Mining and Modeling Real-world Networks: Patterns, Anomalies, and Tools

Leman Akoglu

CMU-CS-12-136

August, 2012

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christos Faloutsos, Chair
Andrew Moore
Aarti Singh
Andrew Tomkins, Google Inc.

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-0534205, IIS-0705359, and IIS-0808661; Lehigh University under grant number C000022761; Lawrence Livermore National Laboratory under grant number B573265; the U.S. Army Research Laboratory (UIUC) under grant number W911NF-09-2-0053; the Defense Threat Reduction Agency (UIUC) under grant number HDTRA1-10-1-0120; and ICAST. 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: Data mining, knowledge discovery, graph mining, network analysis, information theory, graph structure and dynamics, pattern discovery, graph generators, clustering, anomaly and event detection, compression, sensemaking, summarization

Dedicated to my dearest parents, for their endless love and support.

Abstract

Large real-world graph (a.k.a network, relational) data are omnipresent, in online media, businesses, science, and the government. Analysis of these massive graphs is crucial, in order to extract descriptive and predictive knowledge with many commercial, medical, and environmental applications. In addition to its general structure, knowing what stands out, i.e. anomalous or novel, in the data is often at least, or even more important and interesting.

In this thesis, we build novel algorithms and tools for mining and modeling large-scale graphs, with a focus on: (1) *Graph pattern mining*: we discover surprising patterns that hold across diverse real-world graphs, such as the "fortification effect" (e.g. the more donors a candidate has, the super-linearly more money s/he will raise), dynamics of connected components over time, and power-laws in human communications, (2) *Graph modeling*: we build generative mathematical models, such as the RTG model based on "random typing" that successfully mimics a long list of properties that real graphs exhibit, (3) *Graph anomaly detection*: we develop a suite of algorithms to spot abnormalities in various conditions; for (a) plain weighted graphs, (b) binary and categorical attributed graphs, (c) time-evolving graphs, and (d) sensemaking and visualization of anomalies.

Acknowledgments

There is a long list of truly inspiring individuals that made the completion of this thesis possible and very rewarding for me.

I am extremely fortunate to have my PhD advisor Christos Faloutsos, who has been a true source of driving force, encouragement, learning, and support. I wholeheartedly thank Christos for inspiring me towards studying networks, for the fruitful research discussions, for all the time he was able to fit into his schedule when I needed, and for many other inspiring things I cannot find enough space to list here. I am also grateful to my thesis committee members Andrew W. Moore, Aarti Singh, and Andrew Tomkins for their time and effort in providing me with invaluable feedback in putting together and improving my thesis.

A large body of my thesis would not have been possible without the wonderful list of collaborators I had: Duen Horng Chau, Nan Du, Tina Eliassi-Rad, Brian Gallagher, Keith Henderson, U Kang, Lei Li, Mary McGlohon, Brendan Meeder, Pedro Olmo Vaz de Melo, Hanghang Tong, and Jilles Vreeken. I am also thankful to all my peers and friends in the Database Lab at CMU for fun research meetings, their feedback on papers and talks, and good times spent in Pittsburgh, including: Ramnath Balasubramanyan, Alex Beutel, Jos P. Gonzlez-Brenes, Rishi Chandy, Bhavana Dalvi, Fan Guo, Danai Koutra, Ni Lao, Frank Lin, Ippokratis Pandis, Vagelis Papalexakis, B. Aditya Prakash, Kriti Puniyani. Last but not least, I thank my closest friend at CMU: Sivaraman Balakrishnan, for being a true best friend, source of moral support, and joy in my life.

Several internships and visits at IBM T. J. Watson Research Labs gave me the opportunity to work closely with top researchers in industry including Rohit Khandekar, Ravi B. Konuru, Vibhore Kumar, Srinivasan Parthasarthy, Deepak Rajan, Jimeng Sun, Hanghang Tong, and Kun-Lung Wu. I am particularly indebted to my mentors Rohit Khandekar and Hanghang Tong, for opening new ways of thinking in my mind, showing me how to approach a problem and how to precisely formulate and write it down, as well as teaching me how to keep my eyes open to good subproblems hidden in the big ones.

I am honored to complete my Ph.D. studies at Carnegie Mellon University for the unique opportunities and a wonderful research environment it offers. I am grateful to many professors from whom I have learned invaluable new knowledge and numerous skills, including: William W. Cohen, Carlos Guestrin, Karl Crary, Gary Miller, Onur Mutlu, Daniel Neill, and Kathryn Roeder. I also thank Marilyn Walgora and Charlotte Yano for helping life run smoothly at CMU, seamlessly handling all administrative work.

Before coming to CMU, I was introduced to the joys of research and data mining at Bilkent University. I cannot thank enough my senior thesis advisor Ozgur Ulusoy and my mentor Sengor Altingovde for opening my eyes to this uniquely rewarding experience, and guiding me in my very early pursuits as a researcher.

Most of all, I am grateful to my family. My mother Ayse Akoglu; who has been so selfless in supporting me at all stages of my life and career, my father Fevzi Akoglu; who has been a source of encouragement and influence in the direction of my career and my lovely sister Leyla Akoglu; who has inspired me with her hardworking, courageous, enthusiastic, and always cheerful character in life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Applications

In this thesis, our focus is *mining patterns and anomalies in large, time-varying, real-world graphs using scalable algorithms and tools*. Graphs provide a powerful machinery for modeling many types of relations for both natural and human-made structures in physical, biological, and social systems. As a result, graph data has become ubiquitous –Internet, social networks, food web, protein networks and many more. These real-world graphs have posed a wealth of fascinating research questions and high-impact applications. For example, on-line social networks have been generating billions of dollars of revenues; anomaly detection in terrorism networks as well as computer networks is vital for national security; blogs have become an important medium of spreading information and ideas on World Wide Web; brains networks and gene regulatory networks can help us understand how our brains and cells work.

Graphs also serve as powerful tools for solving real-world problems. In other words, many problems of practical interest can be represented as natural problems on graphs. In history, the well-known "Seven Bridges of Koenigsberg" problem is the first real-world problem that was solved by the study of graphs. Introduced by Leonhard Euler in 1735, the problem asks for a walk through the city that would cross each bridge once and only once. Representing land as nodes and bridges as edges, Euler reformulated and analyzed the problem in abstract terms, laying the foundations of graph theory.

Since Euler, researchers have cast a tremendous number of real-world problems in diverse domains as graph problems. For example, graphs are widely used in sociology as a way to explore the spread of information or diseases among individuals. In biology graphs are used to represent regions of species and their migration paths to study breeding patterns. Graph-centric methods are also useful for the study of molecules in chemistry, with applications like chemical similarity and largest common substructure search. Other applications include community detection, peer recommendation, ad targeting, hijack and phishing detection, etc. In summary, graphs have proven to be a convenient abstraction to reason about numerous problems, and graph algorithms

are at the heart of various problems in many diverse fields.

As graphs are among the most ubiquitous models for real-world data, they rapidly grow in size as more and more data become available. In 1735, the graph that Euler studied had only four nodes and seven edges. Today, social graphs like Facebook and Twitter, by astounding growth in the past decade, have reached hundreds of millions of users around the world. The World Wide Web contains at least nine billion pages. Through impressive technical advances, biologists are gathering more and more genomic data at lower costs. These huge amounts of real-world data present even higher impact opportunities. With this new phenomenon, one of the biggest challenges lies in interpreting the huge volume of data being generated and in extracting descriptive, predictive, and commercially or medically useful information from it.

Under graph mining, our interests group into three interrelated topics:

1. *Graph patterns*: We focus on identifying regularities that hold across real-world graphs, and understanding their formation and evolution. Such patterns prove to be useful for various data mining tasks such as (1) modeling; by providing intuition into the mechanisms by which networks form and evolve, (2) summarization; by providing a compact representation, (3) forecasting; by representing continuing trends, and (4) anomaly detection; by revealing data instances that deviate significantly from the observed trends.
2. *Graph generators*: We focus on developing generative models that produce synthetic graphs that can mimic real graphs. Generators are useful in providing a laboratory environment for generating synthetic graphs with certain size and characteristics, especially for simulation studies.
3. *Anomaly detection*: Our goal is to build models that capture the norms in graph data and later to exploit these models to detect and characterize anomalies and events that significantly deviate from normal behavior. This is a crucial task, with numerous applications in finance, security, health care, law enforcement, and so on.

Next, we give the thesis statement and an overview of how the thesis is organized. We also list the problems we address and provide a summary of our contributions.

1.2 Thesis Statement and Overview

Real-world networks exhibit regularities, which then help reveal anomalies. More specifically, in this thesis we investigate the regularities (patterns) in the formation, structure, and evolution of real-world networks, develop generative models (generators) that explain and capture these regularities and that mimic real-world networks. We then build new methods to spot irregularities (anomalies and events) in data, exploiting these regularities in addition to using several novel graph mining and compression techniques.

Following the thesis statement, we organize the thesis into three parts. The outline is shown in Table 1.1.

Observations	
Part I: Graph Patterns	• Chapter 3 Patterns in graph topology [Akoglu et al., 2008; Kang et al., 2010a; Mcglohon et al., 2008] • Chapter 4 Patterns in human communications [Akoglu et al., 2012a; Du et al., 2009; Vaz de Melo et al., 2010].
Models	
Part II: Graph Generators	• Chapter 6 Models of graph topology [Akoglu and Faloutsos, 2009; Akoglu et al., 2008] • Chapter 7 Models of human communications [Du et al., 2009].
Algorithms	
Part III: Anomaly Detection	• Chapter 9 Anomalies in plain graphs [Akoglu et al., 2010]. • Chapters 10,11 Anomalies in attributed graphs [Akoglu et al., 2012b,c]. • Chapter 12 Events in time-evolving graphs [Akoglu and Faloutsos, 2010]. • Chapter 13 Sensemaking for graph anomalies [Akoglu et al., 2012d].

Table 1.1: Outline of thesis with references to chapters.

In the first part, **Patterns of Networks**, we focus on the structure and dynamics of real-world graphs. We are interested in understanding how new nodes and edges arrive to the network and the structure they generate at large, how they form components and how these components grow and merge over time, and the regularities associated with weights on edges. Another focus of this part is the study of human communications, where we analyze the social circles humans belong to, how often they reciprocate, and their calling behavior. The questions we address are:

> What are the typical patterns in the structure and dynamics of real-world networks from a large collection of diverse domains (e.g. political campaign donations, computer network traffic, online social network, large Web graph)? (Chapter 3)
>
> What are the typical patterns in human communications? How large are our social circles? How reciprocal are we, and what does it depend on? How long are our phone calls, and how can we summarize them? (Chapter 4)

Impact

We are among the first to discover patterns in weighted graphs and in the connected components of graphs.

- **"Rebel probability" and oscillating/constant-size connected components:** We discover that the smaller connected components grow up to a certain size at which they merge into the largest component (§3.1.3), and that newcomers *rebel* this largest component with probability that drops exponentially with their degree (§3.1.5).

- **"Fortification effect" and other non-linear weighted-network patterns:** Our findings include the surprising law of *fortification* (§3.2.2), which formulates a power law relation between edges and weights in real networks.

- **Power-laws in human communications:** We contribute the first model (§4.2.3) that succinctly captures the reciprocity behavior in human communications, and a new model (§4.3.2) that summarizes call duration behavior of millions of humans. Both models consistently provide the *best* fit to real data, among popular competitors (Pareto, Lognormal, Yule).

In the second part, **Generative Models of Networks**, we create models that produce synthetic but realistic graphs. First, we develop models that mimic general real-world graphs. Second, we focus particularly on human communication behavior and build a model to mimic this type of behavior. The main questions of interest are:

> How could we have a realistic generative model that will produce synthetic graphs that look like real, i.e. graphs that obey all the patterns we know so far, as well as the newly discovered ones for dynamic and weighted graphs? (Chapter 6)
>
> How could we design a realistic and intuitive generative model that will naturally reproduce real human-to-human communication behavior? (Chapter 7)

Impact

We are among the first to propose a graph generator for weighted graphs that also captures all other related properties.

- **Realistic generative models for graphs:** Our first model for general graphs, the Recursive Tensor Model (§6.1), provably models certain graph laws and is the *first* to model bursty traffic. Our later model, the Random Typing Graphs (RTG) (§6.2), unlike previous models, mimics *all eleven* of the known properties of real-world graphs (see Table 2.2 for a list). As such, RTG won the **Best Knowledge Discovery Paper award** in the Conference on Principles and Practice of Knowledge Discovery (PKDD) 2009.

- **Utility-driven model for human communications:** We designed a realistic utility-driven graph generator, the Pay-and-Call model (§7.1), for modeling human communication behavior. This agent-based model allows us to understand the local mechanisms that play role in the formation of communication networks and to answer what-if scenarios.

In the last and third part, **Anomaly Detection**, we focus on the anomaly and event detection problem. We exploit compression based techniques, model fitting, and graph mining techniques to address the following problems: anomalous node detection in plain graphs, cluster and outlier detection in attributed graphs, and event detection in time series of graphs. We also develop visualization and sensemaking tools for the end user or analyst of the detected anomalies. The list of problems under the anomaly detection setting are:

> Given a large, weighted graph, how can we find anomalies? Which rules should be violated, before we label a node as an anomaly? (Chapter 9)

Given a graph with node attributes, how can we find meaningful patterns such as clusters of nodes and clusters of attributes, as well as bridges and outliers? (Chapter 10)

Given a database with categorical attributes, how can we find meaningful patterns that succinctly describe the data and spot outliers? (Chapter 11)

At what points in time many of the nodes in a given time-varying graph change their behavior significantly? Can we attribute the change to specific nodes, that is, can we characterize which nodes change in behavior the most? (Chapter 12)

How can we use the network structure to summarize a set of (anomalous) nodes, by partitioning them such that for each part we have a simple tour connecting the grouped nodes, while nodes in different parts are not easily reachable? In other words, how can we summarize the set of (anomalous) nodes for easy sensemaking? (Chapter 13)

Impact

We develop a collection of novel methods to detect anomalies in a graph under various settings: for graphs for which we only know the graph structure, for richer graphs with attributes, and for graphs changing over time. These methods serve as an ensemble that can be employed under different or changing conditions.

- **Automatic anomaly detection in plain graphs:** We are *among the first* to detect outlier nodes in *graph* data (§9.1), rather than in a collection of data points. As such, our ODDBALL won the **Best Paper award** in the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD) 2010. It has been integrated into Carnegie Mellon's Pegasus Tera-Scale Graph Mining system, and is a main component of ADAMS (Anomaly Detection at Multi Scale) research initiative supported by DARPA.
- **Parameter-free mining and clustering in attributed graphs:** We introduced a novel clustering model, called PICS (§10.1), to find groups of nodes in attributed graphs. PICS spots bridge-nodes with connections across clusters and outlier-nodes that do not belong well to any cluster. We also proposed a new approach, called COMPREX (§11.1), to identify anomalies in attributed graphs, that exploits pattern-based compression. Compression has been primarily used in communications theory and databases; our work is *one of few* to explore compression for data mining tasks. This work and its extension to time-evolving graphs led to **two patents** (filed with high-impact score) at IBM Research Labs.
- **Automatic event detection and "attribution" in time-evolving graphs:** We developed an algorithm (§12.1) to quantify "change" for time-varying graphs that flags "events" at which the change is significant. It also does "attribution"; to point out the specific nodes that contribute most to the change. This work was developed for the Network Science Collaborative Technology Alliance (NS-CTA) supported by the Army Research Labs.

We note that all of our methods (1) work for unlabeled and weighted graphs, and (2) operate in a completely unsupervised fashion. Moreover we enable "model-driven anomaly characterization", that is, we provide the reasoning for flagged anomalies, as well as novel tools for visualization and sensemaking (§13.1).

Part I

Patterns of Networks

Chapter 2

Preliminaries

2.1 Introduction

What do real-world networks look like on a global scale? What are the typical patterns in the *structure* and *dynamics* of networks at large? For example, how do nodes join a network and form edges over time? After an edge forms, what are the patterns associated with the weights on the edges? How do the different connected components of a network form and evolve? What are the typical patterns in human communications?

In this part, we begin to answer these questions by studying many networks from diverse domains. The idea is to analyze topological and dynamical features of these networks such as the diameter, connected components, edge weights, etc. and how they correlate with each other and evolve over time.

There has been extensive work focusing on *static* snapshots of *unweighted* graphs, where fascinating properties have been discovered, the most striking ones being the "small-world" phenomenon [Watts and Strogatz, 1998] (also known as "six degrees of separation" [Milgram, 1967]) and the power law degree distributions [Barabási and Albert, 1999; Faloutsos et al., 1999]. Time-evolving graphs have attracted attention only recently, where even more fascinating properties have been discovered, like *shrinking* diameters, and the so-called *densification power law* [Leskovec et al., 2005b].

One central topic of this part is the study of some of the most important properties apparent in networks, with a particular emphasis on *dynamic* networks, as well as some of the newer findings with respect to *weighted* networks. Most analyses in previous work focused on the "giant connected component" (GCC), either explicitly or implicitly, and moreover it ignored multiple links between nodes or weights on edges. Here we shift our focus to the components that are of moderate size but "disconnected" from the GCC of the graph, which we will refer to as the "next-largest connected components" (NLCCs). In particular, we study the dynamics of how these components form and merge. We also study edge weights, particularly how weighted edges are added over time and their correlation with other structural properties.

Another main theme of this part is the study of human communication networks. Most previous work in network science and social network analysis focus on node level degree distributions [Broder et al., 2000; Faloutsos et al., 1999; Leskovec et al., 2007b], communities [Nussbaum et al., 2010; Satuluri and Parthasarathy, 2009; Tantipathananandh et al., 2007], and triadic relations, such as clustering coefficients and triangle closures [Granovetter, 1973]. Here we build on triadic relations and study *social circles* people belong to in human-to-human networks, like phone call, text and instant messaging networks. We also study dyadic relations; specifically the *reciprocity* relations among individuals and the associated bivariate distributions. Furthermore, we find new patterns with respect to *call durations* of mobile phone users.

The questions of interest are:

- *How do networks behave over time?* Does the structure vary as the network grows? In what fashion do new entities enter a network? Does the network retain certain graph properties as it grows and evolves? Does the graph undergo a "phase transition", in which its behavior suddenly changes?
- *How do the next-largest connected components change over time?* One might argue that they grow, as new nodes are being added; and their size would probably remain a fixed fraction of the size of the GCC. Someone else might counter-argue that they shrink, and they eventually get absorbed into the GCC. What is happening, in real graphs?
- *What distributions and patterns do weighted graphs maintain?* How does the distribution of weights change over time– do we also observe a densification of weights as well as single-edges? How does the distribution of weights relate to the degree distribution? Is the edge weight related to the popularity of its adjacent nodes? Is the addition of weight bursty over time, or is it uniform?
- *What patterns should we expect in a network of human-to-human interactions?* How large are our social circles on average? If someone has many contacts, does that indicate popularity? How long are the phone calls of mobile phone users? What are the chances of a call to end, given its current duration? What is the best way to summarize the calling behavior of a user? What are typical reciprocity relations between individuals? What is reciprocity correlated in the network?

Answering these questions is important to understand how natural graphs form and evolve, and to (a) spot anomalous graphs and sub-graphs; (b) answer questions about entities in a network and what-if scenarios; and (c) develop understanding on human communication behavior.

Let's elaborate on each of the above applications: (a) Spotting anomalies is vital for determining abuse or fault in social and computer networks, such as unwanted calls and faulty equipment in communication networks, link-spamming in a web graph, fraudulent reputation building in e-auction systems [Pandit et al., 2007], detection of dwindling/abnormal social sub-groups in a social-networking site like Yahoo-360 (`360.yahoo.com`), Facebook (`www.facebook.com`) and LinkedIn (`www.linkedin.com`), network intrusion [Lazarevic et al., 2003]. (b) Analyzing network properties is also useful for identifying authorities and search algorithms [Borodin et al., 2005; Chakrabarti et al., 1999; Kumar et al., 2006], for discovering the "network value" of customers for using viral marketing [Richardson and Domingos, 2002], or to improve recommendation systems [Bell et al., 2007]. What-if scenarios are vital for extrapolation, provisioning

and algorithm design: For example if we expect that the number of links will double within the next year, we should provision for the appropriate hardware and infrastructure to store and process the upcoming queries. (c) Human behavior models provide insight on natural human behavior at individual level (e.g. evidence shows that reciprocal relationships are highly probable to persist in the future [Cesar A. Hidalgo, 2008]) as well as on global behavior at network scale (propagation of viruses in computer networks or spreading information and ideas in social networks clearly depend on the presence of mutual links in the network).

The following two chapters in this part are based on work as cited below.

- Chapter 3 Patterns in graph topology
  [Akoglu et al., 2008; Kang et al., 2010a; Mcglohon et al., 2008]
- Chapter 4 Patterns in human communications
  [Akoglu et al., 2012a; Du et al., 2009; Vaz de Melo et al., 2010].

Before delving into the study of real graph patterns, we next establish the terms and definitions we use in the rest of this part, present related work, and introduce the datasets we studied.

2.2 Definitions

In this section we provide basic definitions and terms we use. A full list of symbols is listed in Table 2.1.

Networks as Graphs

A network is typically represented by a *graph*. Throughout the thesis we use *network* and *graph* interchangeably.

A static, unweighted graph G consists of a set of nodes V and a set of edges E: $G = (V, E)$. We represent the sizes of V and E as $|V|$ and $|E|$. A graph may be *directed* or *undirected*– for instance, a phone call may be from one party to another, and will have a directed edge, or a mutual friendship may be represented as an undirected edge.

Graphs may also be *weighted*, where there may be multiple edges occurring between two nodes (e.g. repeated phone calls) or specific edge weights (e.g. monetary amounts for transactions). In a weighted graph $\mathcal{G}$, let $e_{i,j}$ be the edge between node i and node j. We shall refer to these two nodes as the "neighboring nodes" or "incident nodes" of edge $e_{i,j}$. Let $w_{i,j}$ be the weight on edge $e_{i,j}$. The *total weight* w_i of node i is defined as the sum of weights of all its incident edges, that is $w_i = \sum_{k=1}^{d_i} w_{i,k}$, where d_i denotes its degree. As we show later, there is a relation between a given edge weight $w_{i,j}$ and the weights of its neighboring nodes w_i and w_j.

