:** Chapter 9 derives an information theoretically grounded ranking function for the abnormality of nodes based on their adjacent edge attributes, and proposes the EdgeCentric algorithm to compute it. EdgeCentric attains *87%* precision over top-ranked users on Flipkart data.
- **Characterizing the Multimodality of Link Fraud:** Chapter 10 discovers and discusses the multifaceted nature of link fraud on Twitter via honeypots, and proposes entropy features based on follower attributes to discriminate fraudsters from real users.

11.2 Impact

In addition to the above contributions, the work in this thesis offers notable impact on the academic, industrial and code-release fronts which we highlight below.

11.2.1 Academic Impact

- fBox (Chapter 3) was included in the *Multimedia Databases and Data Mining* (15-826) course at Carnegie Mellon University and at a KDD 2015 tutorial on *Graph-Based User Behavior Modeling*.
- TimeCrunch (Chapter 6) was invited to appear in the 2017 Data Engineering Bulletin on Graph Systems.

- TimeCrunch (Chapter 6) was included in the *Mining Large-Scale Graph Data* (EECS 598) course at the University of Michigan and the *Topics in Data Mining* (CS69000-DM1) course at Purdue University. Furthermore, TimeCrunch was featured in the 2016 Carnegie Mellon University CyLab Partners Conference, the 2017 Army Research Lab Network Science panel and an SDM 2017 tutorial on *Summarizing Large-Scale Graph Data: Algorithms, Applications and Open Challenges.*

11.2.2 Industrial Impact

- FLOCK (Chapter 8) is used in *production* at *Twitch.tv* to combat view astroturfing.
- M3A (Chapter 7) is used in *production* at *Google* to identify search spammers and abnormal search behaviors.
- EdgeCentric (Chapter 9) is used in *production* at *Flipkart* to identify ratings fraud.

11.2.3 Code-release Impact

- fBox (Chapter 3) has been open-sourced, and downloaded *24* times since March 2017.
- TimeCrunch (Chapter 6) has been open-sourced, and downloaded *28* times since March 2017.
- EdgeCentric (Chapter 9) has been open-sourced, and downloaded *24* times since March 2017.

Chapter 12

Future Directions

Despite the breadth of work discussed in this thesis, future avenues for anomaly detection are plentiful. Even in the simplest cases, successful anomaly detection requires a sound understanding of relevant data generating processes, concise and accurate models that reflect them, and firm definitions about what constitutes anomalous behavior. In the most difficult scenarios, anomaly detection is an adversarial, cat-and-mouse game which requires continuous work in thwarting increasingly intelligent and sophisticated adversaries who will continue behaving maliciously as long as it remains profitable. There are a number of future directions upon which this thesis has touched upon, but not yet fully developed. The breadth of directions is itself staggering in scope, and requires considerable attention. Below, we outline several key areas which would greatly benefit from further work.

12.1 Adversarial Fraud Detection

In recent years, many works have been proposed for detecting and combating various types of fraudulent behaviors. However, given that comparing and contrasting these works and reproducing their results is challenged by the use of different datasets, limited ground truth, privacy concerns, different data sources leveraged and more, how the spectrum of works complement each other, succeed and fail is poorly understood. Adversarial thinking in the spirit of "what could an adversary do?" can guide us towards answers to these questions, and more: What is the best detection method to use in each scenario, and to what extent should we employ different policies for different types of fraud? Can we engineer attacks that existing methods will not be able to catch, and if so, how can we adapt our algorithms to catch those attacks? How should we set parameters in our algorithms to be sensitive and aware of real-world resource constraints and interests? Answering the first two questions offers promise in autonomously building better detection algorithms, and maintaining a lead on adversaries. The third question is crucial in pushing fraud detection to its utmost potential in industrial practice – we must be aware that the decisions of our algorithms motivate real-world actions, and these actions translate to profits and losses. Carefully tailoring fraud detection algorithms to be aware of this information can guide us to better choices for algorithmic parameters, more advantageous precision/recall tradeoffs for filtering suspicious users while still being amenable to business growth, and generally improved business viability.

12.2 Complex Behavioral Signals

Given the intimate relationship between modeling normal behavior and detecting anomalous behavior, we must strive towards understanding more sophisticated signals which are harder to spoof from the fraudster perspective. The more complex signals we model and look to for judging normalcy, the harder fraudsters have to work to satisfactorily simulate the relevant, "normal" data generating process. In recent years, recommendation systems has worked to incorporate implicit feedback from users such as time spent on pages, scrolling behavior, lingering of mouse movement and more [OK+98]. Analagously, incorporating implicit user behavior for anomaly detection also has exciting prospects. [MR00, KM09] have already shown that keystroke dynamics, or users' timing habits in entering their passwords, is a strong signal for the veracity of authentication attempts. Similar implicit signals such as device canvas fingerprinting [MS12], mobile movement and accelerometer readings, and typographical error habits are all complex and interesting behavioral signals to aim to understand. Emulating such complex signals in order to dupe detection algorithms incurs further cost on fraudsters, making their actions less and less profitable.