Finally, graphs may be *unipartite* or *multipartite*. Most social networks one thinks of are unipartite– people in a group, papers in a citation network, etc. However, there may also be multi-partite graphs– that is, there are multiple classes of nodes and edges that are only drawn between nodes

Symbol	Description
$\mathcal{G}$	Graph representation of datasets
$\mathcal{V}$	Set of nodes for graph $\mathcal{G}$
$\mathcal{E}$	Set of edges for graph $\mathcal{G}$
N	Number of nodes, or $\|\mathcal{V}\|$
E	Number of edges, or $\|\mathcal{E}\|$
$e_{i,j}$	Edge between node i and node j
$w_{i,j}$	Weight on edge $e_{i,j}$
w_i	Weight of node i (sum of weights of incident edges)
$\mathbf{A}$	0-1 Adjacency matrix of the unweighted graph
$\mathbf{A}_w$	Real-value adjacency matrix of the weighted graph
$a_{i,j}$	Entry in matrix $\mathbf{A}$
λ_1	Principal eigenvalue of unweighted graph
$\lambda_{1,w}$	Principal eigenvalue of weighted graph

Table 2.1: Table of symbols used in notation.

of different classes. Bipartite graphs, like the IMDB movie-actor graph, consist of disjoint sets of nodes V_1 and V_2, say, for authors and movies, with no edges among nodes of same type.

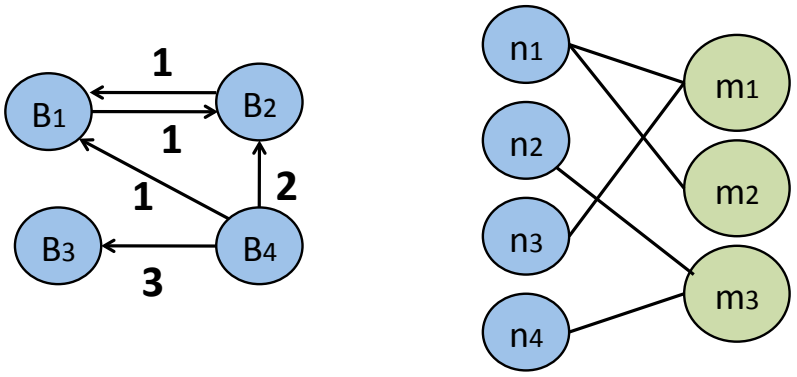
We can also represent a network with an *adjacency matrix* $\mathbf{A}$, where nodes are in rows and columns, and numbers in the matrix indicate the existence of edges. For unweighted graphs, all entries are 0 or 1; for weighted graphs the adjacency matrix contains the values of the weights. Figure 2.1 shows examples of graphs and their adjacency matrices.

We next introduce other important concepts we use in analyzing these graphs.

Components and Cliques

We refer to a *connected component* in a graph as a set of nodes and edges where there exists a path between any two nodes in the set (For directed graphs, this translates to a *weakly connected component* whereas a *strongly connected component* requires a directed path between its pairs of nodes). We find that in real graphs over time, a giant connected component (GCC) forms. However, it is also of interest to study the smaller components– when do they choose to join the GCC, what size do they reach before doing so, and what does the distribution of component sizes look like? We provide answers to these questions in Chapter 3.

Given a subgraph G_i of G, if $\forall u, v \in VG_i, \exists (u, v) \in EG$, then G_i is called a complete subgraph or a *clique* of G. Furthermore, if there is no other subgraph G_j that is also a clique of G with $VG_j \supset VG_i$, G_i is called a *maximal clique* of G. In Chapter 4, we treat the maximal cliques in a graph as *social circles*, and study their properties in human communication networks.



	B1	B2	B3	B4
B1	0	1	0	0
B2	1	0	0	0
B3	0	0	0	0
B4	1	2	3	0

	n1	n2	n3	n4
m1	1	0	1	0
m2	1	0	0	0
m3	0	1	0	1

Figure 2.1: Illustrations of example graphs. On the left is a unipartite, directed, weighted graph and the corresponding adjacency matrix. On the right is an undirected, bipartite graph and the corresponding adjacency matrix.

Diameter and Effective Diameter

For a given (static) graph, its *diameter* is defined as the maximum *distance* between any two nodes, where distance is the minimum number of hops (i.e., edges that must be traversed) on the path from one node to another, ignoring directionality and edge-weights.

Since the diameter is defined as the *maximum*-length shortest path between all possible pairs, it can easily be hijacked by long chains. Therefore, often the *effective diameter* is used as a more robust metric, which is the 90-percentile of the pairwise distances among all reachable pairs of nodes. In other words, the *effective diameter* is the minimum number of hops in which some fraction (usually 90%) of all connected node pairs can be reached [Siganos et al., 2006].

Calculating the diameter of a graph is $O(N^2)$. Therefore, we choose to estimate the graph diameter by sampling nodes from the giant component. For $s = \{1, 2, ..., S\}$, we choose two nodes at random and calculate the distance (using breadth-first search). We then choose to record the 90 percentile value of distances, so we take the $.9S$ largest recorded value. The distance operation is $O(dk)$, where d is the graph diameter and k the maximum degree of any node– on average this is a much smaller cost. Intuitively, the diameter represents how much of a "small world" the graph is– how quickly one can get from one "end" of the graph to another [Tauro et al., 2001]. Alternative methods to sampling would include ANF [Palmer et al., 2002].

Radius and Effective Radius

While the diameter is defined over an entire graph, radius is defined for a particular node in the graph. Specifically, for a given node, its *radius* is defined as the maximum of all its *distances* to other nodes in the graph. Similar to effective diameter as described before, its *effective radius* is the 90-percentile of these distances.

Heavy-tailed Distributions

While the Gaussian distribution is common in nature, there are many cases where the probability of events far to the right of the mean is significantly higher than in Gaussians. In the Internet, for example, most routers have a very low degree (perhaps "home" routers), while a few routers have extremely high degree (perhaps the "core" routers of the Internet backbone) [Faloutsos et al., 1999] Heavy-tailed distributions attempt to model this. They are known as "heavy-tailed" because, while traditional exponential distributions have bounded variance (large deviations from the mean become nearly impossible), $p(x)$ decays polynomially quickly instead of exponentially as $x \to \infty$, creating a "fat tail" for extreme values on the PDF plot.

One of the more well-known heavy-tailed distributions is the power law distribution. Two variables x and y are related by a power law when:

$$y(x) = Ax^{-\gamma} \tag{2.1}$$

where A and γ are positive constants –γ is often called the power law exponent.

A random variable is distributed according to a power law when the probability density function (pdf) is given by:

$$p(x) = Ax^{-\gamma}, \quad \gamma > 1, x \geq x_{min} \tag{2.2}$$

The extra $\gamma > 1$ requirement ensures that $p(x)$ can be normalized. Power laws with $\gamma < 1$ rarely occur in nature, if ever [Newman, 2005].

Skewed distributions, such as power laws, occur very often in real-world graphs, as we will discuss. Figures 2.2(a) and 2.2(b) show two examples of power laws.

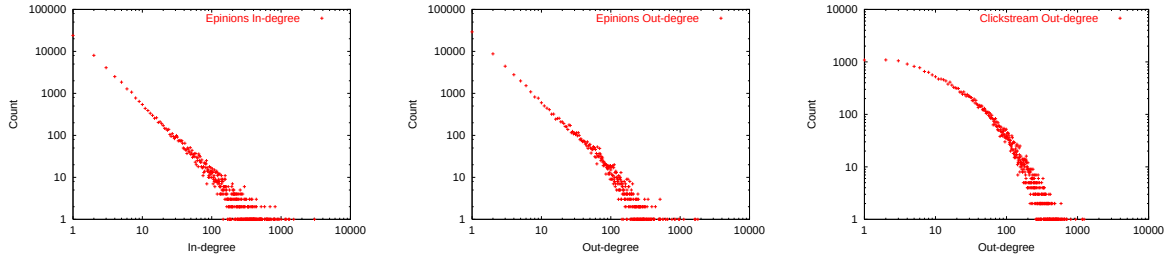
While power laws appear in a large number of graphs, deviations from a pure power law are also observed. Two most common deviations are exponential cutoffs and lognormals.

Sometimes, the distribution looks like a power law over the lower range of values along the x-axis, but decays very fast for higher values. Often, this decay is exponential, and this is usually called an exponential cutoff:

$$y(x = k) \propto e^{-k/\kappa} k^{-\gamma} \tag{2.3}$$

where $e^{-k/\kappa}$ is the exponential cutoff term and $k^{-\gamma}$ is the power law term.

Similar distributions were studied by [Bi et al., 2001], who found that a discrete truncated lognormal (called the Discrete Gaussian Exponential or "DGX" by the authors) gives a very good fit. A lognormal is a distribution whose logarithm is a Gaussian; it looks like a truncated parabola



(a) Epinions In-degree (b) Epinions Out-degree (c) Clickstream Out-degree

Figure 2.2: *Power laws and deviations:* Plots (a) and (b) show the in-degree and out-degree distributions on a log-log scale for the *Epinions* graph (an online social network of $75,888$ people and $508,960$ edges [Richardson and Domingos, 2001]). Both follow power-laws. In contrast, plot (c) shows the out-degree distribution of a *Clickstream* graph (a bipartite graph of users and the websites they surf [Montgomery and Faloutsos, 2001]), which deviates from the power-law pattern.

in log-log scales. The DGX distribution has been used to fit the degree distribution of a bipartite "clickstream" graph linking websites and users (see Figure 2.2(c)).

Methods for fitting heavy-tailed distributions are described in [Clauset et al., 2009; Newman, 2005].

Burstiness and Entropy Plots

Human activity, including edge additions in graphs, is often bursty. If the traffic is *self-similar*, then we can measure the burstiness, using the intrinsic, or *fractal* dimension of the cloud of timestamps of edge-additions (or weight-additions). Let $\Delta W(t)$ be the total weight of edges that were added during the t-th interval, e.g., the total network flow on day t, among all the machines we are observing.

Among the many methods that measure self-similarity (Hurst exponent, etc. [Schroeder, 1991]), we choose the *entropy plot* [Wang et al., 2002], which plots the entropy $H(r)$ versus the resolution r. The resolution is the scale, that is, at resolution r, we divide our time interval into 2^r equal sub-intervals, sum the weight-additions $\Delta W(t)$ in each sub-interval k ($k = 1 \ldots 2^r$), normalize into fractions p_k (= $\Delta W(t)/W_{total}$), and compute the Shannon entropy of the sequence p_k: $H(r) = -\sum_k p_k \log_2 p_k$. If the plot $H(r)$ is linear in some range of resolutions, the corresponding time sequence is said to be *fractal* in that range, and the slope of the plot is defined as the *intrinsic* (or *fractal*) dimension D of the time sequence. Notice that a uniform weight-addition distribution yields D=1; a lower value of D corresponds to a more bursty time sequence like a Cantor dust [Schroeder, 1991], with a single burst having the lowest D=0: the intrinsic dimension of a point. Also note that the "b-model" [Wang et al., 2002], a variation of the 80-20 model, generates such self-similar traffic.

2.3 Related Work

For the purpose of organization, we divide related work into those applying to the study of graph structure and dynamics and to the study of human communications.

2.3.1 Previous patterns in structure and dynamics

Properties of real-world graphs observed to date can be summarized under a "2-by-2" grid structure: those on *static* graphs (structural patterns) and those on *dynamic* graphs (temporal patterns), each of which can further be subdivided into two –*unweighted* and *weighted* patterns.

An overview of all known real-world graph properties, including the newer ones that we will describe in this thesis, is given in Table 2.2. The patterns highlighted in **bold** are introduced in this thesis in Chapter 3. As one can notice, our focus is mostly on weighted graphs and dynamic properties. In this section, we present previously discovered patterns; namely patterns in degree distributions, diameter, triangles, eigenvalues, community structure, and graph density. We describe those in detail next.

	Unweighted	**Weighted**
Static	SU-1 Heavy-tailed degree distribution SU-2 Small diameter SU-3 Triangle Power Law (TPL) SU-4 Eigenvalue power law (EPL) SU-5 Community structure SU-6 **Chain-like NLCCs**	SW-1 **Edge Weights Power Law (EWPL)** SW-2 **Snapshot Power Law (SPL)**
Dynamic	DU-1 Shrinking diameter DU-2 Densification Power Law (DPL) DU-3 **Diameter plot and Gelling point** DU-4 **Constant/Oscillating NLCCs** DU-5 **Stable fractal dimension of CCs** DU-6 **'Rebel' prob. to NLCCs** DU-7 **Principal eigenvalue over time**	DW-1 **Weight Power Law (WPL)** DW-2 **Bursty weight additions** DW-3 **Weighted principal eig. over time**

Table 2.2: Summary of real graph properties. Patterns in bold face are introduced in this thesis.

SU-1: Heavy-tailed Degree Distribution

The degree distribution of many real graphs obey a power law of the form $f(d) \propto d^{-\alpha}$, with the exponent $\alpha > 0$, and $f(d)$ being the fraction of nodes with degree d. Such power-law relations as well as many more have been reported in [Chakrabarti et al., 2004b; Faloutsos et al., 1999; Kleinberg et al., 1999; Newman, 2005]. Intuitively, power-law-like distributions for degrees state that there are many low degree nodes, whereas only a few high degree nodes (also referred to as "hubs") exist in real graphs.

SU-2: Small Diameter

One of the most striking patterns that real-world graphs have is a small diameter, which is also known as the "small-world phenomenon" or the "six degrees of separation". Intuitively, the diameter represents how much of a "small world" the graph is– how quickly one can get from one "end" of the graph to another.

Many real graphs were found to exhibit surprisingly small diameters– for example, 19 for the Web [Albert et al., 1999], and the well-known number 6 in social networks [Barabási, 2003]. Most recently [Kang et al., 2010b] showed that even a billion-node Yahoo! Web graph has diameter only 7.6.

SU-3: Triangle Power Law (TPL)

The number of triangles Δ and the number of nodes that participate in Δ number of triangles follow a power-law in the form of $f(\Delta) \propto \Delta^{\sigma}$, with the exponent $\sigma < 0$ [Tsourakakis, 2008]. The TPL intuitively states that while many nodes have only a few triangles in their local neighborhoods, a few nodes participate in many number of triangles with their neighbors. The local number of triangles is also related to the clustering coefficient of graphs.

SU-4: Spectral properties and Eigenvalue Power Law (EPL)

There have been several studies on spectral properties of power-law graphs. [Siganos et al., 2003] examined the spectrum of the adjacency matrix of the AS Internet topology and reported that the 20 or so largest eigenvalues of the Internet graph are power-law distributed with exponent between .45 and .5. [Mihail and Papadimitriou, 2002] later provided an explanation for the "Eigenvalue Power Law", showing that it is a consequence of the "Degree Power Law".

[Farkas et al., 2001] studied the numerical and analytical properties of the adjacency matrices of complex networks and reported surprising results on the spectra of adjacency matrices corresponding to several models of real-world graphs. Moreover, [Chung et al., 2003] analyzed the eigenvalues of random graphs for which the number of nodes of degree d follow a power law and reported bounds on the first and second eigenvalues of such graphs for certain parameters.

SU-5: Community Structure

Real-world graphs are found to exhibit a modular structure, with nodes forming groups, and possibly groups within groups [Flake et al., 2002; Girvan and Newman, 2002; Schwartz and Wood, 1992]. In a modular graph, the nodes form communities where groups of nodes in the same community are tighter connected to each other than to those nodes outside the community. Quantitative measures for such a structure include *modularity* [Girvan and Newman, 2002] and *conductance* [Andersen et al., 2006].

DU-1: Shrinking Diameter

[Leskovec et al., 2005b] showed that not only is the diameter of real graphs small, but it also *shrinks* and then *stabilizes* over time. This pattern can be attributed to the "gelling point" and the "densification" in real graphs both of which are described in the following sections. Briefly, at the "gelling point" many small disconnected components merge and form the largest connected component in the graph. This can be thought as the 'coalescence' of the graph at which point the diameter 'spikes'. Afterwards, with the addition of new edges the graph 'densifies' and the diameter keeps shrinking until it reaches an equilibrium.

DU-2: Densification Power Law (DPL)

Time-evolving graphs follow the "Densification Power Law" with the equation $E(t) \propto N(t)^{\beta}$, at all time ticks t [Leskovec et al., 2005b], where β is the densification exponent, and $E(t)$ and $N(t)$ are the number of edges and nodes at time t, respectively.

Real graphs studied are shown to obey the DPL, with exponents between 1.03 and 1.7. The power-law exponent being greater than 1 indicates a super-linearity between the number of nodes and the number of edges in real graphs. That is, it implies that for example when the number of nodes N in a graph doubles, the number of edges E more than doubles– hence the densification. It also explains away the shrinking diameter phenomenon described earlier.

2.3.2 Previous studies on human communications

There has been previous work on studying human-to-human communication networks in order to understand various aspects of human behavior. [Onnela et al., 2007b] and [Onnela et al., 2007a] built a network from mobile phone calls records and, from it, they make a detailed analysis of its network properties. They identified relationships between node weights and network topology, finding that the weak ties are commonly responsible for linking communities, thus having a high betweenness centrality or low link overlap. Moreover, [Hidalgo and Rodriguez-Sickert, 2008] verified that the persistence of an edge is highly correlated to its reciprocity and to the topological overlap. It is also common to analyze the networks from mobile companies in order to improve their services. For instance, [Cortes et al., 2001; Hill and Nagle, 2009] proposed a framework

and data structures for identifying fraudulent consumers on telecommunication networks based on their degree distribution and dynamics and, [Nanavati et al., 2006] proposed metrics that can be employed by a business strategy planner involved in the telecom domain.

Another use for a mobile phone dataset is to study the individual attributes of the users. [Guo et al., 2007] analyzed mobile phone calls that arrived in a mobile switch center in a GSM system of Qingdao, China, and they found that the duration of the phone calls is best modeled by a log-normal distribution. Later, [Seshadri et al., 2008] proposed the DPLN distribution to model the distributions of the number of phone calls per customer, the total talk minutes per customer and the distinct number of calling partners per customer. [Willkomm et al., 2008] studied the duration of mobile calls arriving at a base station during different periods and also found that they are neither exponentially nor log-normally distributed, possessing significant deviations that make them hard to model. They verified that about 10% of calls have a duration of around 27 seconds, that correspond to calls which the called mobile users did not answer and the calls were redirected to voice-mail.

2.4 Datasets

We studied several large real networks, described in detail in Table 2.3, in terms of size, weights, and explanation of edges. Several of our graphs had no obvious weighting scheme: for example, a single paper or patent will cite another only a single time. The graphs that did have weights are also further divided into two schemes, *multi-edges* and *edge-weights*. In the edge-weights scheme, there is an obvious weight on edges, such as dollar amounts in campaign donations, or packet-counts in network traffic. For multi-edges, weights are added if there is more than one interaction between two nodes. For instance, if a blog cites another blog at a given time, its weight is 1. If it cites the blog again later, the weight becomes 2. In all the datasets, we assume that edges are never deleted, because edge deletion never explicitly appeared.

The datasets are gathered from publicly available data. *NIPS*[1], *Arxiv* and *Patent* [Leskovec et al., 2005b] are academic paper or patent citation graphs with no weighting scheme. *IMDB* indicates movie-actor information, where an edge occurs if an actor participates in a movie [Barabási and Albert, 1999]. *Netflix* is the dataset from the Netflix Prize competition[2], with user-movie links (we ignored the ratings); we also noticed that it only contained users with 100 or more ratings. *BlogNet* and *PostNet* are two representations of the same data, hyperlinks between blog posts [Leskovec et al., 2007b]. In *PostNet* nodes represent individual posts, while in *BlogNet* each node represents a blog. Essentially, *PostNet* is a paper citation network while *BlogNet* is an author citation network (which contains multi-edges).

Oregon is an autonomous systems network [3], which contains AS peering information inferred from Oregon route-views BGP data. *Enron* contains email interactions at Enron collected from

[1] `www.cs.toronto.edu/~roweis/data.html`
[2] `www.netflixprize.com`
[3] University of Oregon *Route Views* project, `www.routeviews.org`

Name	Uni/bipartite	Weights	$\|V\|$, $\|E\|$, time	Description
NIPS	Unipartite	None	2K, 3K, 13 yr.	Citation network from NIPS
Arxiv	Unipartite	None	30K, 60K, 13 yr.	Physics citations
Patent	Unipartite	None	6M, 10M, 19 yr.	U.S. Patent citations
IMDB	Bipartite	None	757K, 2M, 114 yr.	Actor-movie network
Netflix	Bipartite	None	125K, 14M, 72 mo.	User-movie ratings
BlogNet	Unipartite	Multi-edges	60K, 125K, 80 days	Social network of blogs based on citations
PostNet	Unipartite	None	250K, 218K, 80 days	Post citations from blogs
Oregon	Unipartite	None	12K, 38K, 6 mo.	Autonomous systems network
Enron	Unipartite	None	36K, 183K, N/A	Email associations at Enron Inc.
NetTraffic	Unipartite	Packet-size	21K, 2M, 52 mo.	Network traffic
Auth-Conf	Bipartite	Multi-edges	17K, 22K, 25 yr.	DBLP Author-to-Conference associations
Key-Conf	Bipartite	Multi-edges	10K, 23K, 25 yr.	DBLP Keyword-to-Conference associations
Auth-Key	Bipartite	Multi-edges	27K, 189K, 25 yr.	DBLP Author-to-Keyword associations
CampOrg	Bipartite	$ Amounts	23K, 877K, 28 yr.	U.S. electoral donations from organizations to candidates
CampIndiv	Bipartite	$ Amounts	6M, 10M, 22 yr.	U.S. electoral donations from individuals to organizations (available from FEC)
HEP-PH	Unipartite	None	30K, 347K, 11 yr.	Physics PHenomenology paper citations
HEP-TH	Unipartite	None	27K, 351K, 11 yr.	Physics THeory paper citations
YahooWeb	Unipartite	None	1.4B, 6.6B, N/A	Web pages crawled by Yahoo in 2002
CALL	Unipartite	Duration	1.87M, 49.5M, 6 mo.	Phone call interactions of individuals
SMS	Unipartite	Multi-edges	1.87M, 8.8M, 6 mo.	SMS texting interactions of individuals
Commun	Unipartite	Multi-edges	2.4M, 5.3M, 5 mo.	Voice, IM, SMS interactions of individuals

Table 2.3: Graph data sets studied for pattern discovery. K: thousand, M: million, B:billion.

about 1998 to 2002 (made public by the Federal Energy Regulatory Commission during its investigation). *NetTraffic* records IP-source/IP-destination pairs, along with the number of packets sent, per unit time. *Auth-Conf*, *Key-Conf*, and *Auth-Key* are all from DBLP [4], with the obvious meanings. *CampOrg* and *CampIndiv* are bipartite graphs from U.S. Federal Election Commission. They record donations (dollar amounts) between organizations and political candidates, and between individuals and organizations [5].

HEP-PH and *HEP-TH* contain Physics phenomenology and theory paper citations, respectively. *YahooWeb* graph contains 1.4 billion Web pages and 6.6 billion links among them. It was crawled by Yahoo-Altavista search engine in 2002. It contains around 31 million unique sites. The top 3 sites with largest number of pages are http://metaquest.bc.edu/, http://www.cricket.org/, http://www.nyu.edu/.