12.3 Feature Selection and Extraction

As with most machine learning tasks, performance is fundamentally only as good as the quality of the features used. With ever-growing amounts and types of behavioral data collected, anomaly detection too faces the curse of dimensionality. Anomalies in d-dimensional data could manifest in any of the $2^d - 1$ subspaces. Which features are the most useful for anomaly detection? How can we automatically find these in an efficient way? Exact solutions are combinatorially exhaustive, but finding good-enough approximate solutions and heuristics to guide our algorithms into the correct subspaces is essential. Dimensionality reduction techniques offer promise in this space, and works such as [PSS+10] have shown that reduced representations can lend well to anomaly detection purposes. In recent years, deep learning methods such as autoencoders have gained particular popularity for generating compact, low-dimensional representations of a variety of data sources (text, graphs, etc.) useful for clustering and similarity tasks [PARS14, GL14]. Such tools offer promise for anomaly detection usecases.

12.4 Heterogeneous and Higher-Order Dependency Networks

Traditional network science and graph research has focused on homogeneous networks with uniformly represented nodes and edges. Conversely, heterogeneous information networks offer the capacity to represent rich, multi-typed node and edge interactions such as citation networks (interactions between papers, authors, venues, institutions, etc.) or e-commerce networks (interactions between users, products, sellers, etc.) with varying relation types (*paper-cites-paper*, *author-writes-paper*, *user-rates-product*, *seller-sells-product*, etc.) Previous work on heterogeneous information networks has demonstrated the use of *meta-paths* between nodes of different types for similarity computation [SHY+11], classification [KYDW12], recommendation [LYGS14] and more. Heterogeneous representations and meta-path logic also offers attractive avenues for detecting complex, anomalous interactions between multiple node types.

Traditional networks also make first-order dependency assumptions in which complex interactions which might latently involve several nodes are distilled into pairwise edges in the graph. For example, if Alice sends an e-mail to Bob, who sends that e-mail to Charlie, this information is represented in a traditional network with an edge from Alice to Bob, and Bob to Charlie, missing the higher-order dependency structure present in the underlying interactions. Higher-order networks create conditional versions of each node to account for n^{th}-order dependency structure, and preserve greater information for subsequent analysis. This more accurate representation of underlying interactions lends well to interpretable and more accurate anomaly detection.

12.5 Extensions to Additional Domains

Anomaly detection has broad applications beyond social networks and online graph data. For example, in the education and massive online open courses (MOOC) domain, can we infer when and why a student is struggling, and how he or she can be helped? In healthcare informatics, can we automatically discern fraudulent insurance claims, spot incongruent or odd medical histories, and detect nascent disease outbreaks? In transportation networks, can we automatically detect anomalous passenger routing and infer accidents or road closures? Leveraging domain-specific knowledge and incorporating relevant information into domain-specific anomaly detection algorithms offers promising avenues towards building the best possible solutions for domain practitioners.

Bibliography

[ACF13] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion Fraud Detection in Online Reviews using Network Effects. In *ICWSM*. AAAI, 2013. 155, 156, 168, 169

[ACK$^+$12] Leman Akoglu, Duen Horng Chau, U Kang, Danai Koutra, and Christos Faloutsos. OPAvion: Mining and Visualization in Large Graphs. In K Selçuk Candan, Yi Chen, Richard T Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *SIGMOD Conference*, pages 717–720. ACM, ACM, 2012. 57

[AF02] David Aldous and James Allen Fill. Reversible Markov Chains and Random Walks on Graphs, 2002. 42

[AF10] Leman Akoglu and Christos Faloutsos. Event detection in time series of mobile communication graphs. In *27th Army Science Conference*, 2010. 57, 58

[AGMF14] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. Beyond blocks: Hyperbolic community detection. 2014. 60, 62, 63, 65, 67

[AK15] Anupama Aggarwal and Ponnurangam Kumarguru. What They Do in Shadows: Twitter Underground Follower Market. In *PST*, 2015. 177

[AKY99] Charles J Alpert, Andrew B Kahng, and So-Zen Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90(1):3–26, 1999. 11, 87, 88

[AMF10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining*, pages 410–421. Springer, 2010. 21, 134, 155, 156

[APG$^+$14] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: Fast Automatic Discovery of Temporal ("Comet") Communities. In *PAKDD*, pages 271–283. Springer, 2014. 88, 90

[AR13] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC press, 2013. 10

[ATK14] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph-based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery (DAMI)*, apr 2014. 57

[ATMF12] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. PICS: Parameter-free Identification of Cohesive Subgroups in Large Attributed Graphs. In *SDM*. SIAM, 2012. 156

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 11

[AWO10] Charu C Aggarwal, Haixun Wang, and Others. *Managing and mining graph data*, volume 40. Springer, 2010. 157

[AY05] Charu C Aggarwal and Philip S Yu. Online Analysis of Community Evolution in Data Streams. In *SDM*, 2005. 87

[Bar05] Albert-Laszlo Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207–211, 2005. 128

[BCD$^+$08] Luca Becchetti, Carlos Castillo, Debora Donato, Ricardo Baeza-Yates, and Stefano Leonardi. Link analysis for web spam detection. *ACM Transactions on the Web (TWEB)*, 2(1):2, 2008. 127

[Ber06] Pavel Berkhin. Survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, San Jose, CA, 2006. 142

[BGHS12] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *KDD*. ACM, 2012. 156, 157

[BGHS13] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. RMiCS: a robust approach for mining coherent subgraphs in edge-labeled multi-layer graphs. In *SSDBM*. ACM, 2013. 156, 157

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. 11, 61, 62, 63, 69, 87, 88

[BI07] Judit Bar-Ilan. Position paper: Access to query logs-an academic researcher's point of view. In *Query Log Analysis Workshop, the 16th international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2007. 110

[BJ$^+$06] David M Blei, Michael I Jordan, et al. Variational inference for dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143, 2006. 12

[BKM$^+$08] Lars Backstrom, Ravi Kumar, Cameron Marlow, Jasmine Novak, and Andrew Tomkins. Preferential behavior in online groups. 2008. 62

[BMFS14] Alex Beutel, Kenton Murray, Christos Faloutsos, and Alexander J Smola. CoBaFi - Collaborative Bayesian Filtering. In *WWW*, 2014. 20, 157

[BMRA10] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on twitter. In *CEAS*, 2010. 178

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998. 42

[BXG$^+$13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, pages 119–130. International World Wide Web Conferences Steering Committee, 2013. 20, 25, 57, 133, 134, 141, 156, 157, 175, 177, 178

[CBWG11] Rajmonda Sulo Caceres, Tanya Y Berger-Wolf, and Robert Grossman. Temporal Scale of Processes in Dynamic Networks. pages 925–932, 2011. 40

[CCD$^+$08] Carlos Castillo, Claudio Corsi, Debora Donato, Paolo Ferragina, and Aristides Gionis. Query-log mining for detecting spam. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 17–20. ACM, 2008. 127

[CG10] Yihua Chen and Maya R Gupta. Em demystified: An expectation-maximization tutorial. In *Electrical Engineering*. Citeseer, 2010. 12

[CH94] Diane J. Cook and Lawrence B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. 1, 1994. 61, 87

[CJ12] Zhuhua Cai and Christopher Jermaine. The Latent Community Model for Detecting Sybils in Social Networks. In *NDSS*, Feb 2012. 178

[CKL$^+$09] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On Compressing Social Networks. 2009. 59, 61

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S Modha, and Christos Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88. ACM, 2004. 11, 62, 87, 156

[CSP$^+$14] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J Keogh. Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. *Knowl. Inf. Syst.*, 39(1):123–151, 2014. 127

[CSYP12] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *NSDI*, pages 197–210, 2012. 134

[CT06] Thomas M Cover and Joy A Thomas. *Elements of information theory*. Wiley-Interscience New York, 2006. 66, 91

[CV05] Rudi Cilibrasi and Paul Vitányi. Clustering by Compression. 51(4), 2005. 62

[CYYP14] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *CCS*, pages 477–488. ACM, 2014. 134, 141, 178

[DAFL10] Pedro O S Vaz De Melo, Leman Akoglu, Christos Faloutsos, and Antonio A F Loureiro. Surprising patterns for the call duration distribution of mobile phone users. In *Machine learning and knowledge discovery in databases*, pages 354–369. Springer, 2010. 127, 157

[dbl14] DBLP network dataset. `konect.uni-koblenz.de/networks/dblp_coauthor`, jul 2014. 100

[DDS+04] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, and Others. Adversarial classification. In *SIGKDD*, pages 99–108. ACM, 2004. 21

[DGK05] Inderjit Dhillon, Yuqiang Guan, and Brian Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD*. ACM, 2005. 156

[dMFAL13] Pedro Olmo S de Melo, Christos Faloutsos, Renato Assunção, and Antonio Loureiro. The self-feeding process: a unifying model for communication dynamics in the web. In *WWW*, pages 1319–1330. International World Wide Web Conferences Steering Committee, 2013. 113, 127, 128

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *Proc. 9th KDD*, pages 89–98, 2003. 87

[DPS14] Atish Das Sarma, Nish Parikh, and Neel Sundaresan. E-commerce product search: personalization, diversification, and beyond. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 189–190. International World Wide Web Conferences Steering Committee, 2014. 128

[DS84] Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*. Mathematical Association of America, 1984. 42

[EC06] Shinto Eguchi and John Copas. Interpreting kullback–leibler divergence with the neyman–pearson lemma. *Journal of Multivariate Analysis*, 97(9):2034–2040, 2006. 140

[ER59] Paul Erdös and Alfréd Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959. 26

[EY36] C Eckart and G Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. 12

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. *SIGCOMM*, Aug-Sept. 1999. 60

[FFL+08] Jure Ferlez, Christos Faloutsos, Jure Leskovec, Dunja Mladenic, and Marko Grobelnik. Monitoring Network Evolution using MDL. *ICDE*, 2008. 88

[fli] Flipkart. `http://www.flipkart.com`. 168

[FM07] Christos Faloutsos and Vasilis Megalooikonomou. On Data Mining, Compression and {K}olmogorov Complexity. volume 15. Springer-Verlag, 2007. 62, 64