Datasets used to analyze patterns in human communications span several months of activity and cover different types of communications including phone call, SMS (Short Message Service), and IM (Instant Message). The edge weights denote the total count of phone calls, SMSs, and IMs exchanged as well as total durations of calls aggregated in minutes. In particular, CALL and SMS are the phone call and SMS interactions among the same anonymous individuals. The dataset was collected over a period of six months, December 1, 2007 through May 31, 2008 in a big city in Asia. Finally, *Commun* is another communications dataset among a different set of individuals from another big city in Asia.

[4] `dblp.uni-trier.de/xml/`

[5] `www.cs.cmu.edu/~mmcgloho/fec/data/fec_data.html`

Chapter 3

Patterns in network topology

PROBLEM STATEMENT: *What are the typical patterns in the structure and dynamics of real-world networks from a large collection of diverse domains (e.g. political campaign donations, computer network traffic, online social network, large Web graph)?*

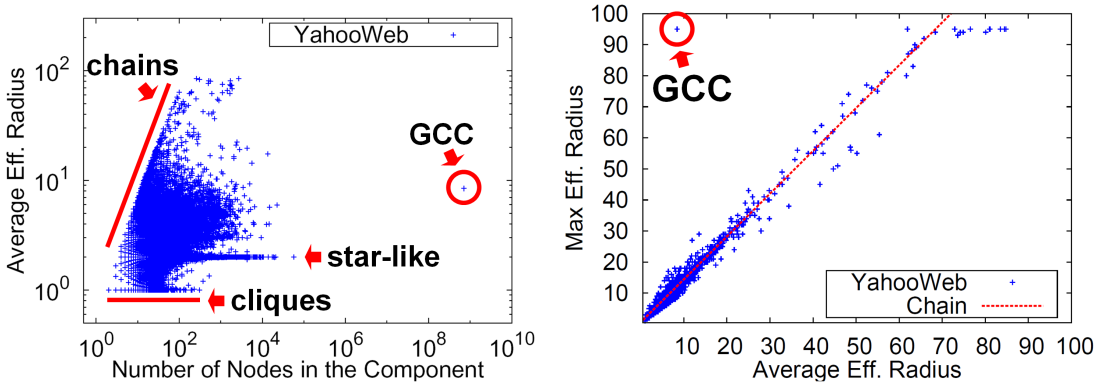
In this chapter, we introduce several new patterns we found in real networks we studied. An overview of all the existing as well as new patterns has been given in Table 2.2, the ones in **boldface** are our contributions. As one can notice, our major contributions include mainly *dynamic* and *weighted* patterns. For the purposes of organization we will present them in two categories; unweighted and weighted graph patterns.

3.1 Unweighted graph patterns

3.1.1 Pattern SU-6: Chain-like Next-Largest Connected Components (NLCCs)

We study the relations between the average effective radius and the order of connected components. Recall that the effective radius of a node in a component is the 90-percentile of all the distances from that node to other nodes in the component. The average effective radius (AER) of a connected component is then the average of the effective radii in the component.

Figure 3.1 (a) shows the number of nodes and the AER of connected components in *YahooWeb*. We first see that there are many components which form cliques (=AER close to 1), stars (=AER close to 2), and chains(=AER proportional to the number of nodes). Since a component that behaves like a clique seems suspicious, we looked at the top 4 largest clique-like components. It turns out that they all belong to Germany; they have the same number of nodes (305), and seem to be phishing sites from the same owner as the contents look similar, which existed several years ago but non-existing any more.



(a) AER vs. $|V|$ (b) MER vs. AER

Figure 3.1: (a) Connected components map of the *YahooWeb* graph, showing the Average Effective Radius(AER) vs. number of nodes in each component. Each point corresponds to a connected component. Notice the effective radius is bounded by the maximum(by chains) and the minimum(by clique). **(b)** Maximum Effective Radius(MER) vs. Average Effective Radius(AER). Notice that small components behave like chains in terms of radius, and the giant connected component (GCC) is very different from others due to the large MER compared to the AER.

Another observation is that the upper left boundary of Figure 3.1 (a) forms a near-line. Obviously, these nodes came from chains since they have the maximum average effective radius. We can prove that the boundary is a near-line since the average effective radius and the number of nodes in chain graphs have the following near-linear relationship:

Lemma 1 (Small Radius of Chain Graph) *The average effective radius of a chain graph with* n *nodes is* $0.6525n - 0.45$.

Proof 1 *We need to consider only the first* $\frac{n}{2}$ *nodes by the symmetry. Since the effective radius is the 90th-percentile, the* i*th nodes where* $1 \leq i \leq 0.45n$ *have the effective radius* $(0.9n - i)$*, and the last* $0.05n$ *nodes have radius* $0.45n$*. Therefore, the average effective radius is* $\frac{\sum_{i=1}^{0.45n}(0.9n-i)+0.45n\times 0.05n}{0.5n} = 0.6525n - 0.45$.

Next, we look at the maximum effective radius versus the average effective radius of each component in Figure 3.1 (b). We have the following observation.

Observation 3.1.1 (Chain-like Next-largest Connected Components) *Next-largest connected components have relatively small MER (Maximum Effective Radius) vs. AER (Average Effective Radius) ratio. Only the giant connected component has the high ratio.*

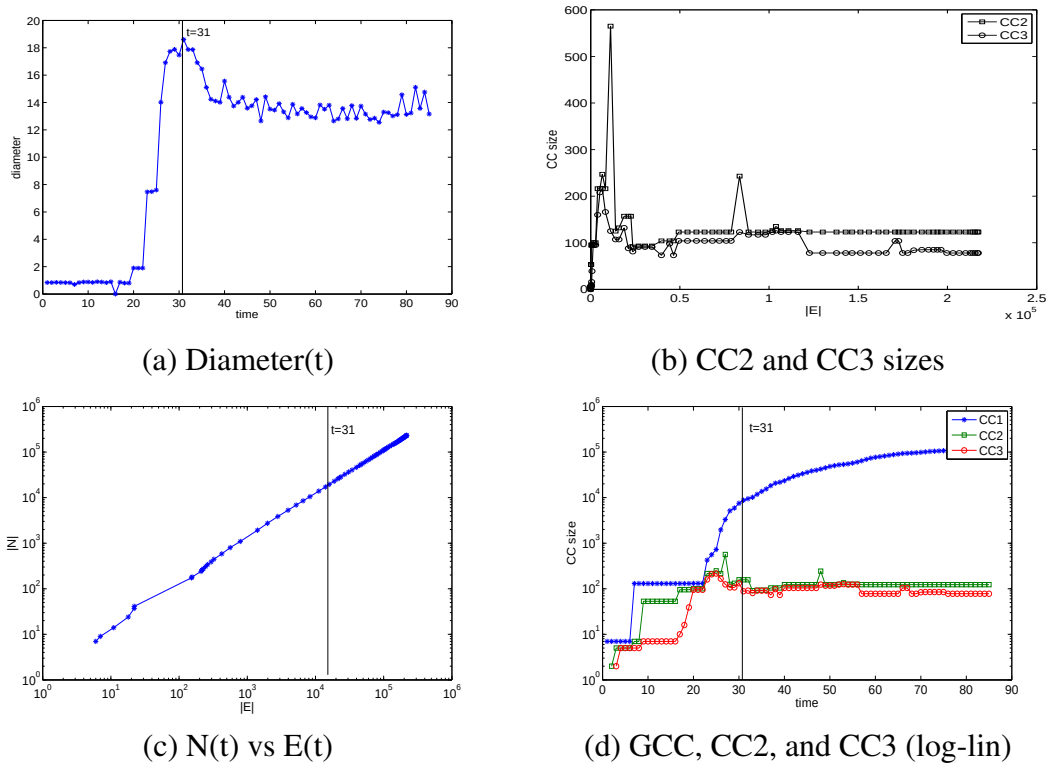
The reason of the high maximum radius vs. average effective radius ratio of the GCC can be explained by the thick cores containing majority of nodes and several tendrils it has [Broder et al., 2000]. The thick cores decrease the average effective radius, while the tendril increases the maximum radius. On the contrary, NLCCs of the *YahooWeb* graph do not have enough number of nodes to form thick cores which could have decreased the average effective radius. In terms of MER vs. AER ratio, the NLCCs are similar to chain graphs.

3.1.2 Pattern DU-3: Diameter Plot and "Gelling" Point

Studying the effective diameter of the graphs, we notice that there is often a point in time when the diameter spikes. Before that point, the graph is more or less in an establishment period, typically consisting of a collection of small, disconnected components. This "*gelling point*" seems to also be the time where the GCC "takes off". After the gelling point, the graph obeys the expected rules, such as the densification power law; its diameter decreases or stabilizes; the giant connected component keeps growing, absorbing the vast majority of the newcomer nodes.

Observation 3.1.2 (Gelling point) *Real graphs exhibit a gelling point, at which the diameter spikes and (several) disconnected components gel into a giant component.*

We show diameter plots over time in Fig. 3.2 (a), Fig. 3.3 (a)s, and Fig. 3.4 (a)s. In most of these graphs, both unipartite and bipartite, there are clear early gelling points. For example, in *NIPS* the diameter spikes at $t = 8$ years, which is a reasonable time for an academic community to gel. In some networks, we only see one side of the spike, due to data construction (the nature of trace-routes in *Oregon* and *NetTraffic*) or massive network size (*Patent*).



(a) Diameter(t)

(b) CC2 and CC3 sizes

(c) N(t) vs E(t)

(d) GCC, CC2, and CC3 (log-lin)

Figure 3.2: Properties of *PostNet* network. Notice that we experience an early gelling point at (a) (diameter versus time), stabilization/oscillation of the NLCC sizes in (b) (size of 2nd and 3rd CC, versus time). The vertical line marks the gelling point. Part (c) gives $N(t)$ vs $E(t)$ in log-log scales - the good linear fit agrees with the Densification Power Law. Part (d): component size (in log), vs time - the GCC is included, and it clearly dominates the rest, after the gelling point.

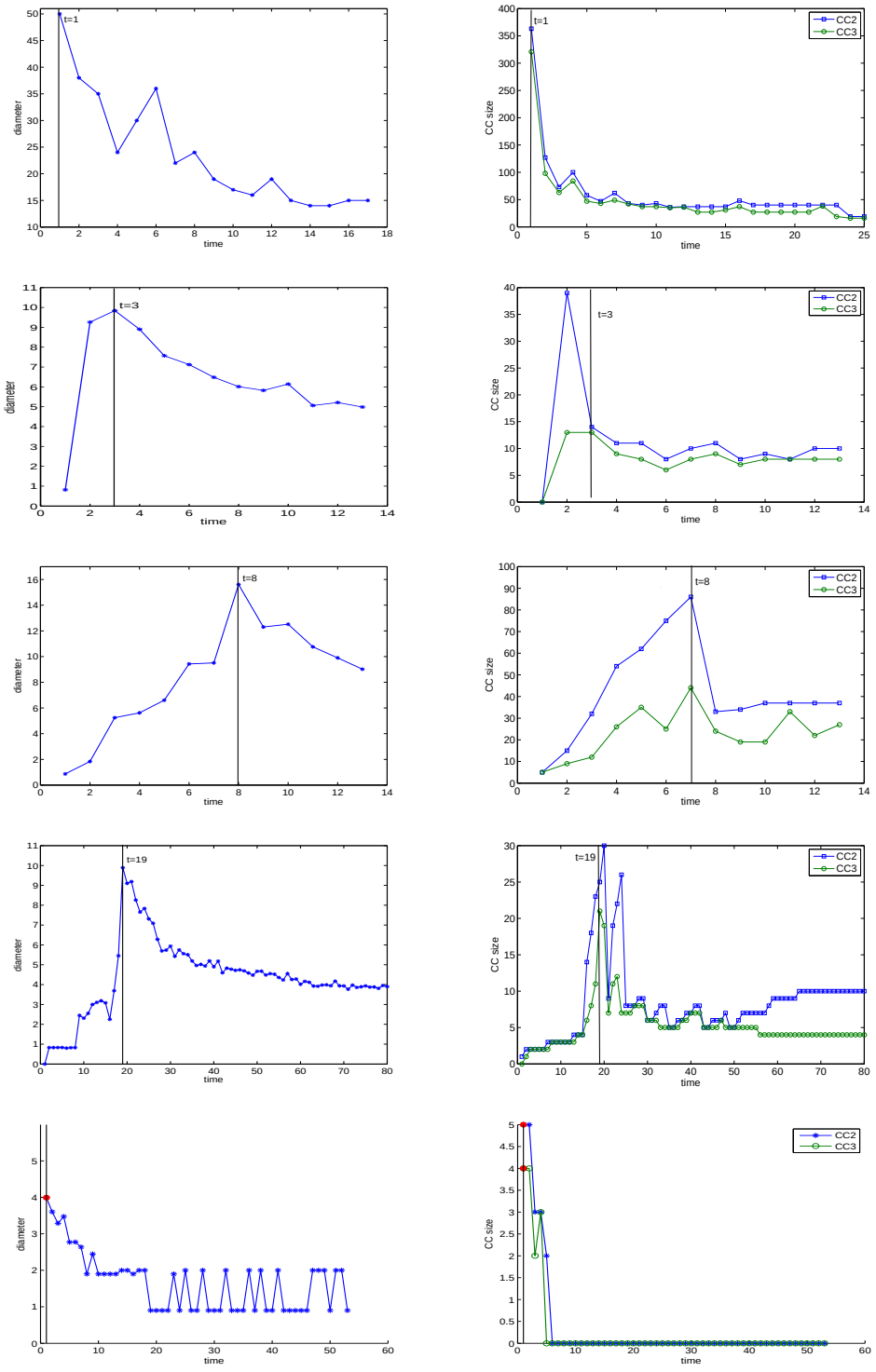


Figure 3.3: Properties of other unipartite networks. Diameter plot (left column), and NLCCs over time (right); vertical line marks the gelling point. Datasets from top to bottom: *Patent*, *Arxiv*, *NIPS*, *BlogNet*, *NetTraffic*. All datasets exhibit an early gelling point, and stabilization of the NLCCs.

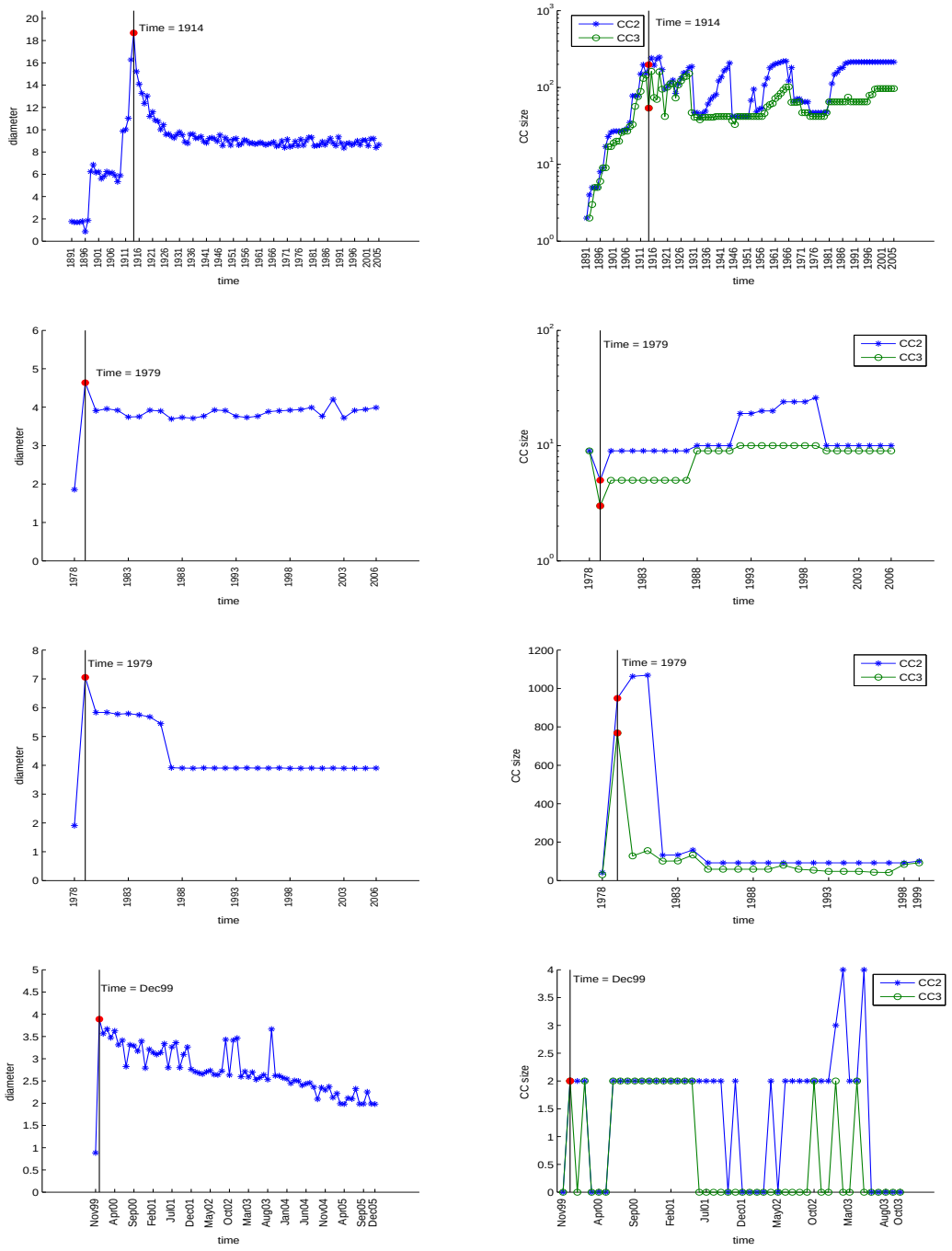


Figure 3.4: Properties of bipartite networks. Diameter plot (left column), and NLCCs over time (right), with vertical line marking the gelling point. Datasets from top to bottom: *IMDB*, *CampOrg*, *CampIndiv*, *Netflix*. Again, all datasets exhibit an early gelling point, and stabilization of the NLCCs. *Netflix* has strange behavior because it is masked (see data description 2.4) and text.

3.1.3 Pattern DU-4: Constant/Oscillating NLCCs

How do the next-largest connected components evolve, especially after the gelling point? We particularly study the second and the third largest connected components over time. We notice that, after the gelling point, the sizes of these components *oscillate* over time. Further investigation shows that the oscillation may be explained as follows: new-comer nodes typically link to the GCC; very few of the newcomers link to the 2nd (or 3rd) CC, helping them to grow slowly; in very rare cases, a newcomer links both to an NLCC, as well as the GCC, thus leading to the absorption of the NLCC into the GCC. It is exactly at these times that we have a drop in the size of the 2nd CC: Note that edges are not removed, thus, what is reported as the size of the 2nd CC is actually the size of yesterday's 3rd CC, causing the apparent "oscillation".

An unexpected (to us, at least) observation is that the largest size these components can get seems to be a constant. This is counter-intuitive – based on random graph theory, we would expect the size of the NLCCs to grow with increasing N [Bollobás, 2001]. Using scale-free arguments, we would expect the NLCCs to have size that would be a (small, but constant) fraction of the size of the GCC – to our surprise, this *never* happened, on any of the real graphs we analyzed. If some underlying growth does exist, it was small enough to be impossible to observe throughout the (often lengthy) time in our datasets.

Observation 3.1.3 (Constant/Oscillating NLCCs) *After the gelling point, the secondary and tertiary connected components remain of approximately constant size, with small oscillations.*

We show full results for *PostNet* in Fig. 3.2, including the diameter plot (Fig. 3.2 (a)), sizes of the NLCCs (Fig. 3.2 (b)), densification plot (Fig. 3.2 (c)), and the sizes of the three largest connected components in log-linear scale, to observe how the GCC dominates the others (Fig. 3.2 (d)). Results from other networks are similar, and are shown in condensed form for space (Fig. 3.3 for unipartite graphs, and Fig. 3.4 for bipartite graphs).

The second columns of Fig. 3.3 and Fig. 3.4 show the NLCC sizes versus time. Notice that, after the "gelling" point (marked with a vertical line), they all oscillate about constant value (different for each network). The only extreme cases are datasets with unusually high connectivity. For example, *Netflix* has very small NLCCs. This may be explained by the fact the dataset is masked, omitting users with less than a hundred ratings (possibly to further protect the privacy of the encrypted user-ids). Therefore, the graph has abnormally high connectivity. We note that *Oregon* and *NetTraffic* have unusually high connectivity due to the nature of network traffic– it benefits from having a single GCC, so NLCCs shrink to zero.

3.1.4 Pattern DU-5: Stable Fractal Dimension of Connected Components

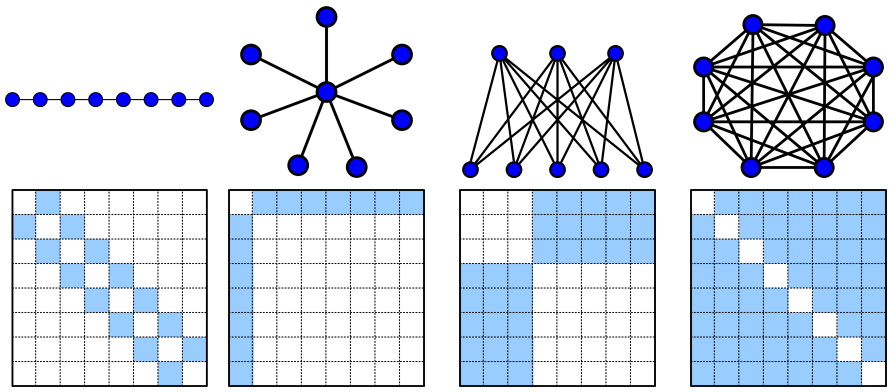
Do next-largest connected components have the same structural properties as the giant connected component? We study the homogeneity of components in large networks. The main questions of interest are the followings: (a) How can we characterize the density of a connected component? Can we have an intrinsic measure that is invariant to the growth of the connected component? (b) Do connected components have same densities?

For the study, we investigate the connected components of *YahooWeb*. The giant connected component contains 690 million web pages, which is about 50% of the total pages. The second and the third largest connected components contains 57,000 and 21,000 pages which is much smaller than the giant connected component. Except the isolated(=one node) connected components, there are 2.6 million connected components. Our goal is to find interesting patterns on the so-many connected components of different sizes to better understand the evolution of networks.

We first characterize the density of connected components. Many different definitions can be given: the most intuitive one is arguably the ratio of the size (number of edges) and the order (number of nodes) of the graph. However, [Leskovec et al., 2005b] states that there is a power-law relationship between the size and the order of the whole graph, and the exponent remains constant over time. Therefore, scaling it by log is more natural and thus we propose the Graph Fractal Dimension (GFD) to characterize the "density" of a connected component.

Definition 1 (Graph Fractal Dimension) *The graph fractal dimension GFD(C) of a connected component C is the ratio of the size and the order in log scale. That is,* $GFD(C) = \frac{log|E(C)|}{log|V(C)|}$.

For example, a clique will have GFD ≈ 2, because there are $n^2 - n$ edges for n nodes. A chain will have GFD ≈ 1, because there are $n - 1$ edges for n nodes. Several graphs and their fractal dimensions are shown in Figure 3.5.

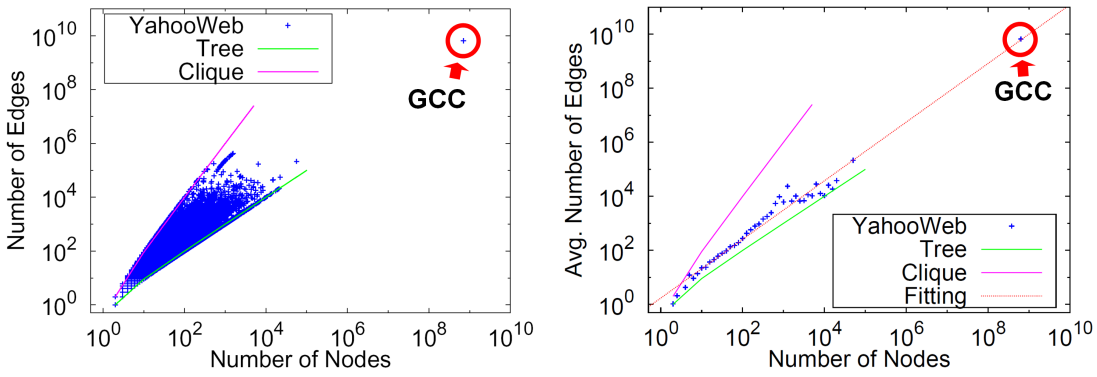


(a) Chain: GFD 1.27 (b) Star: GFD 1.27 (c) Bipartite-Core: GFD 1.64 (d) Clique: GFD 1.94

Figure 3.5: Graphs, adjacency matrices and their fractal dimensions. Notice that the GFD can be used as a measure of density of graphs: chain or star graphs have smaller GFDs while cliques have higher GFDs.

Graph Fractal Dimension

How are the graph fractal dimensions of connected components distributed? Do they share regularities? Figure 3.6 shows the graph fractal dimension of connected components in *YahooWeb* graph. In Figure 3.6 (a), there exists various components with wide range of number of edges for



(a) $|E|$ vs. $|V|$ (b) Avg. $|E|$ vs. $|V|$

Figure 3.6: Homogeneity in the fractal dimension of components in *YahooWeb*. **(a)** Number of edges vs. number of nodes. Each point corresponds to a connected component. **(b)** Average number of edges of components (y-axis) with the corresponding number of nodes (x-axis). Notice the fractal dimension of components fits in a line.

a fixed number of nodes, between the minimum (tree) and the maximum (clique). However, after averaging the number of edges in Figure 3.6 (b), we observe the following striking pattern.

Observation 3.1.4 (Homogeneity of Components GFD) *Graph fractal dimensions of connected components in YahooWeb graph are constant on average.*

This observation implies that on average, the connected components of the Web graph are self-similar, regardless of the size of the network.

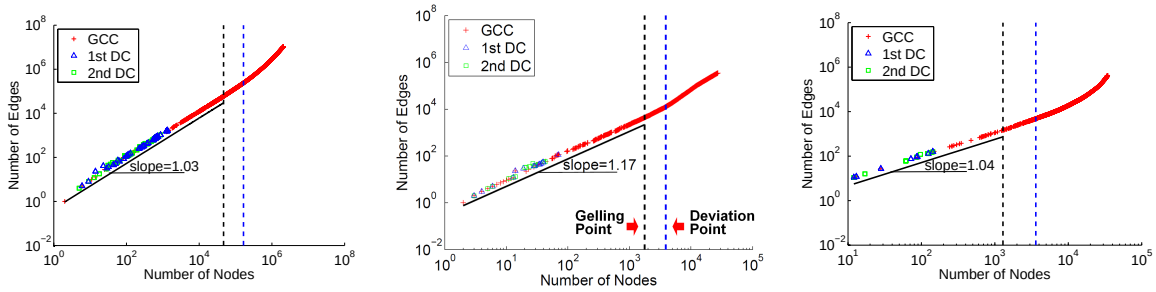
Graph Fractal Dimension over time

Here we broaden our focus to time-evolving graphs and study the dynamic aspects of the connected components. The main questions is: Will the connected components grow with the same rate? Will there be a change of growth rate around the gelling' point where the diameter of the graph starts to shrink?

To answer the above question, we look at the graph fractal dimension of top 3 largest connected components over time in Figure 3.7 and summarize findings in the following observation.

Observation 3.1.5 (Evolution of Top 3 Connected Components) *The giant connected component and two next-largest connected components grow with the same rate. Their graph fractal dimensions remain the same until a deviation point. The deviation point is close to the "gelling" point where the diameter starts to shrink.*

This observation is interesting, since it implies that some barriers between the nodes seem to collapse after the gelling point, and the nodes in the network are connected with higher rate than before the gelling point.



(a) *Patent*: Top 3 CC (b) *HEP-TH*: Top 3 CC (c) *HEP-PH*: Top 3 CC

Figure 3.7: Growth of connected components in terms of the graph fractal dimension. Each point represents the snapshot of a connected component over time. Notice that the slope remains constant until a "deviation" point (the second vertical line) close to a "gelling" point (the first vertical line), and starts to increase after that. The deviation points are about one year after the gelling points.

3.1.5 Pattern DU-6: Exponential "Rebel" Probability to NLCCs

Given a newcomer to a network, what is the probability that it will be absorbed, or not absorbed to the GCC? We call it as the "rebel" probability and give its relations to the degree of newcomers and the portion of nodes in the NLCCs.

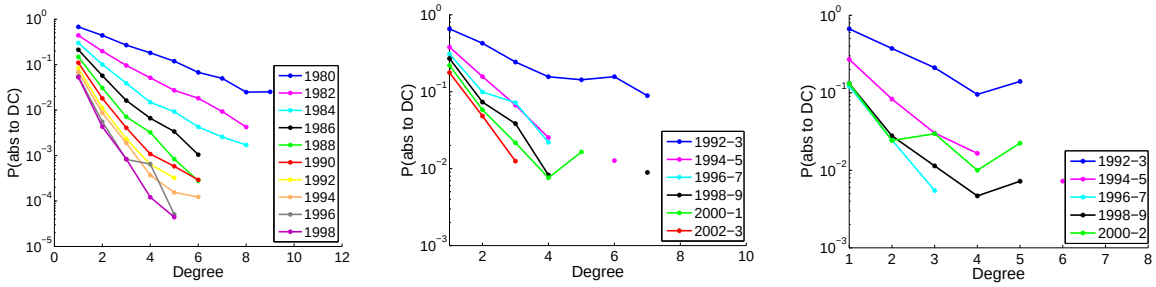
We first give the relationship of the rebel probability and the degree of newcomers in Figure 3.8. In the figure, we see that the probability is linear to the degree in log-lin scale where the slope decreases as the network grows. In addition, we show the relationship of the rebel probability and the portion of nodes in the NLCC in Figure 3.9. From Figure 3.9, we see the probability is linear to the portion of nodes in NLCC in log-log scale, and the slope increases as the degree increases. Given these two observations, we give empirical rebel probability of newcomers as a function of the degree(d) and the portion(s) of nodes in the NLCC in the following observation, which we call the ERP (Exponential Rebel Probability) pattern.

Observation 3.1.6 (Exponential Rebel Probability (ERP)) *Given the node portion s of NLCCs, the probability P_{rebel} of a newcomer to be absorbed in NLCCs is exponential to the product of a constant α, the degree d of the newcomer, and the log of the node portion s of NLCCs:*

$$P_{rebel} \propto e^{\alpha d(logs)} \tag{3.1}$$

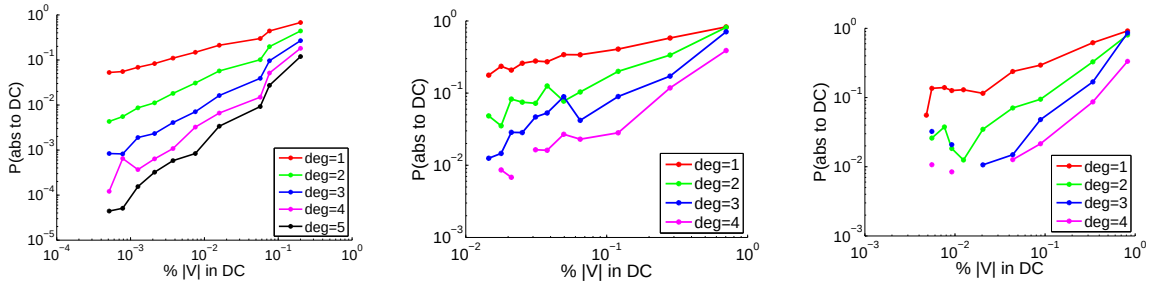
3.1.6 Pattern DU-7: Principal Eigenvalue over time

How does the principal eigenvalue of a graph behave over time? The principal eigenvalue is the largest eigenvalue associated with the adjacency matrix of the graph, and is also described as a measure of overall connection strength of a connectivity matrix. It has also been shown that the epidemic threshold in graphs depends only on this principal eigenvalue, and nothing else [Prakash et al., 2010; Valler et al., 2011].



(a) *Patent* (b) *HEP-TH* (c) *HEP-PH*

Figure 3.8: P(Absorption to NLCCs) vs. Degree in log-lin scale. Notice the linear drop of the probability as the degree increases.



(a) *Patent* (b) *HEP-TH* (c) *HEP-PH*

Figure 3.9: P(Absorption to NLCCs) vs. Portion of Nodes in NLCCs in log-log scale. Notice that the slopes of curves increase as degree increases.

Plotting the largest (principal) eigenvalue of the 0-1 adjacency matrix of our datasets over time, we notice that the principal eigenvalue grows following a power law with increasing number of edges. This observation is true especially after the gelling point. As we discussed before, gelling point is defined to be the point at which a giant connected component (GCC) appears in real-world graphs– after this point, properties such as densification and shrinking diameter become increasingly evident.

Observation 3.1.7 (λ_1 Power Law (LPL)) *In real graphs, the principal eigenvalue $\lambda_1(t)$ and the number of edges $E(t)$ over time follow a power law with exponent less than 0.5, especially after the "gelling" point. That is,*

$$\lambda_1(t) \propto E(t)^{\alpha},\ \alpha \leq 0.5$$

We report the power law exponents in Fig. 3.10. Note that we fit the given lines *after* the gelling point which is shown by a vertical line for each dataset. Notice that the slopes are less than 0.5, with the exception of the *CampOrg* dataset, which has slope ≈ 0.53.

Given the theorem $\lambda_{max}(\mathcal{G}) \leq \left\{2\left(1-\frac{1}{N}\right)E\right\}^{\frac{1}{2}}$ for a connected, undirected graph $\mathcal{G}$ without self-loops and multiple edges, with E edges and N nodes (see [Wilf, 1967] for proof), for large

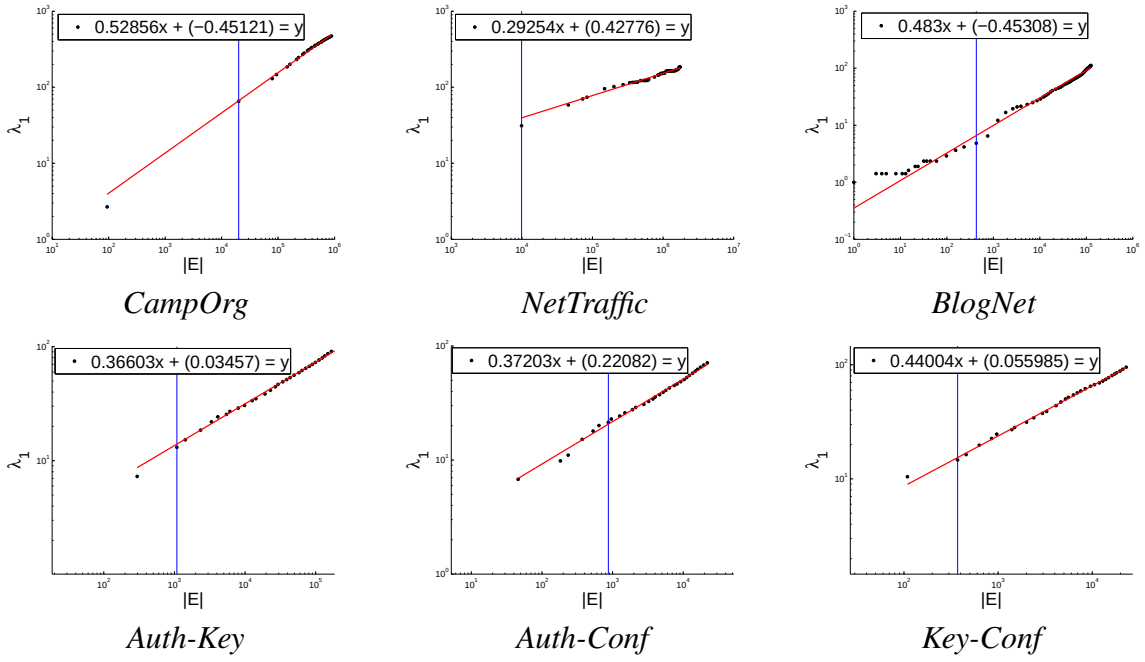


Figure 3.10: Illustration of the LPL (Observation 3.1.7). 1^{st} largest (principal) eigenvalue $\lambda_1(t)$ of the *0-1* adjacency matrix **A** versus number of edges $E(t)$ over time. The vertical lines indicate the gelling point.

N, s.t. $\frac{1}{N} \to 0$, we expect the power law exponent to be less than 0.5. By construction, there are no multiple edges in our graphs (that is, we work with a binary adjacency matrix), and the nodes do not have self-loops by nature. Finally, we claim that the graphs behave as a single connected component after the gelling point at which point the GCC dominates other connected components. The only slight exception to the power law, the *CampOrg* graph, always has many number of disconnected components as well as a GCC. Thus, we conclude that our observation follows early theory.

3.2 Weighted graph patterns

3.2.1 Pattern SW-1: Edge Weights Power Law (EWPL)

We observe that the weight of a given edge and weights of its neighboring two nodes are correlated. Our observation is similar to Newton's Gravitational Law stating that the gravitational force between two point masses is proportional to the product of the masses. Similarly, the tendency of two nodes to interact often would be related to the "popularity" of both.

For each edge (i, j) at the final time step in the graph, we plot $\sqrt{(w_i - w_{i,j}) * (w_j - w_{i,j})}$ versus its weight $w_{i,j}$ as a single point. Notice that we did not include the weight of the edge itself in the total weight of its incident nodes. Next, we fit a line to the median y-axis values after

applying logarithmic binning on the x-axis and report the corresponding slopes for each dataset in Fig. 3.11. Note that, we omit the points which represent edges to avoid the confusion due to overplotting. Instead, we show the 75% and 25%-tile of the data with upper and lower vertical bars, respectively.

Observation 3.2.1 (Edge Weights Power Law (EWPL)) *Given a real-world graph $\mathcal{G}$, 'communication' defined as the weight of the link between two given nodes has a power law relation with the weights of the nodes. In particular, given an edge $e_{i,j}$ with weight $w_{i,j}$ and its two neighbor nodes i and j with weights w_i and w_j, respectively,*

$$w_{i,j} \propto \left(\sqrt{(w_i - w_{i,j}) * (w_j - w_{i,j})} \right)^{\gamma}$$

EWPL can be used in link prediction; that is, one can estimate the probable weight of a future link between two nodes of the graph, given their weights. Moreover, edges with weights deviating too much from the expected might be flagged for further consideration.

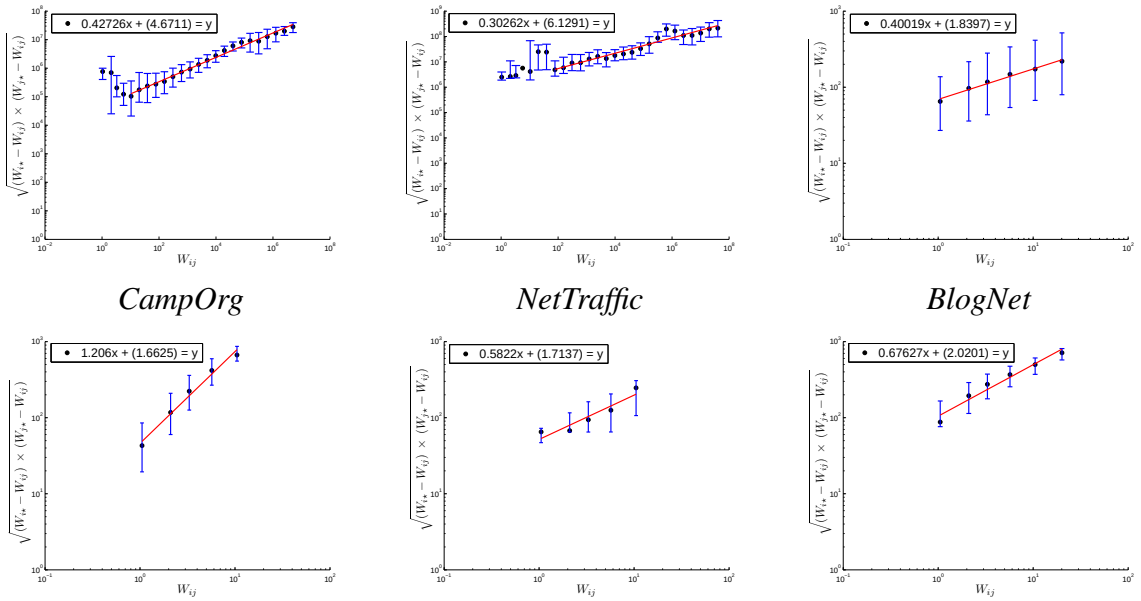


Figure 3.11: Illustration of the EWPL (Observation 3.2.1). Given the weight of a particular edge in the final snapshot of real graphs (x-axis), the multiplication of total weights(y-axis) of the edges incident to two neighboring nodes follow a power law. A line can be fit to the median values after logarithmic binning on the x-axis. Upper and lower bars indicate 75% and 25% of the data, respectively.

3.2.2 Pattern SW-2: Snapshot Power Law (SPL)

Is there any correlation between the in-degree and the in-weight, and between the out-degree and the out-weight, for all the nodes of a graph, at a given time-stamp? If node i has out-degree out_i, what can we say about its out-weight $outw_i$? We find that there is a "fortification effect" here, i.e. superlinear growth, resulting in more power laws, both for out-degrees/out-weights as well as for in-degrees/in-weights.

Specifically, at a given point in time, we plot the scatter-plot of the in/out weight versus the in/out degree, for all the nodes in the graph, at a given time snapshot. An example of such a plot is in Fig. 3.12 (a) and (b). Here, every point represents a node and the x and y coordinates are its degree and total weight, respectively. To achieve a good fit, we bucketize the x axis with logarithmic binning [Newman, 2005], and, for each bin, we compute the median y. We observe that the median values of weights versus mid-points of the intervals follow a power law for all datasets studied. Formally, the "Snapshot Power Law" is:

Observation 3.2.2 (Snapshot Power Law (SPL)) *Consider the i-th node of a weighted graph, at time t, and let out_i, $outw_i$ be its out-degree and out-weight. Then*

$$outw_i \propto out_i^{ow}$$

where ow is the out-weight-exponent *of the SPL. Similarly, for the in-degree, with in-weight-exponent iw.*

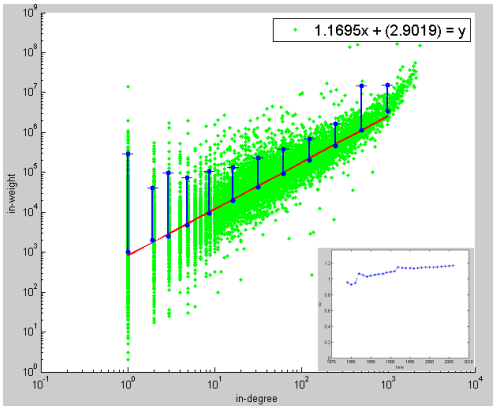
We studied the snapshot plots for several time-stamps (for brevity, we only report the slopes for the final timestamp in Table 3.1 for all the datasets we studied). We observed that SPL exponents of a graph over time remains almost constant. In Fig. 3.12 (a) and (b), the inset plots show how the iw and ow exponent changes over time (years) for the *CampOrg* dataset, respectively. We notice that iw and ow take values in the range [0.9-1.2] and [0.95-1.35], respectively:

Observation 3.2.3 (Persistence of Snapshot Power Law) *The in- and out-exponents iw and ow of the SPL remain about constant, over time.*

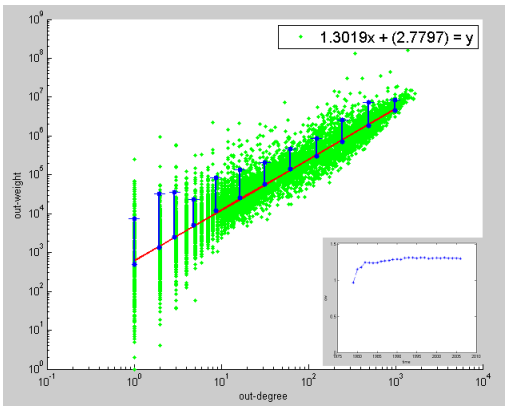
Looking at Table 3.1, we observe that all SPL exponents are > 1, which imply a "*fortification effect*" with super-linear growth. The only exception is the *NetTraffic* dataset. This is explained because the number of nodes N has a limit that can not be exceeded (the total IP addresses at the institution of observation) Until N reaches that point, the slopes are iw=1.19 and ow=1.27 (again, showing a "fortification effect").

3.2.3 Pattern DW-1: Weight Power Law (WPL)

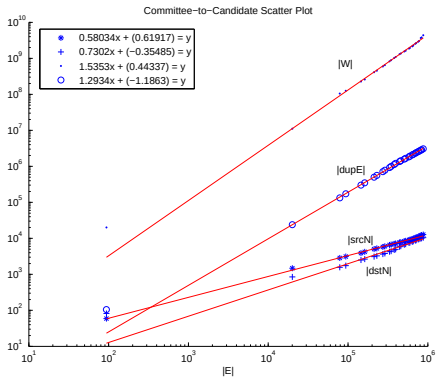
Let $W(t)$ be the total weight up to time t (e.g., the grand total of all exchanged packets in a network), $E(t)$ the number of distinct edges up to time t, and $E_d(t)$ the number of multi-edges (the d subscript stands for *duplicate* edges), up to time t. Is there any correlation between the total weight, the total number of edges and the total number of multi-edges in a graph, as they change over time?