[FMH93] Wolfgang Fischer and Kathleen Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. *Performance Evaluation*, 18(2):149–171, 1993. 107

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3), 2010. 62

[Fre77] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977. 58

[Fre13] David Mandell Freeman. Using naive bayes to detect spammy names in social networks. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 3–12. ACM, 2013. 133, 134, 175

[Fuk90] Keinosuke Fukunaga. *Introduction to statistical pattern recognition.* Access Online via Elsevier, 1990. 48

[GFB$^+$10] Stephan Gunnemann, Ines Farber, Brigitte Boden, Thomas Seidl, Stephan Günnemann, Ines Farber, Brigitte Boden, and Thomas Seidl. Subspace Clustering Meets Dense Subgraph Mining: A Synthesis of Two Paradigms. In *ICDM*, ICDM '10, Washington, DC, USA, 2010. IEEE, IEEE Computer Society. 156

[GGAH14] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. Outlier Detection for Temporal Data. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 5(1):1–129, 2014. 127

[GHW$^+$10] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y Zhao. Detecting and Characterizing Social Spam Campaigns. In *SIGCOMM*, IMC '10, pages 35–47, New York, NY, USA, 2010. ACM. 177

[GL14] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014. 204

[GLF$^+$10] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. On community outliers and their efficient detection in information networks. In *KDD*. ACM, 2010. 156

[GLK13] Aditi Gupta, Hemank Lamba, and Ponnurangam Kumaraguru. $1.00 per RT #BostonMarathon #PrayForBoston: Analyzing fake content on Twitter. In *Proceedings of the eCrime Researchers Summit (eCRS)*, pages 1–12. IEEE, 2013. 177

[GLKJ13] Aditi Gupta, Hemank Lamba, Ponnurangam Kumaraguru, and Anupam Joshi. Faking Sandy: Characterizing and Identifying Fake Images on Twitter During Hurricane Sandy. In *WWW*, WWW '13 Companion, pages 729–736, New York, NY, USA, 2013. ACM. 177

[GN02] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99, 2002. 62

[Grü05] Peter Grünwald. Minimum Description Length Tutorial. In Peter Grünwald and I J Myung, editors, *Advances in Minimum Description Length*. MIT Press, 2005. 13

[GTPZ10] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @ spam: the underground on 140 characters or less. In *CCS*, pages 27–37. ACM, 2010. 175, 177

[GTV11] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. Evaluating cooperation in communities with the k-core structure. IEEE, 2011. 63

[GVK$^+$12] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Phani Gummadi. Understanding and combating link farming in the twitter social network. In *WWW*. ACM, 2012. 20, 155

[Hav03] Taher H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. 15(4):784–796, 2003. 42

[HEO03] Greg Hamerly, Charles Elkan, and Others. Learning the k in k-means. In *NIPS*, volume 3, pages 281–288, 2003. 12

[HERF$^+$10] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B. Aditya Prakash, and Hanghang Tong. Metric forensics: a multi-level approach for mining volatile graphs. In Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang 0001, editors, *KDD*, pages 163–172. ACM, 2010. 57

[Hes] Joao P Hespanha. An efficient matlab algorithm for graph partitioning. *ECE, UCSB*. 62, 63

[HGK14] William A Hamilton, Oliver Garretson, and Andruid Kerne. Streaming on twitch: fostering participatory communities of play within live mixed media. In *CHI*, pages 1315–1324. ACM, 2014. 134

[HIT86] David C Hoaglin, Boris Iglewicz, and John W Tukey. Performance of some resistant rules for outlier labeling. *Journal of the American Statistical Association*, 81(396):991–999, 1986. 140

[HL04] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *KDD*. ACM, 2004. 157

[HLR07] Xiaojun Hei, Yong Liu, and Keith W Ross. Inferring network-wide quality in P2P live streaming systems. *Selected Areas in Communications, IEEE Journal on*, 25(9):1640–1654, 2007. 134

[HSB$^+$15] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Gunnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, Christos Faloutsos, Stephan Gunneman, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. BIRDNEST: Bayesian Inference for Ratings-Fraud Detection. In *SDM*. SIAM, 2015. 134, 156, 157

[HYJT08] Ling Huang, Donghui Yan, Michael I Jordan, and Nina Taft. Spectral Clustering with Perturbed Data. In *NIPS*, volume 21, 2008. 20

[HZZ$^+$02] Daniel Hanisch, Alexander Zien, Ralf Zimmer, Thomas Lengauer, and Others. Co-clustering of biological networks and gene expression data. In *ISMB*, 2002. 156

[IKIK04] Tsuyoshi Ide, Hisashi Kashima, Tsuyoshi Idé, and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004. 58

[JBC$^+$15] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A General Suspiciousness Metric for Dense Blocks in Multimodal Data. In *ICDM*. IEEE, 2015. 156, 157

[JCB$^+$14a] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catch-sync: catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 941–950. ACM, 2014. 13, 134, 155, 156, 178