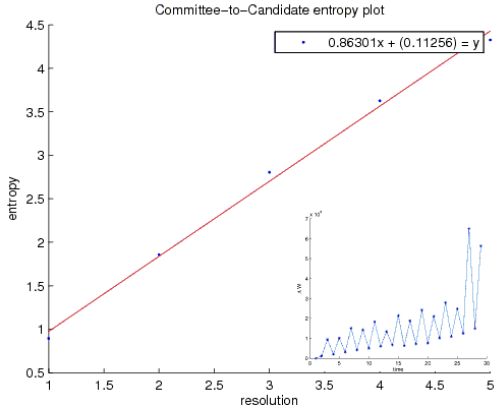
(a) SPL (in-weight vs. in-degree)



(b) SPL (out-weight vs. out-degree)



(c) WPL



(d) entropy plot

Figure 3.12: Weight properties of *CampOrg* donations: SPL plots (a) and (b) have slopes (power-law exponents) > 1 ("fortification effect"), that is, that the more campaigns an organization supports, the superlinearly more money it donates, and similarly, the more donations a candidate gets, the more average amount-per-donation is received. Inset plots show the exponents iw and ow over time, which stay quite stable; (c) shows all the power laws as well as the WPL; the entropy plot has slope ~ 0.86 in (d), indicating bursty weight additions over time.

If every pair generated k packets, the relationships would be linear: if the count of pairs double, the packet count would double, too. This is reasonable, but it doesn't happen! In reality, the packet count over-doubles, following the "WPL" below. In other words, we observe a "*fortification effect*" here, too: more edges in the graph imply super-linearly higher total weight.

Observation 3.2.4 (Weight Power Law (WPL)) *Let $E(t)$, $W(t)$ be the number of edges and total weight of a graph, at time t. They, they follow a power law*

$$W(t) = E(t)^w$$

where w is the weight *exponent. Power-laws also link the number of nodes $N(t)$, and the number of multi-edges $E_d(t)$, to $E(t)$, with exponents n and $dupE$, respectively.*

	w	*nsrc*	*ndst*	*dupE*	*iw*	*ow*	*fd*
CampOrg	1.53	0.58	0.73	1.29	1.16	1.30	0.86
CampIndiv	1.36	0.53	0.92	1.14	1.05	1.48	0.87
NetTraffic	1.48	0.57	N/A	1.29	0.66	0.45	0.59
BlogNet	1.03	0.79	N/A	N/A	1.01	1.10	0.96
Auth-Key	1.01	0.90	0.70	N/A	1.01	1.04	0.95
Auth-Conf	1.08	0.96	0.48	N/A	1.04	1.81	0.96
Key-Conf	1.22	0.85	0.54	N/A	1.26	2.14	0.95

Table 3.1: Power law exponents for all the weighted datasets we studied: The x-axis being the number of non-duplicate edges E, *w*: WPL exponent, *nsrc, ndst*: WPL exponent for source and destination nodes respectively (if the graph is unipartite, then *nsrc* is the number of all nodes), *dupE*: exponent for multi-edges, *iw, ow*: SPL exponents for indegree and outdegree of nodes, respectively. Exponents above 1 indicate fortification/superlinear growth. Last column, *fd*: slope of the entropy plots, or information fractal dimension. Lower fd means more burstiness.

The weight exponent w ranges from 1.01 to 1.5 for the real graphs we have studied. The highest value corresponds to campaign donations: super-active organizations that support many campaigns also tend to spend even more money per campaign than the less active organizations. For bipartite graphs, we show the $nsrc$, $ndst$ exponents for the source and destination nodes (which also follow power laws: $N_{src}(t) = E(t)^{nsrc}$ and similarly for $N_{dst}(t)$).

Fig. 3.12 (c) shows the WPL for our example dataset *CampOrg*. Other datasets are shown in Fig. 3.13. The plots are in log-log scales. We report the slopes in Table 3.1.

3.2.4 Pattern DW-2: Bursty/Self-similar weight additions

We tracked how much weight a graph puts on at each time interval and looking at the entropy plots, we observed that the weight additions over time show self-similarity. For those weighted graphs where the edge weight is defined as the number of recurrences of that edge, the slope of the entropy plot was greater than 0.95, pointing out uniformity. On the other hand, for those graphs where weight is not in terms of multiple edges but some other feature of the dataset such as the amount of donations for the FEC dataset, we observed that weight additions are more bursty, the slope being as low as 0.6 for the Network Traffic dataset.

Observation 3.2.5 (Bursty/self-similar weight additions) *In all our graphs, the addition of weight ($\Delta W(t)$) was self-similar, with fractal dimension ranging from ≈1 (smooth/uniform), down to* 0.6 *(bursty).*

Fig. 3.12 (d) shows the entropy plot for our example dataset *CampOrg*. Other datasets are shown in Fig. 3.14. ΔW values over time are also shown in insets at the bottom right corner of each figure. We report the information fractal dimensions (=slopes in entropy plots) in Table 3.1.

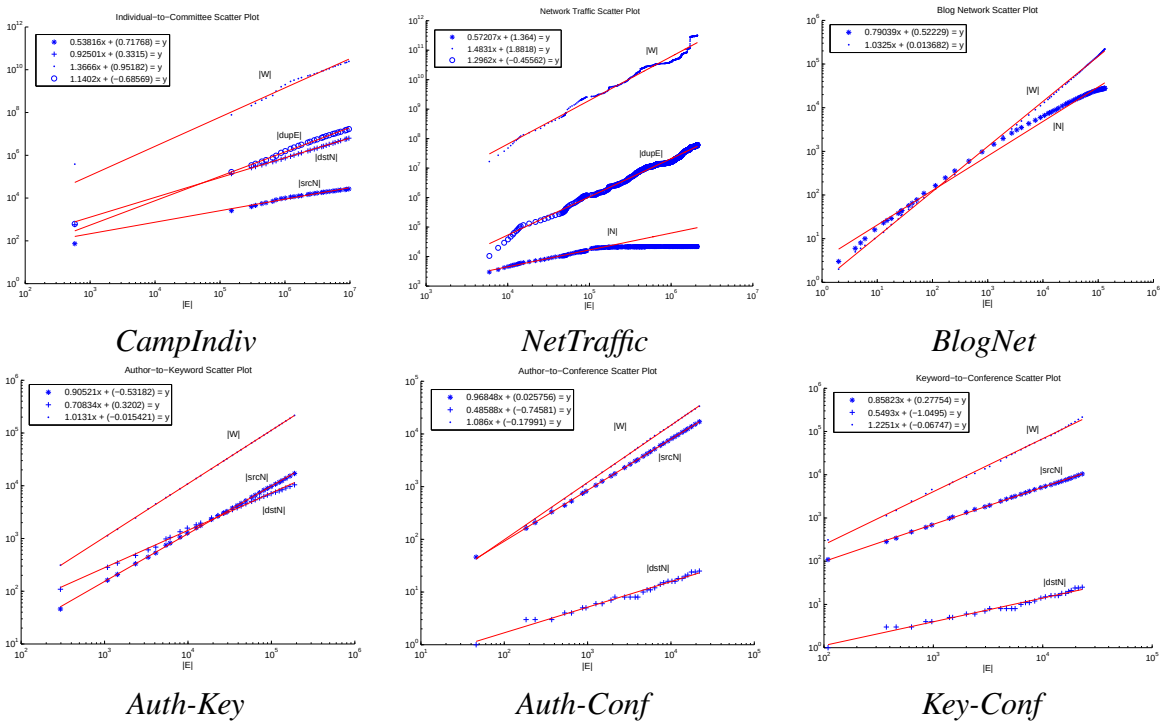


Figure 3.13: Properties of weighted networks. Weight power laws: total weight W, number of duplicate edges E_d, number of nodes N; each versus number of non-duplicate edges E. The slopes for weight W and multi-edges E_d are above 1, indicating "fortification".

3.2.5 Pattern DW-3: Weighted principal eigenvalue over time

Given that unweighted (0-1) graphs follow the λ_1 Power Law, one may ask if there is a corresponding law for weighted graphs. To this end, we also compute the largest eigenvalue $\lambda_{1,w}$ of the *weighted* adjacency matrix $\mathbf{A}_w$. The entries $w_{i,j}$ of $\mathbf{A}_w$ now represent the actual edge weight between node i and j. We notice that $\lambda_{1,w}$ increases with increasing number of edges following a power law with a higher exponent than that of its λ_1 Power Law. We show the experimental results in Fig. 3.15.

Observation 3.2.6 ($\lambda_{1,w}$ Power Law (LWPL)) *Weighted real graphs exhibit a power law for the largest eigenvalue of the weighted adjacency matrix $\lambda_{1,w}(t)$ and the number of edges $E(t)$ over time. That is,*

$$\lambda_{1,w}(t) \propto E(t)^{\beta}$$

In our experiments, the exponent β ranged from 0.5 to 1.6.

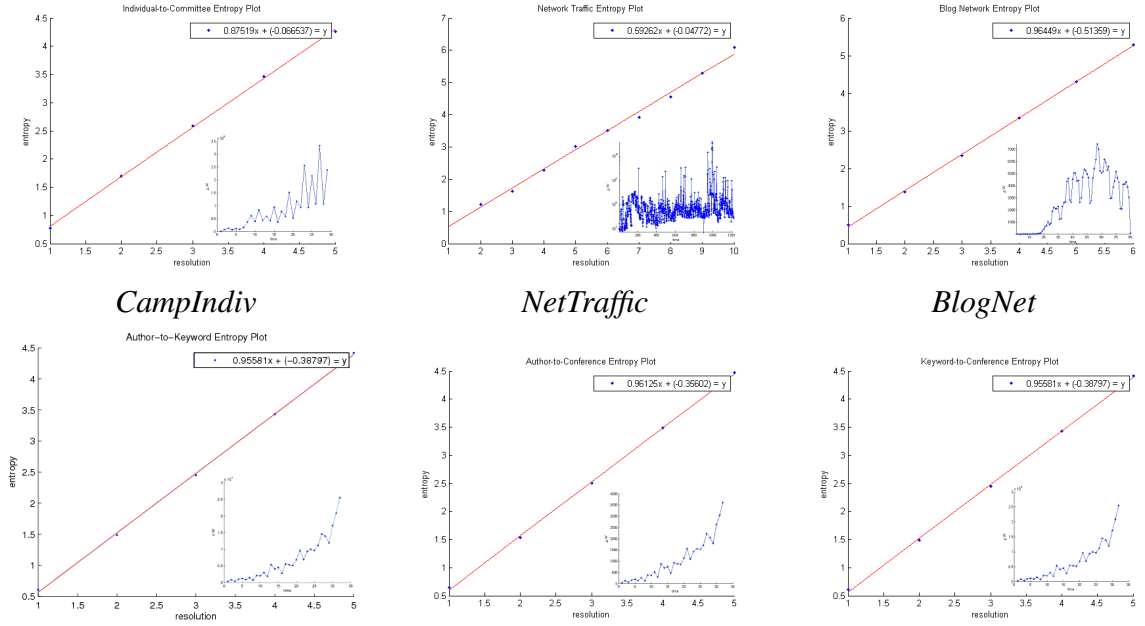


Figure 3.14: Properties of weighted networks. Entropy plots for weight addition. Slope away from 1 indicates burstiness (e.g., 0.59 for *NetTraffic*) The inset plots show the corresponding time sequence ΔW versus time. Notice how bursty *NetTraffic* looks.

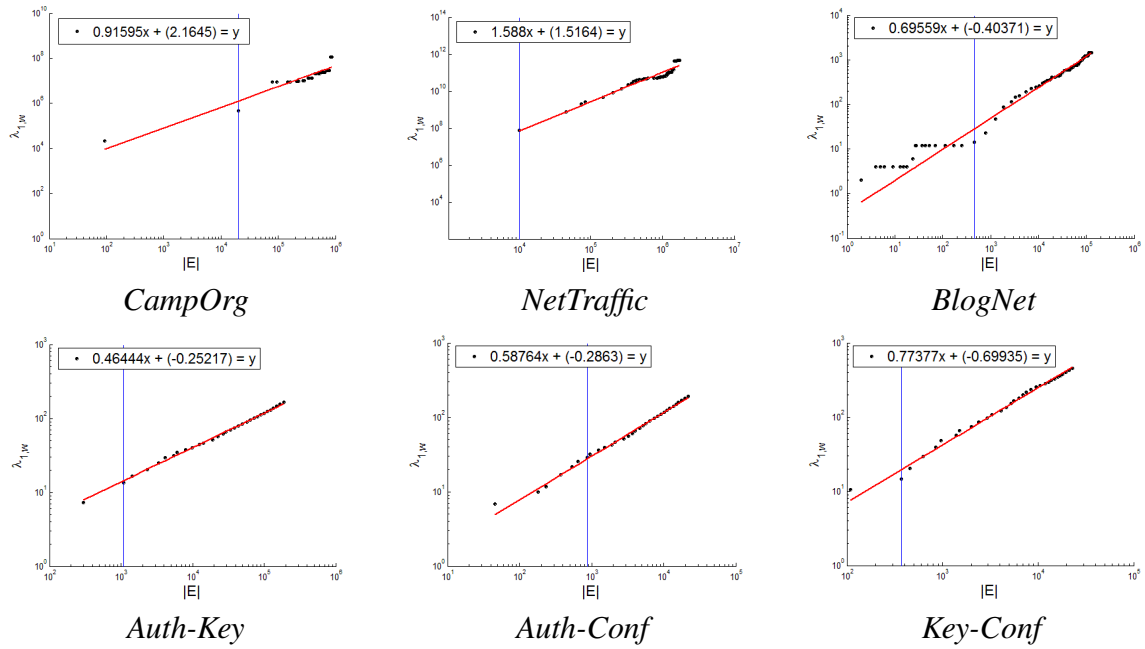


Figure 3.15: Illustration of the LWPL (Observation 3.2.6). 1^{st} largest (principal) eigenvalue $\lambda_{1,w}(t)$ of the *weighted* adjacency matrix $\mathbf{A}_w$ versus number of edges $E(t)$ over time. The vertical lines indicate the gelling point.

3.3 Summary of contributions

The vast majority of previous work on the study of networks focused on *static* networks, with findings such as heavy-tailed degree distributions and small diameter. In this chapter, we described *two novel aspects* of this thesis in the study of real-world networks: (1) *dynamic* networks and (2) *weighted* networks, with interesting new findings (see Table 2.2 for an overview).

More specifically, while previous work focused on the single giant connected component of networks, we shifted our focus to the next-largest connected components and their formation and evolution. We analyzed the structural properties of these smaller components such as their intrinsic dimension, as well as their temporal properties such as how they grow and get merged to the giant component over time. We further studied the behavior of the newcomer nodes and formulated how their *microscopic* dynamics ("rebel" probability) explain the properties observed on the *macroscopic* level (components oscillating around constant-size).

Moreover, we started the study of *weighted* networks where the recurrences of interactions (e.g. multiple emails) or natural weights associated with interactions (e.g. dollar amount donations) are taken into account. We analyzed the correlation of edge weights with other network properties as well as the additions of weights to the network over time.

In summary, our contributions described in this chapter are the following:

- *Study of numerous real-world datasets:* We studied numerous real-world networks from many diverse domains including political campaign donations, computer network traffic, blog citations, social and Web networks. We discovered several new patterns in *dynamic* and *weighted* graphs.
- *"Rebel probability" and other dynamic component patterns:* We focused on the rarely-studied next-largest connected components (NLCCs) and their dynamics. In particular, we showed that (1) NLCCs look like chain graphs; (2) real graphs exhibit a "gelling" point at which the small components merge and a giant connected component (GCC) emerges; (3) after the gelling point, secondary and tertiary components cannot grow beyond a certain size beyond which they get absorbed to the GCC; (4) *graph fractal dimension* of components is stable over time; and (5) *rebel probability* —that a newcomer node will *not* join the GCC— drops exponentially with its degree and increases linearly with the fraction-size of NLCCs.
- *"Fortification effect" and other weighted network patterns:* We focused on the not-studied weights on graph edges and their dynamics. In particular, we showed that (1) total weight of a node and its degree follow power laws, and weight between two nodes follows a gravitational-force-like relation with the total weights of those nodes; (2) total weight of a graph grows *superlinearly* with number of its edges over time; (3) weight additions over time are bursty; and (4) principal eigenvalues and number of edges of a graph follow a power law relation over time.

Chapter 4

Patterns in human communications

PROBLEM STATEMENT: *What are the typical patterns in human communications? How large are our social circles? How reciprocal are we, and what does it depend on? How long are our phone calls, and how can we summarize their distribution?*

In this chapter, we present our findings in millions of communication records, including phone calls, short, and instant messages, of millions of users. In particular we study the social circles (=maximal cliques) people belong to, their reciprocity behavior, as well as call durations.

4.1 Patterns in social circles

4.1.1 Motivation

What patterns should we expect in a network of human-to-human interactions? How can we spot anomalies (e.g., tele-marketers, spammers)? Related applications are numerous and almost everywhere in people's modern life. Online social networks, like Facebook (`www.facebook.com`) and LinkedIn (`www.linkedin.com`), mimic publicly the telecommunication networks where and what people communicate privately. Product recommendation systems, such as Amazon (`www.amazon.com`) and Netflix (`www.netflix.com`), rely on a network of trust and collaboration. Computer networks have predictable relations regarding intrusion detection, security, and virus propagation. It is important in all the above applications to spot anomalies and outliers. Anomaly detection is tightly connected to patterns: if most of the nodes in our network closely follow a pattern, then the few deviations that do exist are probably outliers.

In this section, we are investigating the following questions:

- When we isolate the cliques in a network, what patterns do they follow? How large are our social circles on average? If someone has many contacts, does that indicate popularity?

- What patterns do the edge weights follow, both in triangles and in general cliques? Specifically, in a triangle, all three nodes are equivalent in topology, but is it normal if all three weights are equal as well?

4.1.2 Data Description

The datasets analyzed are built from a large collection of records from several human communication services including voice, data, IM, SMS etc. Each record is represented as a triple $< ID_i, ID_j, Time >$, where $< ID_i >$ and $< ID_j >$ are generally referred to as the *caller* and *callee*. During a particular time period, there can be multiple times for a pair of people to communicate with each other, and the accumulated number of communication times between ID_i and ID_j is defined as the edge weight between node i and j. We have the weighted graphs extracted from the records of three types of services ($S1$, $S2$ and $S3$), referred to as $\mathcal{G}^{S1}$, $\mathcal{G}^{S2}$, and $\mathcal{G}^{S3}$ respectively. Each type of service has on average about 1 million records, which were collected by different geographic locations. Apart from this spatial diversity and the service type variety, we also incorporate temporal diversity by collecting data for each type of service during five consecutive time periods represented from $T1$ to $T5$, so $\mathcal{G}^{S1}_{T1}$ is the graph of service type $S1$ in time period $T1$, and $\mathcal{G}^{S2}_{T5}$ is the graph of service type $S2$ in time period $T5$ etc.

Notice that we only focus on the link between the caller and callee. It is important to know that our work is only an aggregate statistical analysis, and therefore, we do not study any individual's behavior from any specific type of communication service. More importantly, any information that could identify users is stripped to access. We only use the encrypted user id in this study, and restrict our interest only in the statistical findings that are held within the networks.

We note that unlike some artificial social networks, such as the scientific collaboration network which emerges as a one-mode projection of the bipartite graph between authors and papers, the massive anonymized human communication networks are formed from the real-time direct contact events of people. They can fully capture the underlying realistic social structures, and lay a solid foundation for our upcoming work.

4.1.3 Patterns and Observations

We report three newly discovered patterns that our datasets seem to follow, and discuss the potential ways in which they can be utilized. The first is *Clique-Degree Power-Law* (CDPL), correlating the ith largest degree with the average number of maximal cliques, which seems to remain rather stable over time so that we trust them to further detect outliers and spot anomalies. The second is *Clique Participation Law* (CPL), which gives the distribution of the number of maximal cliques that each node participates in. Finally, the third comes *Triangle Weight Law* (TWL), describing how the weights are distributed on the edges of triangles, based on which we could further make predictions about the missing values of the edge weights in time-evolving weighted networks.

Clique-Degree Power-Law

As defined previously, $\forall v_i \in V\mathcal{G}$, d_i is the number of all the partners that v_i has, and $C(v_i)$ represents the set of all the maximal cliques that v_i participates in. Is there any relationship between d_i and $|C(v_i)|$? We can imagine that if a particular user has doubled his partners, it tends to be easier for him to participate in doubled social circles as well. This kind of relationship seems to be linear, and sounds reasonable. However, this is often not the case. For our real world social networks, the number of social circles actually over-doubles by following a *Clique-Degree Power-Law*.

Figure 4.1 plots the number of partners vs. the number of maximal cliques averaged over all the nodes with that many of partners, from $T1$ to $T5$. The result is surprising because for any given node, the clique-participation is super-linearly related to its degree. In addition, we notice that the exponent takes values in the range $[1.84, 1.88]$, $[2.04, 2.21]$ and $[1.41, 1.58]$ for G^{S1}, G^{S2}, and G^{S3}, which seems to be stable over time.

Observation 4.1.1 (Clique-Degree Power-Law (CDPL)) *The number of maximal cliques that a node participates in, is super-linearly related to its degree. Given d_i and $C_{avg}^{d_i}$, they follow a power-law :*

$$C_{avg}^{d_i} \sim d_i^{\alpha} \tag{4.1}$$

where α is the exponent of CDPL, and remains about constant over time.

The direct application of CDPL is to spot outliers. In Figure 4.1, all of the detected anomalies are marked by red circles. We can see that these points present a clear pattern which does not conform to the established normal behavior. In other words, for these users the actual number of the maximal cliques that they belong to is significantly distant from the one that they should have according the number of their friends.

It is also interesting to notice that some outliers are stable and persistent, such as node v_x and v_y from $\mathcal{G}_{T1}^{S1}$ to $\mathcal{G}_{T3}^{S1}$, while others are more casual and bursty, such as node v_z in $\mathcal{G}_{T5}^{S1}$, and the circled outliers in $\mathcal{G}_{T5}^{S3}$. Figure 4.2 shows the egocentric subgraphs centered with node v_y and v_z, which are composed of the connections among their neighbors in $T5$. Clearly, for node v_y, although it has a large number of partners, it only belongs to few maximal cliques on the left upper part of Figure 4.2 (a). As to node v_z shown in Figure 4.2 (b), almost no connections exist among its partners. Because any automatic customer service id is excluded from our communication networks, the anomalous behavior of v_y and v_z makes them more like the tele-marketers. In fact, there are more outliers in the last time period $T5$ than the others, especially in the network $\mathcal{G}_{T5}^{S3}$ of the third communication service. We guess it is probably because there is actually a big holiday in $T5$, and the third communication service is the cheapest for broadcasting messages.