[JCB$^+$14b] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. In *Advances in Knowledge Discovery and Data Mining*, pages 126–138. Springer, 2014. 19, 24, 134, 155

[JHA14] Jiepu Jiang, Daqing He, and James Allan. Searching, browsing, and clicking in a search session: changes in user behavior by task and over time. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 607–616. ACM, 2014. 128

[JL08] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *WSDM*. ACM, 2008. 157

[JST$^+$17] Da-Cheng Juan, Neil Shah, Mingyu Tang, Zhiliang Qian, Diana Marculescu, and Christos Faloutsos. M3a: Model, metamodel, and anomaly detection in web searches. In *IEEE DSAA*, 2017. 107

[JWP$^+$05] Ruoming Jin, Chao Wang, Dmitrii Polshakov, Srinivasan Parthasarathy, and Gagan Agrawal. Discovering frequent topological structures from graph datasets. In *KDD*, pages 606–611, 2005. 87

[Kar] Efe Karakus. *A Look Into Streaming*. 135

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8), 2009. 157

[KC10] Nguyen Lu Dang Khoa and Sanjay Chawla. Robust Outlier Detection Using Commute Time and Eigenspace Embedding. volume 6119 of *Lecture Notes in Computer Science*. Springer, 2010. 57

[KF11a] U. Kang and Christos Faloutsos. Beyond ‘Caveman Communities’: Hubs and Spokes for Graph Compression and Mining. 2011. 61, 62, 69, 88

[KF11b] U Kang and Christos Faloutsos. Beyond’caveman communities’: Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309. IEEE, 2011.

[KG10] B Kulis and Y Guan. Graclus - Efficient graph clustering software for normalized cut and ratio association on undirected graphs, 2008. 2010. 87, 88

[KK95] George Karypis and Vipin Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system. 1995. 45, 155, 156

[KK99] George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. In *DAC*, pages 343–348, 1999. 61, 62, 63, 69

[KK00] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000. 11, 87, 88

[KKK$^+$11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. 2011. 42, 46, 54, 58, 70, 97

[KKPF13] Danai Koutra, Vasileios Koutras, B Aditya Prakash, and Christos Faloutsos. Patterns amongst Competing Task Frequencies: Super-Linearities, and the Almond-DG model. In *Advances in Knowledge Discovery and Data Mining*, pages 201–212. Springer, 2013. 125

[KKR$^+$99] Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *Computing and combinatorics*, pages 1–17. Springer, 1999. 90

[KKVF14a] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and Understanding Large Graphs. 2014. xvi, 60, 61, 62, 67, 71, 74, 75, 76, 77

[KKVF14b] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and understanding large graphs. In *SDM*. SIAM, SIAM, 2014. 88, 156

[kle99a] The Web as a Graph: Measurements, Models and Methods. In *COCOON*, 1999. 60, 65

[Kle99b] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999. 58

[KLKF14] U Kang, Jay-Yoon Lee, Danai Koutra, and Christos Faloutsos. Net-Ray: Visualizing and Mining Web-Scale Graphs. 2014. 57

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW*. ACM, 2010. 18, 28, 34

[KM09] Kevin S Killourhy and Roy A Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 125–134. IEEE, 2009. 204

[KM13] Trupti M Kodinariya and Prashant R Makwana. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013. 11

[KMPF14] U Kang, Brendan Meeder, Evangelos E Papalexakis, and Christos Faloutsos. HEigen: Spectral Analysis for Billion-Scale Graphs. *TKDE*, 26(2):350–362, 2014. 20

[KPF12] Danai Koutra, Evangelos E Papalexakis, and Christos Faloutsos. Tensorsplat: Spotting latent anomalies in time. In *Informatics (PCI), 2012 16th Panhellenic Conference on*, pages 144–149. IEEE, 2012. 57

[KSC$^+$12] Mehdi Kaytoue, Arlei Silva, Loic Cerf, Meira Jr Wagner, and Chedy Raissi. Watch me playing, i am a professional: a first study on video game live streaming. In *WWW*, pages 1181–1188. ACM, 2012. 134

[KSV$^+$15] Danai Koutra, Neil Shah, Joshua Vogelstein, Brian Gallagher, and Christos Faloutsos. DeltaCon: A Principled Massive-Graph Similarity Function with Attribution. *ACM Transactions on Knowledge Discovery from Data*, 2015. 39, 40, 43, 46

[Kuh04] Jouni Kuha. Aic and bic: Comparisons of assumptions and performance. *Sociological methods & research*, 33(2):188–229, 2004. 13

[KVF$^+$13] Danai Koutra, Joshua T Vogelstein, Christos Faloutsos, Danai Koutra, and Joshua T Vogelstein. DELTACON: A Principled Massive-Graph Similarity Function. In *CoRR*, volume abs/1304.4657, pages 162–170. SIAM, 2013. 40

[KY04] Bryan Klimt and Yiming Yang. Introducing the Enron Corpus. 2004. 56

[KYDW12] Xiangnan Kong, Philip S Yu, Ying Ding, and David J Wild. Meta path-based collective classification in heterogeneous information networks. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1567–1571. ACM, 2012. 205

[LCW10] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering Social Spammers: Social Honeypots + Machine Learning. In *SIGIR*, SIGIR '10, pages 435–442, New York, NY, USA, 2010. ACM. 178

[LHK10] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *SIGCHI*, pages 1361–1370. ACM, 2010. 28

[LJL$^+$06] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M Ni, and Dafu Deng. AnySee: Peer-to-Peer Live Streaming. In *INFOCOM*, volume 25, pages 1–10. Citeseer, 2006. 134

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, jun 2014. 74

[LKKF13] Jay Yoon Lee, U Kang, Danai Koutra, and Christos Faloutsos. Fast anomaly detection despite the duplicates. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 195–196. International World Wide Web Conferences Steering Committee, 2013. 57

[LLM10] Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical Comparison of Algorithms for Network Community Detection. New York, NY, USA, 2010. ACM, ACM. 62

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. 11

[LM05] Daniel Lowd and Christopher Meek. Adversarial learning. In *SIGKDD*, pages 641–647. ACM, 2005. 21

[LSK15] Yike Liu, Neil Shah, and Danai Koutra. An Empirical Comparison of the Summarization Power of Graph Clustering Methods. In *NIPS NSIS Workshop 2015*, 2015. 59, 62, 63

[LSSK] Yike Liu, Tara Safavi, Neil Shah, and Danai Koutra. Reducing Million-Node Graphs to a Few Structural Patterns: A Unified Approach. In *KDD MLG Workshop 2016*. 59

[LT10] Kristen LeFevre and Evimaria Terzi. Grass: Graph structure summarization. In *SDM*. SIAM, 2010. 61

[LVV90] M Li, P Vitanyi, and P M B Vitányi. *An Introduction to Kolmogorov complexity and its applications*. Springer-Verlag, New York, 2nd edition, 1990. 89

[LWD10] Chao Liu, Ryen W White, and Susan Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 379–386. ACM, 2010. 128

[LYGS14] Xiaozhong Liu, Yingying Yu, Chun Guo, and Yizhou Sun. Meta-path-based ranking with pseudo relevance feedback on heterogeneous graph for citation recommendation. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 121–130. ACM, 2014. 205

[LZSJ$^+$08] Shao Liu, Rui Zhang-Shen, Wenjie Jiang, Jennifer Rexford, and Mung Chiang. Performance bounds for peer-assisted live streaming. In *ACM SIGMETRICS Performance Evaluation Review*, volume 36, pages 313–324. ACM, 2008. 134

[LZWY06] Bo Long, Zhongfei Mark Zhang, Xiaoyun Wu, and Philip S Yu. Spectral clustering for multi-type relational data. In *ICML*. ACM, 2006. 156, 157

[MBY16] Rishabh Mehrotra, Prasanta Bhattacharya, and Emine Yilmaz. Uncovering Task Based Behavioral Heterogeneities in Online Search Behavior. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1049–1052. ACM, 2016. 128

[MC11] Michael Mccord and M Chuah. Spam detection on twitter using traditional classifiers. In *ICATC*, pages 175–186. Springer, 2011. 194

[MGF11] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *ASONAM*, pages 203–210. IEEE, 2011. 57

[ML13] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*, pages 165–172. ACM, 2013. 28, 35

[MLC07] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, 2007. 127, 128

[MLK$^+$10] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Re: CAPTCHAs: Understanding CAPTCHA-solving Services in an Economic Context. In *USENIX Security*, USENIX Security'10, page 28, Berkeley, CA, USA, 2010. USENIX Association. 177

[MMV05] Mark Meiss, Filippo Menczer, and Alessandro Vespignani. On the lack of typical behavior in the global Web traffic network. In *Proceedings of the 14th international conference on World Wide Web*, pages 510–518. ACM, 2005. 127, 128

[MPL15] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring Networks of Substitutable and Complementary Products. In *KDD*. ACM, 2015. 168

[MR00] Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4):351–359, 2000. 204

[MS12] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, pages 1–12, 2012. 204

[MV13] Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013. 11

[MWP$^+$14] Ching-Hao Mao, Chung-Jung Wu, Evangelos E Papalexakis, Christos Faloutsos, and Tien-Cheu Kao. MalSpot: Multi2 Malicious Network Behavior Patterns Analysis. In *PAKDD*, 2014. 20, 57, 133

[Nan10] Volker Nannen. A short introduction to model selection, kolmogorov complexity and minimum description length (mdl). *arXiv preprint arXiv:1005.2364*, 2010. 13

[NC03] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *KDD*. ACM, 2003. 156

[Net06] Netflix. Netflix Competition. 2006. 28

[New05] Mark E J Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1):39–54, 2005. 58

[NG04] Mark E J Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):26113, 2004. 11, 87, 88

[NJW01] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. *NIPS*, 14:849–856, 2001. 20

[Nob03] Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003. In Lise Getoor, Ted E Senator, Pedro Domingos, and Christos Faloutsos, editors, *KDD*. ACM, 2003. 40

[NPR] NPR. Top Reviewers on Amazon Get Tons of Free Stuff. \url{http://www.npr.org/sections/money/2013/10/29/241372607/top-reviewers-on-amazon-get-tons-of-free-stuff}. 173