Clique Participation Law

Based on the discovered maximal cliques, we are able to study how people get involved into them. Figure 4.3 shows the distribution of the number of maximal cliques that people actually

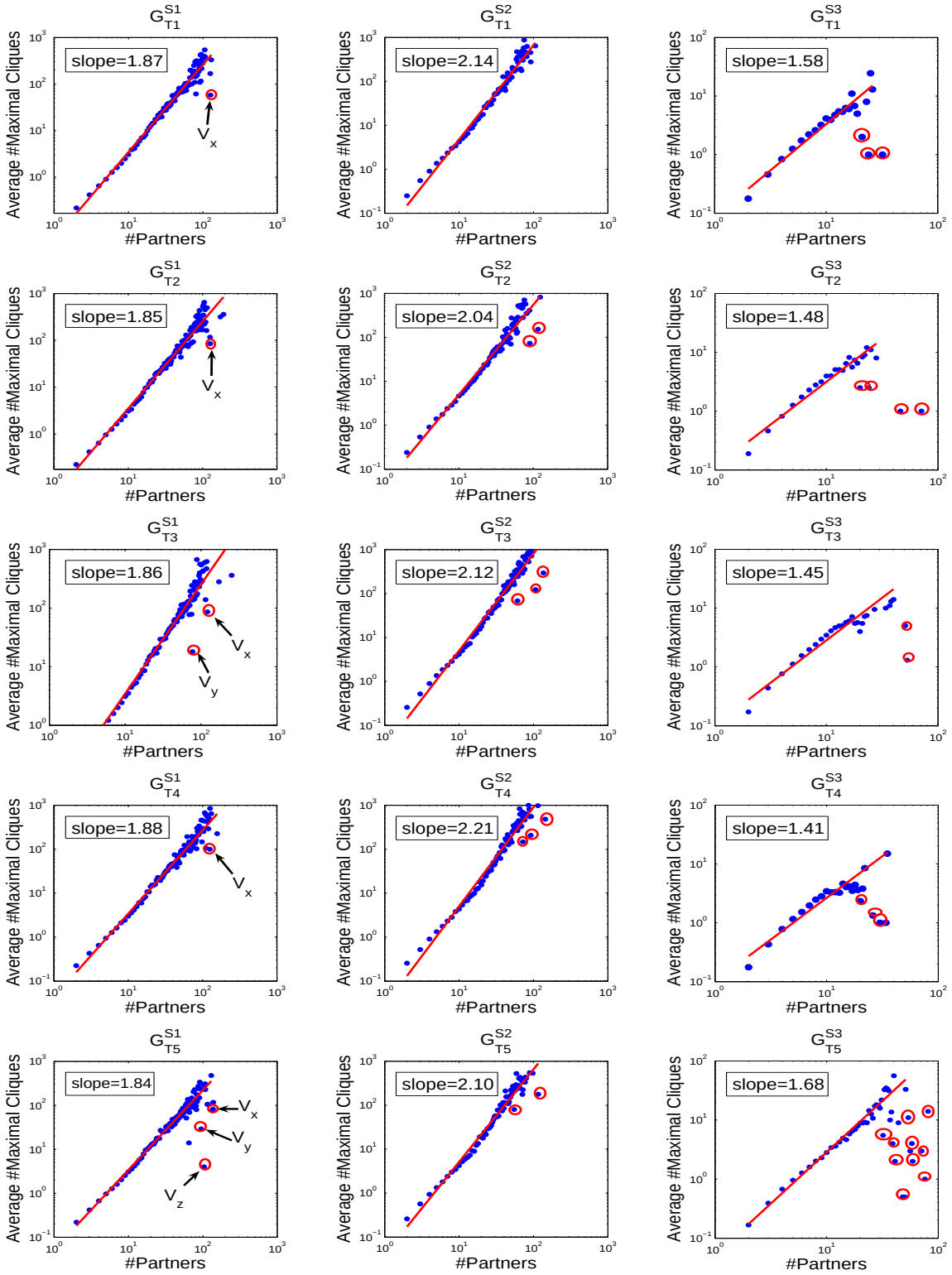
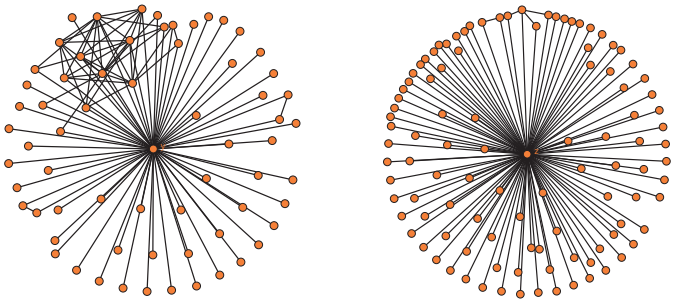


Figure 4.1: Illustration of Clique-Degree Power-Law (Observation 4.1.1). Number of partners vs. the average number of maximal cliques in $G^{S1} \sim G^{S3}$ (rows) from $T1$ to $T5$ (columns). All exponents are fitted with $R^2 > 0.95$. Notice that CDPL is very stable over time. Outliers are marked by red circles.



(a) Centered with vertex v_y (b) Centered with vertex v_z

Figure 4.2: Detected typical outliers. Both v_y and v_z (in $\mathcal{G}_{T3}^{S1}$ and $\mathcal{G}_{T5}^{S1}$ from Figure 4.1) have too many unrelated partners, resulting in a star-like subgraph.

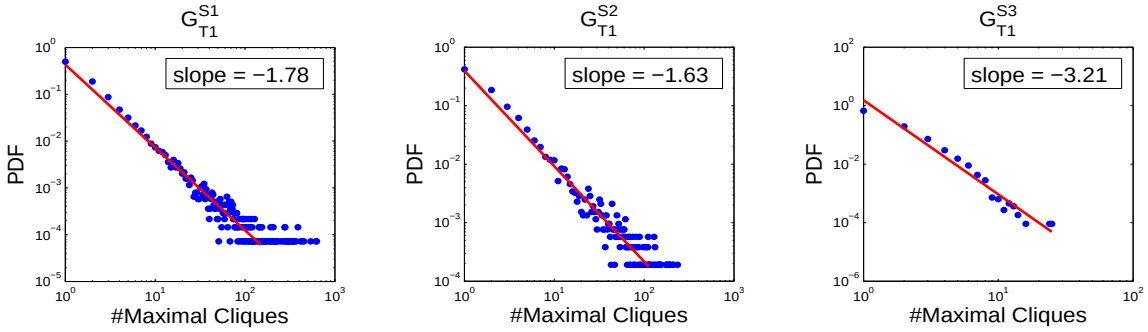


Figure 4.3: Illustration of Clique-Participation Law (Observation 4.1.2). PDF of #Maximal Cliques in $\mathcal{G}_{T1}^{S1} \sim \mathcal{G}_{T1}^{S3}$. The rest graphs behave similarly.

participate in. That is, in graph $\mathcal{G}$, it plots the correlation between the number of maximal cliques (x-axis) and the PDF of nodes (y-axis) that get involved in that many of maximal cliques. We observe that there exists a power-law followed by this kind of relationship, which is called *Clique Participation Law*.

Observation 4.1.2 (Clique Participation Law (CPL)) *For a given number of maximal cliques, say n_{clique}, and the set $V_{clique} = \{v_i | v_i \in V(\mathcal{G}), |C(v_i)| = n_{clique}\}$, we have*

$$n_{clique} \sim |V_{clique}|^{cp} \tag{4.2}$$

where cp is the clique participation exponent of CPL, and keeps about constant over time.

	T1	T2	T3	T4	T5
G^{S1}	-1.78	-1.74	-1.76	-1.70	-1.68
G^{S2}	-1.63	-1.56	-1.52	-1.56	-1.54
G^{S3}	-3.21	-3.50	-3.46	-3.50	-3.01

Table 4.1: Power-Law exponents of CPL in $G^{S1} \sim G^{S3}$ from $T1$ to $T5$. Notice the stability.

According to the above discussion, for most people in real world social networks, they are often involved in a small number of maximal cliques (or social circles). Only a few of them are really "social butterflies" that can actively span many social circles simultaneously. In Figure 4.3, we report the results from $\mathcal{G}^{S1} \sim \mathcal{G}^{S3}$ only in $T1$ for brevity, because in Table 4.1 we observe that CPL is rather stable over time, leading to similar plots in the rest.

Actually, the CPL pattern could be potentially applied to help the operators to make better designed family plans. Because we have a model of the distribution of user behavior to form close-knit groups, we can propose better pricing strategies that charge users differently according to the size of their social circles. For example, in most cases people only belong to one or two cliques, which may be formed by their families or best friends. We can design specific billing plans which are favorable to the communications among members of the same clique who are also the customers of the same operator. Even if our friends are the customers of other operators, we may still like to invite them to join us, because we know that it will be good for all of us. As a result, this could implicitly improve the loyalty of the current users, and may further help to increase the rate at which new customers sign up the plans. Moreover, we can also reward a few loyal users who span multiple social groups, because they might help to achieve a quick market promotion by introducing new products and services to their friends.

Triangle Weight Law

According to the clique definition, each node in a clique has connections with all the other nodes. Although it is very intuitive that all these nodes are equivalent in topology, will this also mean that they could have equally close relationships? In our communication networks, the edge weight w_{ij} gives the total number of contact times between i and j, which is an important indicator to show how intimately they could relate to each other. Since that triangle is the base case of a clique, given any triangle $\{i, j, k\}$, will w_{ij}, w_{ik}, and w_{jk} hold approximately equal values because of the structure equivalence between i, j, and k? Although this intuitive conjecture seems to make sense, we have made very unexpected and striking discoveries in the real social networks, which are described as follows.

Observation 4.1.3 (Triangle Weight Law (TWL)) *For any triangle, let MaxWeight, MidWeight, and MinWeight denote the maximum, medium, and the minimum edge weight respectively. In all our graphs, they follow three power-laws:*

$$MaxWeight \sim MidWeight^{\alpha} \tag{4.3}$$

$$MaxWeight \sim MinWeight^{\beta} \tag{4.4}$$

$$MidWeight \sim MinWeight^{\gamma} \tag{4.5}$$

where α, β, and γ are the power-law exponents which remain constant in weighted time-evolving communication networks.

As a result, for the given triangle $\{i, j, k\}$, rather than being approximately equal, w_{ij}, w_{ik} and w_{jk} are significantly different from each other. Figure 4.4 gives the results from the networks $\mathcal{G}^{S1} \sim \mathcal{G}^{S3}$ in the same time period $T1$. To achieve a good fit, we bucketize the x-axis with

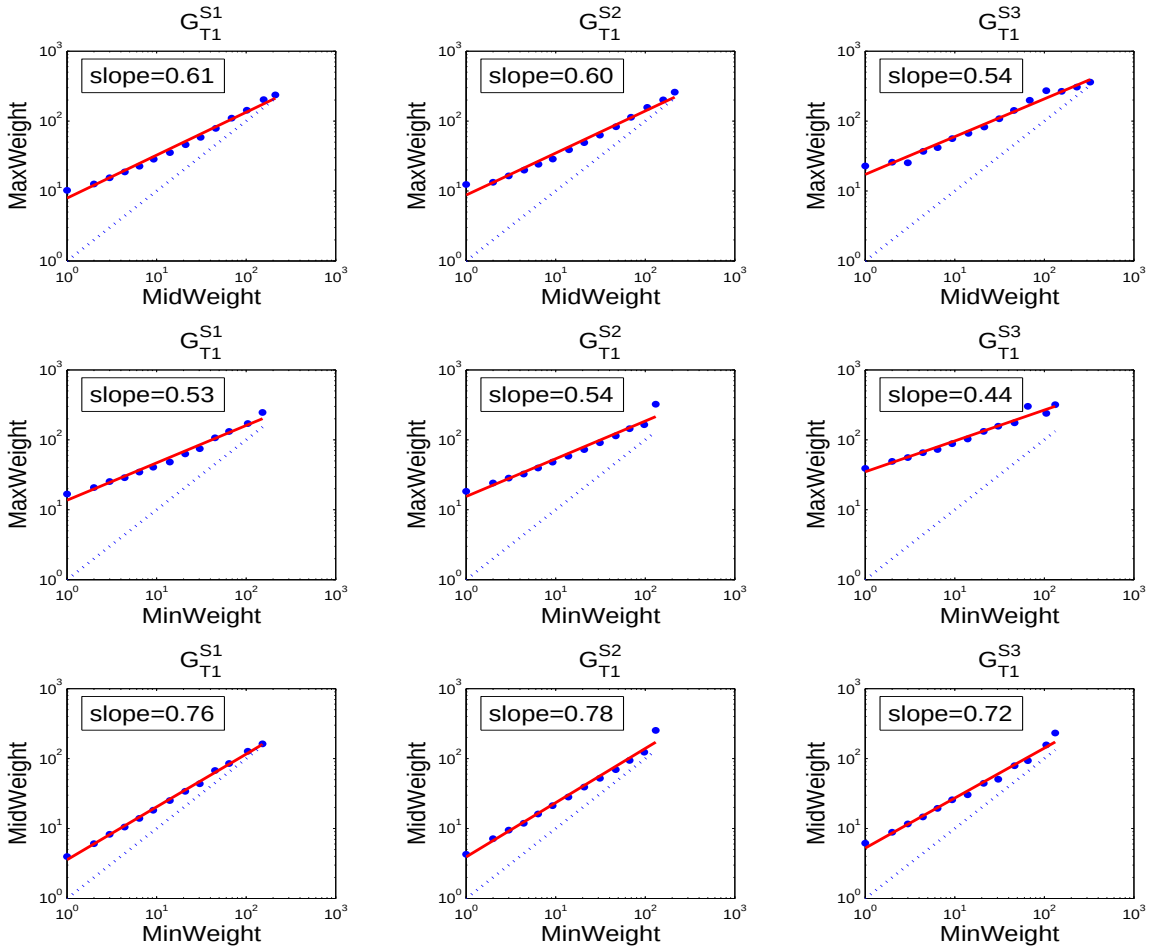


Figure 4.4: Illustration of Triangle Weight Law (Observation 4.1.3). Minimum, medium, and maximum weights in all 3 pairs are plotted in logarithmic scales. Least square fits all have $R^2 > 0.95$ in $\mathcal{G}_{T1}^{S1} \sim \mathcal{G}_{T1}^{S3}$.

logarithmic binning [Newman, 2005], and for each bin, we compute the average value of y. The dotted line means the value of x equals to the value of y.

Moreover, Figure 4.5 shows the three exponents of TWL in $G^{S1} \sim G^{S3}$ from $T1$ to $T5$. Notice that α, β, and γ of these graphs take values in the range [0.5,0.7], [0.4,0.6], and [0.7,0.8], which seem persistent and stable.

In practical situations, due to missing data we can only have partial network information to analyze. For example, in Figure 4.6, given the weighted egocentric subgraph that link e_{23} belongs to, what can we say about the missing w_{23}? While the link prediction tries to predict between which unconnected nodes a link will form next, our problem here concerns how to estimate the value of an edge weight, because we already know there is a link between node 2 and 3. We formulate this problem as the *weight prediction* problem, which not only is important to fill and complete the missing values, but also is useful for discovering anomalous links– if the actual

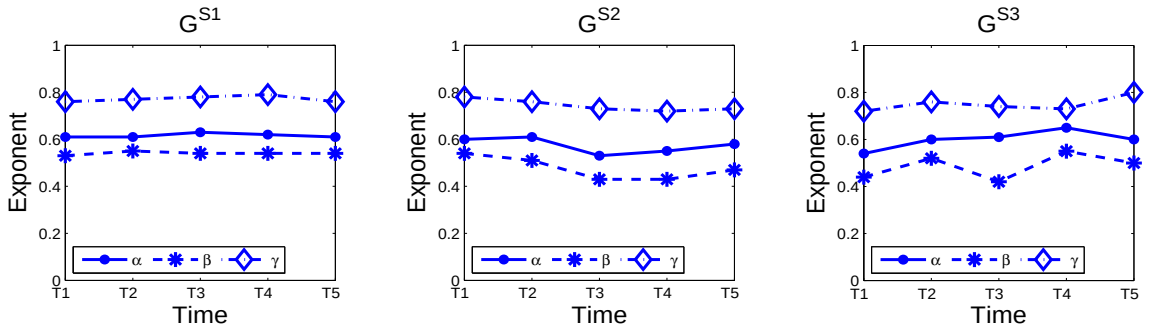


Figure 4.5: Persistence of Triangle Weight Law. Exponent α, β, and γ (red, blue, green) in $G^{S1} \sim G^{S3}$ remain about constant from $T1$ to $T5$.

value of w_{23} is significantly different from the predicted value, it would be highly unusual.

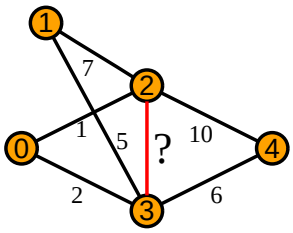


Figure 4.6: Weight Prediction Problem. What can we say about w_{23}?

Based on the above discussion, TWL can help us to solve the weight prediction problem. Formally, given $e_{ij} \in E\mathcal{G}$, let $\triangle$ denote the set of all the edges (excluding e_{ij} itself) of the triangles that e_{ij} belongs to. $\forall e_k \in \triangle$, $w(e_k)$ denotes the weight of e_k. The minimum and maximum values of $w(e_k)$ are represented as $\triangle_{min}$ and $\triangle_{max}$ accordingly. On one hand, if $w_{ij} < \triangle_{min}$ or $w_{ij} > \triangle_{max}$, the numerical relationship between w_{ij} and the weights on the other two edges is determined, so we can use either equation (4) and (5) or (3) and (4) to estimate w_{ij} directly. On the other, if $w_{ij} \in [\triangle_{min}, \triangle_{max}]$, w_{ij} might be the minimum in one triangle, while might be the maximum in another triangle. Thus, for $\forall e_k \in \triangle$, we define $\phi_{(e_{ij},e_k)}(x)$ to represent one of the three equations $(3) \sim (5)$ based on the particular numerical relationship that e_{ij} and e_k could hold. The return value of $\phi_{(e_{ij},e_k)}(x)$ is the estimated weight for edge e_k given the possible value x of w_{ij}. Here, we assume that all edge weights are positive integers. Let w_{min} be the minimum estimated value of w_{ij} when $w_{ij} < \triangle_{min}$, and w_{max} be the maximum estimated value of w_{ij} when $w_{ij} > \triangle_{max}$. Then the optimal value of w_{ij} is:

$$\hat{w}_{ij} = argmin \sum_{e_k \in \triangle} (w(e_k) - \phi_{(e_{ij},e_k)}(x)) \tag{4.6}$$

where $x \in [w_{min}, w_{max}]$. We evaluate this approach in $\mathcal{G}^{S1} \sim \mathcal{G}^{S3}$ by comparing $\hat{w}_{ij}$ with w_{ij} for each edge in $T1$, $T3$ and $T5$. Due to the persistence of TWL we set $\alpha = 1.5$, $\beta = 2.2$, and $\gamma = 1.2$ for $\mathcal{G}^{S1}$; $\alpha = 1.3$, $\beta = 1.7$, and $\gamma = 1.4$ for $\mathcal{G}^{S2}$; $\alpha = 1.4$, $\beta = 2.1$, and $\gamma = 1.3$ for $\mathcal{G}^{S3}$. Let $\epsilon = |\hat{w}_{ij} - w_{ij}|$ denote the prediction error. The the average prediction accuracy of $\epsilon = 0$ (the exact prediction) and $\epsilon = 1$ is around 0.21 and 0.32 accordingly. One problem of this simple method is that it can not predict w_{ij}, if the edge e_{ij} does not belong to any triangle. To solve this problem, and further improve the prediction accuracy is an area of future work.

4.2 Patterns in reciprocity

4.2.1 Motivation

One of the important aspects in human relations is the reciprocity, a.k.a. mutuality. Reciprocity can be defined as the tendency towards forming mutual connections with one another by returning similar acts, such as email and phone calls. In a highly reciprocal relationship both parties share equal interest in keeping up their relationship, while in a relationship with low reciprocity, one person is much more active than the other.

It is important to understand the factors that play role in the formation of reciprocity as there exists evidence that reciprocal relationships are highly probable to persist in the future [Cesar A. Hidalgo, 2008]. Also, [Nguyen et al., 2010] shows that reciprocity related behaviors provide good features for ranking and classification based methods for trust prediction. Reciprocity plays other important roles in social and communication networks. For example, if the network supports a propagation process, such as spreading of viruses in email networks or spreading of information and ideas in social networks, then the presence of mutual links clearly speeds up the propagation. Non-existence of reciprocal links can also reveal unwanted calls and emails in spam detection.

Despite its importance, reciprocity has remained an under-explored dynamic in networks. Most work in network science and social network analysis focus on node level degree distributions [Broder et al., 2000; Faloutsos et al., 1999; Leskovec et al., 2007b], communities [Nussbaum et al., 2010; Satuluri and Parthasarathy, 2009; Tantipathananandh et al., 2007], and triadic relations, such as clustering coefficients and triangle closures [Granovetter, 1973]. The study of *dyadic* relations [Xiang et al., 2010] and the related *bivariate* distributions they introduce is, however, mostly overlooked, and thus is the focus of this section.

The motivation behind our work is grouped into two topics:

M1. *Modeling bivariate distributions in real data*

Two vital components of understanding data at hand are studying the simple distributions in it and visualizing it [Yang et al., 2008]. The study of reciprocity introduces bivariate distributions, such as the distribution $\Pr(w_{ij}, w_{ji})$ of edge weights on mutual edges, where association between *two* quantitative variables needs to be explored. A vast majority of existing work focus on *univariate* distributions in real data such as power-laws [Clauset et al., 2009], log-normals [Bi et al., 2001], and most recently DPLNs [Seshadri et al., 2008]. The study of *multivariate* distributions in real data, however, has very limited focus.

In addition, visualization of multivariate data in 2D is hard and often misleading due to issues regarding over-plotting. More importantly, mere visualization does not provide a compact data representation as opposed to data modeling. Summarization via aggregate functions such as the average or the median loses a lot of information and is also not representative, especially for skewed distributions as found in real data.

Models, on the other hand, provide compact data representations by capturing the patterns in the data, and are ideal tools for applications such as data compression and anomaly detection.