[NRS08] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph Summarization with Bounded Error. 2008. 59, 60, 61

[OK$^+$98] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83. Menlo Park, CA: AAAI Press, 1998. 204

[PAIM14] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *KDD*. ACM, 2014. 156

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014. 11, 204

[PCT06] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *InfoScale*, volume 152, page 1. Citeseer, 2006. 109, 110

[PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*, pages 201–210. ACM, 2007. 20, 134, 155, 156

[Per13] Nicole Perloth. Fake Twitter Followers Become Multimillion-Dollar Business, apr 2013. 177

[PJZ05] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005. 88

[PMO00] Dan Pelleg, Andrew W Moore, and Others. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *ICML*, volume 1, 2000. 12, 142, 168

[PSB13] Evangelos E Papalexakis, Nicholas D Sidiropoulos, and Rasmus Bro. From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE TSP*, 61(2):493–506, 2013. 98

[PSS$^+$10] B A Prakash, M Seshadri, A Sridharan, S Machiraju, and C Faloutsos. Eigenspokes: Surprising patterns and community structure in large graphs. *PAKDD*, 84, 2010. 11, 13, 19, 60, 134, 141, 155, 156, 175, 178, 204

[PWWB09] Rhonda D Phillips, Layne T Watson, Randolph H Wynne, and Christine E Blinn. Feature reduction using a singular value decomposition for the iterative guided spectral class rejection hybrid classifier. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(1):107–116, 2009. 12

[Ras00] Carl Edward Rasmussen. The infinite gaussian mixture model. In *Advances in neural information processing systems*, pages 554–560, 2000. 12

[RGM03] Sriram Raghavan and Hector Garcia-Molina. Representing web graphs. IEEE, 2003. 61

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. 13, 91

[Ris83] Jorma Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. 11(2), 1983. 62, 65, 67, 68

[RLG$^+$10] Chotirat Ann Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn J Keogh, Michail Vlachos, and Gautam Das. Mining Time Series Data. In *Data Mining and Knowledge Discovery Handbook*, pages 1049–1077. 2010. 127

[RSK$^+$15] Stephen Ranshous, Shitian Shen, Danai Koutra, Steven Harenberg, Christos Faloutsos, and Nagiza F Samatova. Graph-based Anomaly Detection and Description: A Survey. *WIREs Computational Statistics*, jan 2015. 57

[SA04] Jitesh Shetty and Jafar Adibi. The Enron email dataset database schema and brief statistical report. *Inf. sciences inst. TR, USC*, 4, 2004. 100

[SBGF14] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*, pages 959–964. IEEE, 2014. v, 17, 87, 88, 90, 133, 134, 155, 156, 175, 178

[SBH$^+$16] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnemann, Disha Makhija, Mohit Kumar, and Christos Faloutsos. Edgecentric: Anomaly detection in edge-attributed networks. In *ICDMW*, 2016. v, 134, 153

[Scu10] David Sculley. Web-scale k-means clustering. In *WWW*, pages 1177–1178. ACM, 2010. 11, 142

[SD14] Kumar Sricharan and Kamalika Das. Localizing Anomalous Changes in Time-evolving Graphs. ACM, 2014. 50, 54, 56, 57, 58

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696. ACM, 2007. 87, 88

[SFŚ$^+$04] Krzysztof Simek, Krzysztof Fujarewicz, Andrzej Świerniak, Marek Kimmel, Barbara Jarząb, Małgorzata Wiench, and Joanna Rzeszowska. Using svd and svm methods for selection, classification, clustering and modeling of dna microarray data. *Engineering Applications of Artificial Intelligence*, 17(4):417–427, 2004. 12

[SG11] Martin Saveski and Miha Grčar. Web Services for Stream Mining: A Stream-Based Active Learning Use Case. *ECML PKDD 2011*, page 36, 2011. 128

[SH12] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 3(2):1–159, 2012. 157

[Sha17] Neil Shah. FLOCK: Combating Astroturfing on Livestreaming Platforms. In *WWW*. International World Wide Web Conferences Steering Committee, 2017. v, 131

[SHAC12] Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. When will it happen?: relationship prediction in heterogeneous information networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 663–672. ACM, 2012. 157

[SHY$^+$11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011. 157, 205

[SHZ$^+$09] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng, and Tianyi Wu. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 565–576. ACM, 2009. 157

[SKV10] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting Spammers on Social Networks. In *ACSAC*, ACSAC '10, pages 1–9, New York, NY, USA, 2010. ACM. 178

[SKZ$^+$15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. TimeCrunch: Interpretable Dynamic Graph Summarization. In *KDD*. ACM, 2015. 59, 61, 62, 85, 156

[SLBF17] Neil Shah, Hemank Lamba, Alex Beutel, and Christos Faloutsos. Oec: Open-ended classification for future-proof link-fraud detection. *arXiv preprint arXiv:1704.01420*, 2017. 175

[SLZ$^+$17] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2017. 157

[SMO$^+$03] P Shannon, A Markiel, O Ozier, N S Baliga, J T Wang, D Ramage, N Amin, B Schwikowski, and T Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498, 2003. 74