M2. *A weighted approach to reciprocity*

Traditional work [Garlaschelli and Loffredo, 2004] usually study reciprocity on directed, *unweighted* networks as a *global* feature which is quantified as the ratio of the number of mutual links pointing in both directions to the total number of links. Defining reciprocity in such an unweighted fashion, however, prevents understanding the *degree* of reciprocity between mutual dyads. In a weighted network, even though two nodes might have mutual links between them, the skewness and the magnitude of the weights associated with these links would contain more information about how much reciprocity is really there between these nodes. For example, in a phone call network the reciprocity between a mutual dyad where the parties make 80%-20% of their calls respectively is certainly different than that of a mutual dyad with 50%-50% share of their calls. In short, edge weights are crucial to study reciprocity as a property of each dyad rather than as a global feature of the entire network and give more insight into the *level* of mutuality.

In this work, we analyze phone call and SMS records of 1.87 million mobile phone users from a large city collected over six months. The data consists of over half a billion phone calls and more than 60 million SMSs exchanged. Our contributions are:

1. We observe similar bivariate distributions $\Pr(w_{ij}, w_{ji})$ of mutual edge weights in the communication networks we study. We propose the Triple Power Law (3PL) function to model this observed pattern and show that 3PL fits the real data with millions of points very well. We statistically demonstrate that 3PL provides better fits than the well-known Bivariate Pareto and Bivariate Yule distributions. We also use 3PL to spot anomalies, such as a pair of users with low mutuality where one of the parties makes 99% of the calls during the entire working hours, non-stop.
2. We use weighted measures of reciprocity in order to quantify the degree of reciprocal relations and study the correlations between reciprocity and local topological features among user pairs. Our results suggest that mutual users with larger local network overlap and higher degree similarity exhibit greater reciprocity.

To the best of our knowledge, this is one of the few work on modeling bivariate distributions in real data as well as one of the few work that takes a weighted approach to study the strength of reciprocity.

4.2.2 Data Description

In this work, we study anonymous mobile communication records of millions of users collected over a period of six months, December 1, 2007 through May 31, 2008. The data set contains both phone call and SMS interactions.

From the whole six months' of activity, we build three networks, CALL-N, CALL-D, and SMS, in which nodes represent users and directed edges represent phone call and SMS interactions between these users. CALL-N is a who-calls-whom network with edge weights denoting (1)

Network	N	E	W_N	$W_D(min.)$	Network	N	E	W_{SMS}
CALL	1,87M	49,50M	483,7M	915x10^6	SMS	1,87M	8,80M	60,5M
CALL (m)	1,75M	41,84M	468,7M	885x10^6	SMS (m)	0,58M	2,10M	46,6M

Table 4.2: Data statistics. The number of nodes N, the number of directed edges E, and the total weight W in the mutual (m) and non-mutual CALL and SMS networks.

total number of phone calls, CALL-D is the same who-calls-whom network with edge weights denoting (2) total duration of phone calls (aggregated in minutes), and SMS is a who-texts-whom network with edge weights denoting (3) total number of SMSs. Table 4.2 gives the data statistics. Global unweighted reciprocity is r=0.84 for CALL, and r=0.24 for SMS.

When only mutual edges are considered, notice that the SMS network shrinks considerably (with only about one forth of the edges E remaining), whereas CALL networks remain almost intact. The drop in the total weight W, on the other hand, is small for all three networks. This shows that the majority of SMS interactions are lop-sided, however the lop-sided interactions constitute only a small fraction of the total volume.

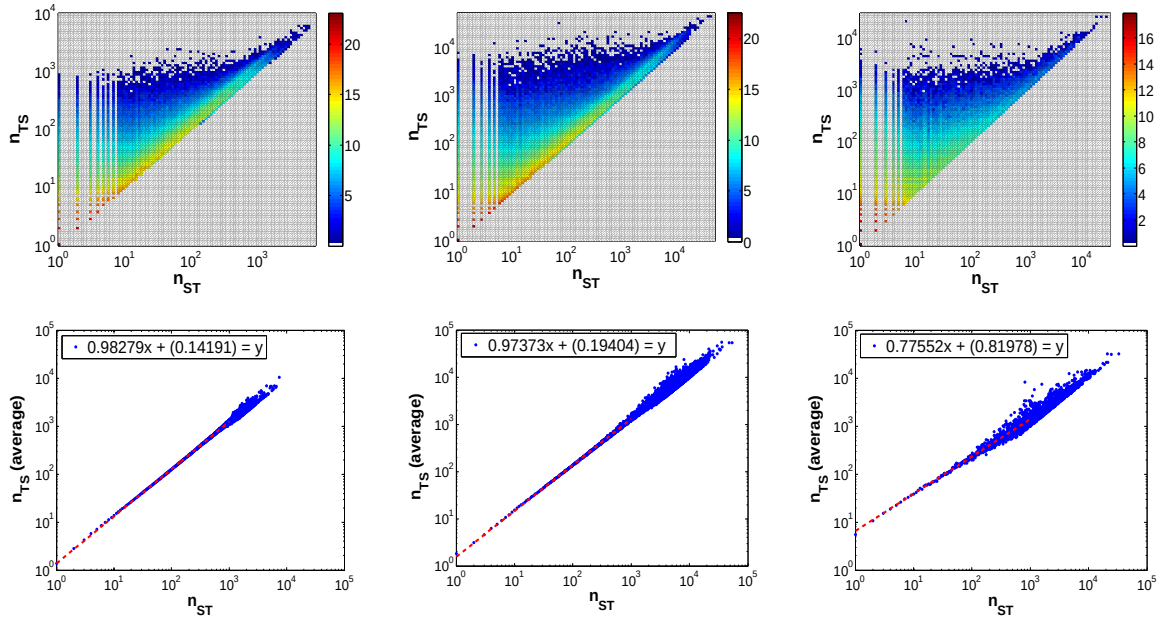
4.2.3 Patterns and Proposed 3PL Model

Given a network of users with *mutual*, *weighted* edges between them, say CALL-N, and given two users i and j in the network, is there a relation between the number of calls i makes to j (w_{ij}) and the number of calls j makes to i (w_{ji})? In this section, we want to understand the association between the weights on the reciprocal edges in human communication networks and study their distribution $\Pr(w_{ij}, w_{ji})$ across mutual dyads. Since we study the pair-wise joint distribution, the order of the weights do not matter. Thus, to ease notation, we will denote the smaller of these weights as n_{ST} (for weight from Silent-to-Talkative) and the larger as n_{TS}, and will study $\Pr(n_{ST}, n_{TS})$.

Figure 4.7 (top-row) shows the weights n_{TS} versus n_{ST} for all the reciprocal edges in (from left to right) CALL-N, CALL-D, and SMS. Each dot in the plots corresponds to a pair of mutual edges. Since there could be several pairs with the same (n_{ST}, n_{TS}) weights, the regular scatter plot of the reciprocal edge weights would result in over-plotting. Therefore, in order to make the densities of the regions clear, we show the *heatmap* of the scatter plots where colors represent the magnitude of volume (red means high volume and blue means low volume).

In Figure 4.7, we observe that most of the points are concentrated (1) around the origin and (2) along the diagonal for all three networks. Concentration around the origin, for example in CALL-N, suggests that the vast majority of people make only a few phone calls with $n_{ST}, n_{TS} < 10$, and much fewer people make many phone calls, which points to skewness. In addition, concentration along the diagonal indicates that mutual people call each other mostly in a balanced fashion with $n_{ST} \approx n_{TS}$. Notice that similar arguments hold for CALL-D and SMS.

Even though heatmaps reveal similar patterns in all the three networks, mere visualization does

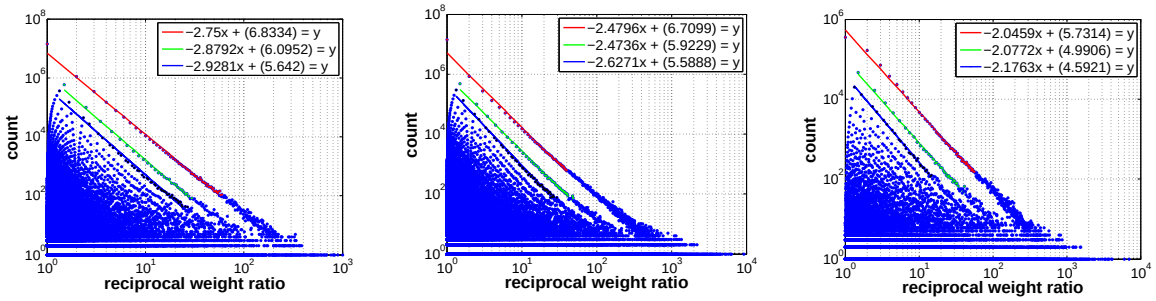


(a) CALL-N (b) CALL-D (c) SMS

Figure 4.7: (top-row) Scatter plot heatmaps: total weight n_{ST} (Silent to Talkative) vs the reverse, n_{TS}, in log scales. Visualization by scatter plots suffers from over-plotting. Heatmaps color-code dense regions but do not have compact representations or formulas. Figures are best viewed in color; red points represent denser regions. The counts are in $\log_2$ scale. (bottom-row) Aggregation by average: summarization and data aggregation, e.g. averaging, loses a lot of information.

not provide compact representations for our data. One way to go around this issue is to do data summarization. For example, Figure 4.7(bottom-row) shows how n_{TS} changes with n_{ST} *on average*. The least square fit of the data points in log-log scales then provides a mathematical representation of the data. Data summarization by means of an aggregate function such as the average, however, loses a lot of information about the actual distribution. For instance in our example, the slope of the least square fit in CALL-N is close to 1, which suggests that n_{TS} is equal to n_{ST} on average, and does not provide any information for the deviations. This issue arises mostly because aggregation by the average is not a good representative, especially for skewed distributions.

To further inspect the correlation between the mutual edge weights, we show the distribution of weight ratios $\frac{n_{TS}}{n_{ST}}$ in Figure 4.8 for (a) CALL-N, (b) CALL-D, and (c) SMS. We observe that the distributions follow "layers" of power-laws in all three plots. By fitting least square lines to the top three so-called layers of points in log-log scales, we notice that the power-law fits have similar exponents with shifted intercepts –many $1, 2, 3, \ldots$ ratios; fewer $1.5, 2.5, 3.5, \ldots$; even fewer $1.33, 1.66, 2.33, \ldots$; and so on. This confirms our visual observation in Figure 4.7 that a similar pattern in the distribution of mutual edge weights holds for all three human communication networks we study.



(a) CALL-N (b) CALL-D (c) SMS

Figure 4.8: Distribution of the ratio of weights on reciprocal edges $\frac{n_{TS}}{n_{ST}}$ follows "layers" of power-laws with similar exponents for all three types of weights (a) number of phone-calls W_N, (b) duration of phone-calls W_D, and (c) number of SMSs W_{SMS}.

Given our observation that the distribution of reciprocal edge weights (n_{ST}, n_{TS}) follows a similar pattern across all three networks, how can we model the observed distributions? Since neither visualization nor aggregation qualify for compact data representation, we propose to formulate the distributions with the following bivariate functional form $\Pr(n_{ST}, n_{TS})$, which we call the Triple Power-Law (3PL) function.

Proposed Model 1 (Triple Power-Law (3PL)) *In human communication networks, the distribution* $\Pr(n_{ST}, n_{TS})$ *of mutual edge weights* n_{ST} *and* n_{TS} *(*n_{ST} *being the smaller) follows a Triple Power-Law in the following form*

$$\Pr(n_{ST}, n_{TS}; \alpha, \beta, \gamma) \propto \boxed{\frac{n_{ST}^{-\alpha} n_{TS}^{-\beta} (n_{TS} - n_{ST} + 1)^{-\gamma}}{Z(\alpha, \beta, \gamma)}}, \alpha > 0, \beta > 0, \gamma > 0, \text{ and}$$

$$n_{TS} \geq n_{ST} > 0, \quad Z(\alpha, \beta, \gamma) = \sum_{n_{ST}=1}^{M} \sum_{n_{TS}=n_{ST}}^{M} n_{ST}^{-\alpha} n_{TS}^{-\beta} (n_{TS} - n_{ST} + 1)^{-\gamma}.$$

where Z is the normalization constant and M is a very large integer.

Next we elaborate on the intuition behind the exponents α, β and γ.

Intuition behind the β exponent : 3PL is the 2D extension of the "rich-get-richer" phenomenon; people who make many phone calls will continue making even more, and even longer ones, leading to skewed, power-law-like distributions. The β exponent is the skewness of the main component, the number n_{TS} of phone-calls from 'talkative' to 'silent'. High β means more skewed distribution; β=0 is roughly uniform distribution. As we show in Figure 4.7, there are many people who make only a few (and short) phone calls and only a few people who make many (and long) phone calls. Visually, the vast majority of people who make only a few phone calls are represented with the high density (dark red) regions around the origin in all three networks.

Intuition behind the α exponent : Similarly, this indicates the skewness for n_{ST}, the number of silent-to-talkative phone-calls. High value of α means high skewness, while α close to zero means uniformity. Notice that $\alpha \approx 0$ for our real phone-call datasets (see Figure 4.9).

Intuition behind the γ exponent : It captures the skewness in asymmetry. High γ means that large asymmetries are improbable. This is the case in all our real datasets. For example, in addition to the origin in Figure 4.7(a), the regions along the diagonal also have high densities. These regions correspond to mutual pairs with about equal interaction in both directions. This suggests that humans tend to reciprocate their communications. 3PL also captures this observation; notice that the probability is higher for n_{TS} close to n_{ST} and drops for larger inequality $(n_{TS} - n_{ST})$ as a power-law with exponent γ.

4.2.4 Comparison of 3PL to Competing Models

In order to strengthen our case for the 3PL, we would like to rule out several plausible competing distributions if possible. We cannot, however, compare the 3PL fit of our data with fits to every competing distribution, of which there are an infinite number. The reason is, it will always be possible to find a distribution that fits the data better than the 3PL if we define a family of functions with a sufficiently large number of parameters.

In this section, we compare our model with two well-known parametric distributions for skewed bivariate data from statistics, the Bivariate Pareto [Kotz et al., 2000] and the Bivariate Yule [Xekalaki, 1986]. Their functional forms are given as two alternative competitor models as follows.

Competitor Model 1 (Bivariate Pareto)

$$f_{X_1,X_2}(x_1, x_2) = k(k+1)(ab)^{k+1}(ax_1 + bx_2 + ab)^{-k-2}, x_1, x_2, a, b, k > 0.$$

Competitor Model 2 (Bivariate Yule)

$$f_{X_1,X_2}(x_1, x_2) = \frac{\rho_{(2)}(x_1 + x_2)!}{(\rho+1)_{(x_1+x_2+2)}}, x_1, x_2, \rho > 0; \alpha_{(\beta)} = \Gamma(\alpha+\beta)/\Gamma(\alpha), \alpha > 0, \beta \in R.$$

We use maximum likelihood estimation to fit the parameters of each model for each of our three networks. In Figure 4.9, we report the best-fit parameters as well as the corresponding data log likelihood scores (the higher, the better). Notice that for CALL-N and CALL-D the 3PL achieves higher data likelihood than both Bivariate Pareto and Bivariate Yule. On the other hand, for SMS, the data likelihood scores of all three models are about the same; with Bivariate Pareto giving a slightly higher score.

The simple sign of the difference between the log likelihoods (log likelihood ratio $\mathcal{R}$), however, does not on its own show conclusively that one distribution is better than the other as it is subject to statistical fluctuation. If its true value over many independent data sets drawn from the same distribution is close to zero, then the fluctuations can easily change its sign and thus the results

of the comparison cannot be trusted. In order to make a firm judgement in favor of 3PL, we need to show that the difference between the log likelihoods is sufficiently large and that it could not be the result of a chance fluctuation. To do so, we need to know the standard deviation σ on $\mathcal{R}$, which we estimate from our data using the method proposed in [Vuong, 1989].

In Figure 4.9, we report the normalized log likelihood ratio denoted by $z = \mathcal{R}/\sqrt{2n}\sigma$, where n is the total number of data points (number of mutual edge pairs in our case). A positive z value indicates that the 3PL model is truly favored over the alternative. We also show the corresponding p-value, $p = erfc(z)$, where *erfc* is the complementary Gaussian error function. It gives an estimate of the probability that we measured a given value of $\mathcal{R}$ when the true value of $\mathcal{R}$ is close to zero (and thus cannot be trusted). Therefore, a small p value shows that the value of $\mathcal{R}$ is unlikely to be a chance result and its sign can be trusted.

	CALL-N	CALL-D	SMS
Triple Power Law (3PL)			
α	1e-06	1e-06	0.8120
β	2.0703	1.8670	1.5896
γ	0.8204	0.9650	0.3005
Loglikelihood	-7.55e+07	-8.88e+07	-5.41e+06
Bivariate Pareto			
k	0.7407	0.7657	0.7862
a	0.2119	0.5723	0.7097
b	10e+05	1.25e+04	0.7553
Loglikelihood	-7.77e+07	-9.26e+07	-5.39e+06
z	803.73	975.75	-41.06
p	0	0	0
Bivariate Yule			
ρ	1.11e-16	5.55e-17	1e-06
Loglikelihood	-8.59e+07	-10.00e+07	-5.41e+06
z	2.14e+03	1.93e+03	1.49
p	0	0	0.03

Figure 4.9: Maximum likelihood parameters estimated for 3PL, Bivariate Pareto and the Bivariate Yule and data log-likelihoods obtained with the best-fit parameters. We also give the normalized log likelihood ratios z and the corresponding p-values. A positive (and large) z value indicates that 3PL is favored over the alternative. A small p-value confirms the significance of the result. Notice that 3PL provides significantly better fits to CALL and is as good as its competitors for SMS.

Notice that the magnitude of z for CALL-N and CALL-D is quite large, which makes the p-value zero and shows that 3PL is a significantly better fit for those data sets. On the other hand, z is relatively much smaller for SMS, therefore we conclude that 3PL provides as good of a fit as its competitors for this data set. Note that the number of mutual edge pairs n in SMS ($\approx$1 million) is much smaller compared to that of the call networks ($\approx$21 million) (Table 4.2). It is worth emphasizing that difference, because the bivariate pattern of reciprocity might reveal itself better in larger data sets, and it would be interesting to see whether 3PL provides a better fit for SMS when more data samples become available.

To this end, we also computed the Kolmogorov-Smirnov (KS) statistic, which is given as

$$max_{x_1,x_2}|F_n(x_1,x_2) - \hat{F}(x_1,x_2)|$$

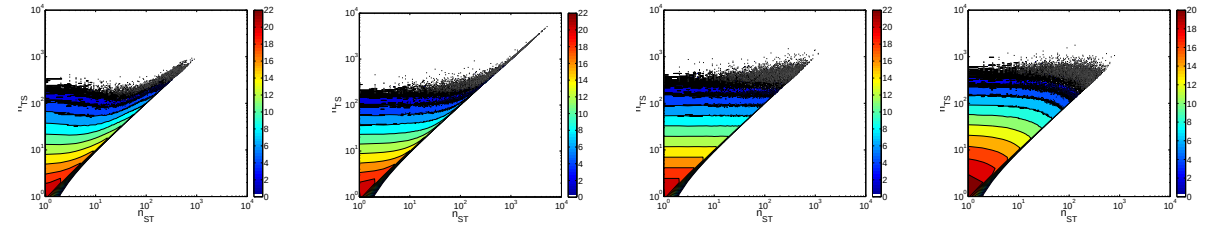
in which F_n is the empirical joint cumulative distribution function (CDF) of our data, and $\hat{F}$ is that from an estimated model. Unlike in one dimension, there is not a closed form

for its distribution under the null, but we can find it by simulation. (note that with bivariate data, the joint cumulative distribution function is $F(x_1, x_2) = Prob(X_1 \leq x_1, X_2 \leq x_2)$). In Table 4.3 we report the KS statistic of all three models for all of our three data sets. Notice that 3PL provides the smallest distance to the empirical distribution for all our call data sets under the KS test.

	CALL-N	CALL-D	SMS
Triple Power Law	**0.0189**	**0.0810**	0.0716
Bivariate Pareto	0.720	0.1147	**0.0113**
Bivariate Yule	0.2605	0.1661	0.0818

Table 4.3: Kolmogorov-Smirnov statistic; the maximum absolute difference between the joint empirical CDF computed from our data and the joint CDF estimated from three different models. The smaller values indicate a better fit to real data.

Next, we would like to demonstrate also visually that 3PL provides a better fit to the real data than its competitors. To this end, having estimated the model parameters for all three models, we generated synthetic data sets with the same number of samples as in each of our networks. We show the corresponding plots for CALL-N in Figure 4.10 (a) for real data, and synthetic data generated by (b) 3PL, (c) Bivariate Pareto, and (d) Bivariate Yule. We notice that the simulated data distribution from 3PL looks more realistic than its two competitors. Similar results for CALL-D and SMS are omitted for brevity.

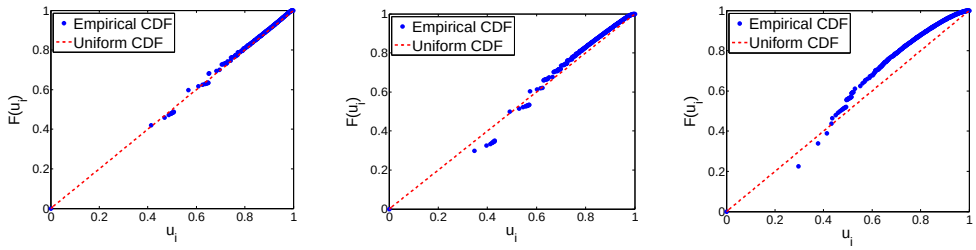


(a) CALL-N (real data) (b) 3PL (synt.) (c) Biv. Pareto (synt.) (d) Biv. Yule (synt.)

Figure 4.10: Contour-maps for the scatter plot n_{TS} versus n_{ST} in CALL-N (a) for real data, and synthetic data simulated from (b) 3PL, (c) Bivariate Pareto and (d) Bivariate Yule functions using the best-fit parameters. Notice that synthetic data generated by 3PL looks more similar to the real data than its competitors also visually. Counts are in $\log_2$ scale. Figures are best viewed in color.

Goodness of Fit

The likelihood ratio test is used to compare two models to determine which one provides a *better* fit to a given data. However, as we mentioned in the previous section, it cannot directly show when both competing models are poor fits to the data; it can only tell which is the least



(a) CALL-N (b) CALL-D (c) SMS

Figure 4.11: Distribution of $\hat{u}_i = \hat{F}(x_{1,i}, x_{2,i})$ for all data points i according to cumulative distribution function (CDF) $\hat{F}$ estimated from our 3PL model. An approximately uniform distribution of $\hat{u}_i$ shows that 3PL provides a good fit to real data.

bad. Therefore, in addition to showing that 3PL provides a better (or as good) fit than its two competitors, we also need to demonstrate that it indeed provides a good fit itself.