[SMR08] Nisheeth Shrivastava, Anirban Majumder, and Rajeev Rastogi. Mining (social) network graphs to detect random link attacks. In *ICDE*. IEEE, 2008. 20

[SMZ04] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. An analysis of live streaming workloads on the internet. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 41–54. ACM, 2004. 134

[SPT13] Bai-En Shie, S Yu Philip, and Vincent S Tseng. Mining interesting user behavior patterns in mobile commerce environments. *Applied intelligence*, 38(3):418–435, 2013. 128

[SS11] Berthold Schweizer and Abe Sklar. *Probabilistic metric spaces.* Courier Dover Publications, 2011. 124

[SWE+13] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng, and Y Ben Zhao. Follow the Green: Growth and Dynamics in Twitter Follower Markets. In *SIGMETRICS*, 2013. 177

[TGSP11] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258. ACM, 2011. 21

[TIB+14] Kurt Thomas, Dmytro Iatskiv, Elie Bursztein, Tadek Pietraszek, Chris Grier, and Damon McCoy. Dialing Back Abuse on Phone Verified Accounts. In *CCS*, CCS '14, pages 465–476, New York, NY, USA, 2014. ACM. 177

[TL11] Hanghang Tong and Ching-Yung Lin. Non-Negative Residual Matrix Factorization with Application to Graph Anomaly Detection. In *SDM*. SIAM, 2011. 155

[TMG+13] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. In *USENIX Security*, SEC'13, pages 195–210, Berkeley, CA, USA, 2013. USENIX Association. 177

[TPER+12] Hanghang Tong, B Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 245–254. ACM, 2012. 58

[TZHH11] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *KDD*, pages 965–973. ACM, 2011. 88

[VAM+02] Eveline Veloso, Virgilio Almeida, Wagner Meira, Azer Bestavros, and Shudong Jin. A hierarchical characterization of a live streaming media workload. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 117–130. ACM, 2002. 134

[VMJS14] Carmen K Vaca, Amin Mantrach, Alejandro Jaimes, and Marco Saerens. A time-based collective factorization for topic discovery and monitoring in news. In *Proceedings of the 23rd international conference on World wide web*, pages 527–538. International World Wide Web Conferences Steering Committee, 2014. 127

[WLR09] Di Wu, Yong Liu, and Keith W Ross. Queuing network models for multi-channel P2P live streaming systems. In *INFOCOM 2009, IEEE*, pages 73–81. IEEE, 2009. 134

[WPK$^+$17] Serene WH Wong, Chiara Pastrello, Max Kotlyar, Christos Faloutsos, and Igor Jurisica. Modeling tumor progression via the comparison of stage-specific graphs. *Methods*, 2017. 5

[WPT11] Ye Wang, Srinivasan Parthasarathy, and Shirish Tatikonda. Locality Sensitive Outlier Detection: A ranking driven approach. pages 410–421, 2011. 40

[WS05] Scott White and Padhraic Smyth. A Spectral Clustering Approach To Finding Communities in Graph. In *SDM*. SIAM, 2005. 155

[WTPP14] Heng Wang, Minh Tang, Y. Park, and C.E. Priebe. Locality statistics for anomaly detection in time series of graphs. *IEEE Transactions on Signal Processing*, 62(3):703–717, Feb 2014. 40

[WWZ$^+$12] Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y Zhao. Serf and Turf: Crowdturfing for Fun and Profit. In *WWW*, WWW '12, pages 679–688, New York, NY, USA, 2012. ACM. 177

[XFH15] Cao Xiao, David Mandell Freeman, and Theodore Hwa. Detecting Clusters of Fake Accounts in Online Social Networks. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 91–101. ACM, 2015. 133

[XKH11] Kevin S Xu, Mark Kliger, and Alfred O Hero III. Tracking communities in dynamic social networks. In *SBP*, pages 219–226. Springer, 2011. 88

[XPYW11] Zhengzheng Xing, Jian Pei, Philip S Yu, and Ke Wang. Extracting Interpretable Features for Early Classification on Time Series. In *SDM*, pages 247–258, 2011. 127

[XZC01] Guorong Xuan, Wei Zhang, and Peiqi Chai. Em algorithms of gaussian mixture model and hidden markov model. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 1, pages 145–148. IEEE, 2001. 12

[YA15] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 2015. 155, 156

[Yah] Yahoo! Webscope. `webscope.sandbox.yahoo.com`. 100

[YCSH12] Yang Yang, Nitesh Chawla, Yizhou Sun, and Jiawei Hani. Predicting links in multi-relational and heterogeneous networks. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 755–764. IEEE, 2012. 157

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 36(4):267–278, 2006. 134, 178

[YL13] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, New York, NY, USA, 2013. ACM, ACM. 62, 63, 69

[YWB11] Xiaowei Ying, Xintao Wu, and Daniel Barbará. Spectrum based fraud detection in social networks. In *ICDE*, pages 912–923. IEEE, 2011. 19

[ZCY09] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *VLDB*, 2009. 157