A general class of tests for goodness of fit work by transforming the data points $(x_{1,i}, x_{2,i})$ according to a cumulative distribution function (CDF) F as $u_i = F(x_{1,i}, x_{2,i})$ for $\forall i, 1 \leq i \leq n$. One can show that if F is the correct CDF for the data, u_i should be *uniformly* distributed (derivation follows from basic probability theory). That is, if the CDF $\hat{F}$ estimated from our model is approximately correct, the empirical CDF of the $\hat{u}_i = \hat{F}(x_{1,i}, x_{2,i})$ should be approximately a straight line from $(0, 0)$ to $(1, 1)$.

For each of our three data sets, we generate synthetic data drawn from our 3PL function with the corresponding estimated best-fit parameters. Then, we compute $\hat{u}_i = \hat{F}(x_{1,i}, x_{2,i})$ for all the data points in each of the data sets, where $\hat{F}$ is the estimated CDF from each synthetic data. In Figure 4.11, we show the CDF of $\hat{u}_i$ as well as the CDF for a perfect uniform distribution. Notice that the distribution of $\hat{u}_i$ is almost uniform for CALL-N and CALL-D, and quite close to the uniform for SMS. This corroborates our case that our model provides a good approximate to the correct CDF of our data sets, and thus indeed provides a *good* fit.

4.2.5 3PL at Work

There exist at least three levels at which we can make use of parametric statistical models for real data: (1) *as data summary*: compact mathematical representation, data reduction; (2) *as simulators*: generative tools for synthetic data; (3) *in anomaly detection*: probability density estimation.

In Figure 4.12 (a), we show top 100 pairs in CALL-D with lowest 3PL likelihood (marked with triangles). Figure 4.12 (b) shows the local neighborhood of one of the pairs, say A and B (marked with circles in (a)). We notice low mutuality; A initiated 99% of the calls in return to less than 2 hours total duration of calls B made. Further inspection revealed constant daily activity by A, including weekends, with about 7 hours call duration per day on average, starting at around 9am in the morning until around 5-8pm in the evening. It is also surprising that all these calls are

addressed to the same contact, B. While for privacy reasons, we cannot fully tell the scenario behind this behavior, this proves to be an interesting case for the service operator to further look into. Other interesting anomalous observations are omitted for brevity.

In CALL-N, one of the outliers we found is a mutual pair of users, say A and B, where A initiated more than 2600 calls in reply to 45 calls only. As it looks suspicious, we examined the spread of these calls over the course of six months for a sanity check. To our surprise, we found out that for two consecutive months, A called B 40 to 60 times every single day, constantly. About 90% of these calls lasted less than 10 minutes, and more than 50% took less than 2 minutes. In addition, all the calls lasted strictly less than 1 hour (see Figure 4.13). This shows that a vast majority of the many calls between this pair were quite short.

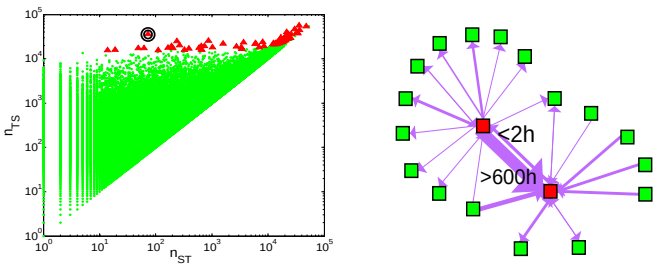


Figure 4.12: (a) Least likely 100 points by 3PL (shown with triangles). (b) Local neighborhood of one mutual pair detected as an outlier (marked with circles). Edge thickness is proportional to edge weight.

One scenario for this situation could be that there exists a specific package from the mobile service offering a constant rate for all the calls that takes 10 minutes or less. And thus for longer calls the parties had to hang up and initiated another call, which would result in excessive number of phone calls as observed. Another scenario could be the case of faulty equipment dropping calls quite frequently and thus making the parties redial. Anyhow, we believe that the high non-reciprocity as well as the large number of calls initiated presents an interesting case for the service operator to further look into.

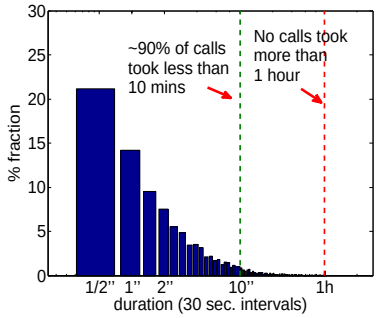


Figure 4.13: Distribution of the duration of over 2600 calls detected as an anomaly in CALL-N. Notice that a majority of the calls took less than 10 minutes.

4.2.6 Reciprocity and Local Network Topology

Given that person i calls person j w_{ij} times and person j calls person i w_{ji} times, what is the *degree* of reciprocity between them? In this section, we discuss several *weighted* metrics that quantify reciprocity between a given mutual pair. Later, we study the relationship between reciprocity among mutual pairs and their topological similarity.

date	time	duration	date	time	duration
01-DEC	9:0:54	1h 3′	03-DEC	9:1:24	1h 55′
01-DEC	10:4:57	1h 11′	03-DEC	11:36:55	1h 42′
01-DEC	11:44:44	1h 31′	03-DEC	13:19:39	2h 49′
01-DEC	13:31:20	20′	03-DEC	16:9:6	44′
01-DEC	15:7:49	1h 20′			
01-DEC	18:10:6	35′	04-DEC	9:7:57	8h 12′
01-DEC	19:3:40	10′			
01-DEC	19:18:45	8′	10-DEC	9:38:58	4h 3′
			10-DEC	13:42:55	50′
02-DEC	9:9:1	2h 8′	10-DEC	14:36:1	1h 56′
02-DEC	11:41:8	3h 16′	10-DEC	18:44:41	2h 5′
02-DEC	14:59:44	2h 31′			
02-DEC	17:47:25	1h 11′	22-DEC	9:20:25	2h 40′
02-DEC	20:24:15	9′	22-DEC	12:36:15	2h 9′
			22-DEC	14:45:44	1h 7′

Figure 4.14: Spread of calls by days for the anomaly detected in CALL-D. Notice that the calls are made during the entire duration of working hours.

Weighted reciprocity metrics

Three metrics we considered in this work to quantify the "similarity" or "balance" of weights w_{ij} and w_{ji} are (1) *Ratio* $r = \frac{\min(w_{ij},w_{ji})}{\max(w_{ij},w_{ji})} \in [0,1]$, (2) *Coherence* $c = \frac{2\sqrt{w_{ij}w_{ji}}}{(w_{ij}+w_{ji})} \in [0,1]$ (geometric mean divided by the arithmetic mean of the edge weights), and (3)*Entropy* $e = -p_{ij}\log_2(p_{ij}) - p_{ji}\log_2(p_{ji}) \in [0,1]$, where $p_{ij} = \frac{w_{ij}}{(w_{ij}+w_{ji})}$ and $p_{ji} = 1 - p_{ij}$. All these metrics are equal to 0 for the (non-mutual) pairs where one of the edge weights is 0, and equal to 1 when the edge weights are equal. Although these metrics are good at capturing the *balance* of the edge weights, they fail to capture the *volume* of the weights. For example, human would score $(w_{ji}$=100, w_{ij}=100) higher than $(w_{ji}$=1, w_{ij}=1), whereas all the metrics above would treat them as equal.

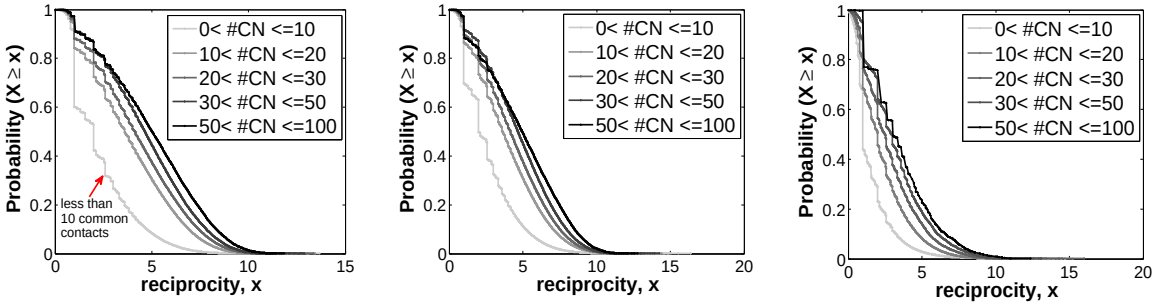
Therefore, we propose to multiply these metrics by the logarithm of the total weight, such that the reciprocity score consists of both a "balance" as well as a "volume" term. In the rest of this section, we use the *weighted* ratio $r_w = \frac{\min(w_{ij},w_{ji})}{\max(w_{ij},w_{ji})}\log(w_{ij}+w_{ji})$ as the reciprocity measure in our experiments. The results are similar for the other weighted metrics, c_w and e_w.

Reciprocity and Network Overlap

Here, we want to understand whether there is a relation between the local network overlap (local density) and reciprocity between mutual pairs. Local network overlap of two nodes is simply the number of common neighbors they have in the network.

In Figure 4.15, we show the cumulative distribution of reciprocity separately for different ranges of overlap. The figures suggest that people with more common contacts tend to exhibit higher

reciprocity, both in their SMS and phone call interactions.



(a) CALL-N (b) CALL-D (c) SMS

Figure 4.15: Complementary cumulative distribution of reciprocity for different ranges of local network overlap (number of Common Neighbors). Notice that the more the number of common contacts, the higher the reciprocity.

Reciprocity and Degree Similarity

Next, we investigate the relation between the degree similarity (degree assortativity) and reciprocity. In Figure 4.16, we show the heatmap for the average reciprocity among pairs with respective degrees d_i and d_j for CALL-N (similar figures for other networks are omitted for brevity). The heatmap plot suggests that two people with more similar number of contacts exhibit larger reciprocity; notice the increase in reciprocity with increasing d_j for fixed d_i (from bottom to diagonal, towards degree similarity) and then the drop from diagonal to the right, towards degree dissimilarity.

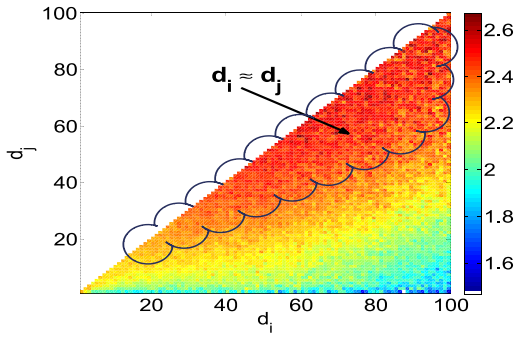


Figure 4.16: Average reciprocity among dyads with degrees (d_i, d_j) in CALL-N. Notice that reciprocity is higher among pairs with similar degrees (number of contacts). Red color represent higher average reciprocity. Figure is best viewed in color.

Discussion

Our observations suggest that people who have many common contacts as well as similar number of contacts exhibit higher reciprocity, i.e., they play equally active roles in maintaining their relationship. In this discussion, we put these findings into context and relate them to the network topology at large.

Assume that our networks consist of hubs and communities. Communities would represent tightly connected groups of people (e.g., social circle of friends), and hubs would represent the

"connector" nodes that are linked from several different communities (e.g. call centers, pizza shops, etc.). Intuitively, the reciprocity is higher among friends in the same social circle, that is within communities, and much lower between a hub node and its neighbors.

We believe that our results confirm this hubs and communities structure assumption. The reason is that, communities are clique-like structures where the nodes in the clique are about the same degree and share many common neighbors. In contrast, hubs exhibit star-like structures; they share much fewer common neighbors and have much lower degree similarity with their neighbors. Therefore, the existence of a hubs and communities structure in our networks is a sufficient condition that would give us our observed results.

We reached these conclusions by studying dyadic relations only. It would be interesting to conduct a similar study at the communities level and see how reciprocity changes between different communities. We conjecture that similar results would hold, due to the communities-within-communities structure also observed in real networks [Chakrabarti et al., 2004b; Girvan and Newman, 2002].

4.3 Patterns in call durations

4.3.1 Motivation

In the study of phone calls databases [Onnela et al., 2007a; Seshadri et al., 2008; Willkomm et al., 2008], a common technique to ease the analysis of the data is the summarization of the phone calls records into aggregated attributes [Hill et al., 2007], such as the aggregate calls duration or the total number of phone calls. By doing that, the size of the database can be reduced by orders of magnitudes, allowing the execution of most well known data mining algorithms in a feasible time. However, we believe that such representation veils relevant temporal information inherent in a user or in a relationship between two people. When all the information about the phone calls records of a user is aggregated into single summarized attributes, we do not know anymore how often this user calls or for how long he talks per phone call. One may suggest, for instance, to use descriptive statistics such as mean and variance to describe the duration of the user's phone calls, but it is well known that the distribution of these values is highly skewed [Willkomm et al., 2008], what invalidate the use of such statistics.

In this work, we tackle the following problem. Given a very large amount of phone records, what is the best way to summarize the calling behavior of a user? In order to answer this question, we examine phone call records obtained from the network of a large mobile operator of a large city. More specifically, we analyze the duration of hundreds of million calls and we propose the *Truncated Lazy Contractor* (TLAC) model to describe how long are the durations of the phone calls of a single user. Thus, the TLAC models the Calls Duration Distribution (CDD) of a user and is parsimonious, having only two parameters, the *efficiency* coefficient ρ and the *weakness* coefficient β. We show that the TLAC model was the best alternative to model the CDD of the

users of our dataset, mainly because it has a heavier tail and head than the log-normal distribution, that is the most commonly used distribution to model CDDs [Guo et al., 2007].

We also suggest the use of the TLAC parameters as a better way to summarize the calls duration behavior of a user. We propose the *MetaDist* to model the population of users that have a determined calls duration behavior. The *MetaDist* is the meta-distribution of the ρ_i and β_i parameters of each user i's CDD and, when its isocontours are visualized, its shape is surprisingly similar to a bivariate Gaussian distribution. This fascinating regularity, observed in a significantly noisy data, makes the *MetaDist* a potential distribution to be explored in the direction of better understanding the call behavior of mobile users.

In summary, the main contributions of this work are:

- The proposal of the TLAC model to represent the individual phone calls durations of mobile customers;
- The *MetaDist* to model the group call behavior of the mobile phone users;
- The use of the *MetaDist* and the *Focal Point* to describe the collective temporal evolution of large groups of customers;

As an additional contribution, we show the usefulness of the TLAC model. We show that it can spot anomalies and it can succinctly verify correlations (or lack thereof) between the TLAC parameters of the users and their total number of phone calls, aggregate duration and distinct patterns. We also emphasize that the TLAC model can be used to generate synthetic datasets and to significantly summarize a very large number of phone calls records.

4.3.2 Patterns and Proposed TLAC Model

In this work, we analyze mobile phone records of millions of mobile phone users in our CALL dataset (see Table 4.2), during four months. In this period, about half a billion phone calls were registered and, for each phone call, we have information about the duration of the phone call, the date and time it occurred and encrypted values that represent the source and the destination of the call, that may be mobile or not. When not stated otherwise, the results shown in this work refer to the phone call records of the first month of our dataset. The results for the other 3 months are explicitly mentioned in the next section.

Problem Definition

The Call Duration Distribution (CDD) is the distribution of the call duration per user in a period of time, that in our case, is one month. In the literature, there is no consensus about what well known distribution should be used to model the CDD. There are researchers that claim that the PDD should be modeled by a log-normal distribution [Guo et al., 2007] and others that it should be modeled by the exponential distribution [Tejinder S. Randhawa, 2003]. Thus, in this section, we tackle the following problem:

Problem 1 *CDD FITTING. Given $d_1, d_2, ..., d_{n_i}$ durations of n_i phone calls made by a user i in a month, find the most suitable distribution for them and report its parameters.*

As we mentioned before, there is no consensus about what well known distribution should be used to model the CDD, i.e., for some cases the log-normal fits well and for others, the exponential is the most appropriate distribution. Thus, finding another specific random distributions that could provide good fittings to a particular group of CDDs would just add another variable to Problem 1, without solving it. Therefore, we propose that the distribution that solves Problem 1 should necessarily obey to the following requirements:

- **R1**: Intuitively explain the intrinsic reasons behind the calls duration;
- **R2**: Provide good reliable fits for the great majority of the users.

In the following sections, we present a solution for Problem 1 and tackle Requirement $R1$ by presenting the TLAC model, which is an intuitive model to represent CDDs. Then, we tackle Requirement $R2$ by showing the goodness of fit of the TLAC model for our dataset.

TLAC Model

Given these constraints, we start solving Problem 1 by explaining the evolution of the calls duration by a survival analysis perspective. We consider that all the calls $c_1, c_2, ..., c_C$ made by a user in a month are individuals which are alive while they are active. When a phone call c_j starts, its initial lifetime $l_j = 1$ and, as time goes by, l_j progressively increments until the call is over. It is obvious that the final lifetime of every c_j would be its duration d_j.

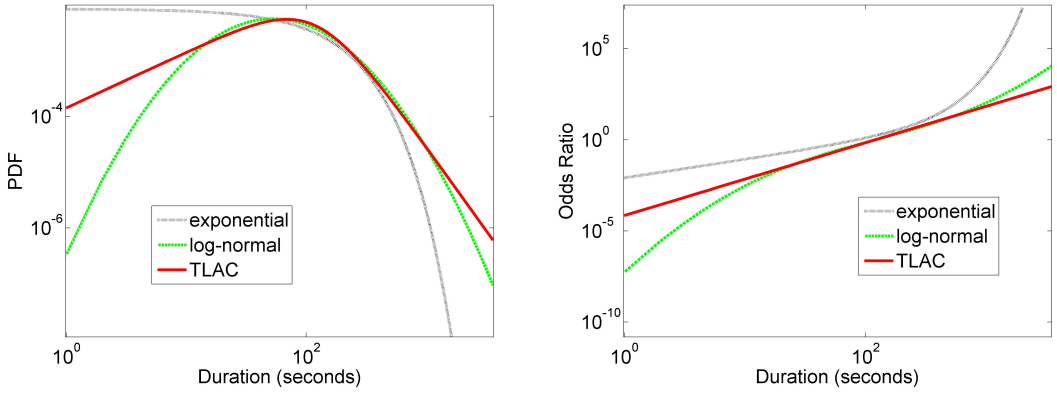
In the survival analysis literature, an interesting survival model that can intuitively explain the lifetime, i.e. duration, of the phone calls is the *log-logistic* distribution. And besides its use in survival analysis [Bennett, 1983; Lawless and Lawless, 1982; Mahmood, 2000], there are examples in the literature of the use of the log-logistic distribution to model the distribution of wealth [Fisk, 1961], flood frequency analysis[M.I. Ahmad and Werritty, 1988] and software reliability[Gokhale and Trivedi, 1998]. All of these examples present a modified version of the well known "rich gets richer" phenomenon. First, for a variable to be "rich", it has to face several risks of "dying" but, if it survives, it is more likely to get "richer" at every time. We propose that the same occurs for phone calls durations. After the initial risks of hanging up the call, e.g., wrong number calls, voice mail calls and short message calls such as "I am busy, talk to you later" or "I am here. Where are you?" type of calls, the call tends to get longer at every time. As an example, the lung cancer survival analysis case [Bennett, 1983] parallels our environment if we substitute endurance to disease with propensity to talk: a patient/customer that has stayed alive/talking so far, will remain such, for more time, i.e., the longer is the duration of the call so far, the more the parties are enjoying the conversation and the more the call will survive.

Thus, to solve Problem 1, we propose the **Truncated Lazy Contractor (TLAC)** model, that is a truncated version of the log-logistic distribution, since it not contains the interval $[0, 1)$. Firstly we show, in Figure 4.17-a, the Probability Density Function (PDF) of the TLAC , the log-normal and exponential distributions, in order to emphasize the main differences between

these models. The parameters were chosen accordingly to a median call duration of 2 minutes for all distributions. The TLAC and log-normal distributions are very similar, but the TLAC is less concentrated in the median than the log-normal, i.e., it has power law increase ratios in its head and in its tail. We believe that this is another indication that the TLAC is suitable to model the users' CDD, since as it was verified by [Willkomm et al., 2008], CDDs have semi-heavy" tails. The basic formulas for the log-logistic distribution and, consequently, for the TLAC , are [Lawless and Lawless, 1982]:

$$\begin{aligned} PDF_{TLAC}(x) &= \frac{\exp(z(1+\sigma)-\mu)}{(\sigma(1+e^z))^2} \\ CDF_{TLAC}(x) &= \frac{1}{1+\exp(-\frac{(\ln(x)-\mu)}{\sigma})} \\ z &= (\ln(x)-\mu)/\sigma \end{aligned}$$

where μ is the location parameter and σ the shape parameter.



(a) PDF (b) Odds Ratio

Figure 4.17: Comparison of shapes of log-normal, exponential and TLAC distributions.

Moreover, in finite sparse data that spans for several orders of magnitude, that is the case of CDDs when they are measured in seconds, it is very difficult to visualize the PDFs, since the distribution is considerably noisy at its tail. One option is to smooth the data by reducing its magnitude by aggregating data into buckets, with the cost of lost of information. Another option is to move away from the PDF and analyze the cumulative distributions, i.e., cumulative density function (CDF) and complementary cumulative density function (CCDF) [Clauset et al., 2009]. These distributions veil the sparsity of the data and also the possible irregularities that may occur for any particular reason. However, by using the CDF (CCDF) you end up losing the information in the tail (head) of the distribution. In order to escape from this drawbacks, we propose the use of the Odds Ratio (OR) function, that is a cumulative function where we can clearly see the distribution behavior either in the head and in the tail. This $OR(t)$ function is commonly used in the survival analysis and it measures the ratio between the number of individuals that have not

survived by time t and the ones that survived. Its formula is given by:

$$OR(t) = \frac{CDF_{TLAC}(t)}{1 - CDF_{TLAC}(t)} \tag{4.7}$$

Therefore, in Figure 4.17-b, we plot the OR function for the TLAC , the log-normal and exponential distributions. The OR function of the exponential distribution is a power law until t reaches the median, and then it grows exponentially. On the other hand, the OR function of the log-normal grows slowly in the head and then fast in the tail. Finally, the OR function for the TLAC is the most interesting one. When plotted in log-log scales, is a straight line, i.e., it is a power law. Thus, as shown in [Bennett, 1983], the $OR(t)$ function can be summarized by the following linear regression model:

$$\ln(OR(t)) = \rho \ln(t) + \beta \tag{4.8}$$

$$OR(t) = e^{\beta} t^{\rho} \tag{4.9}$$

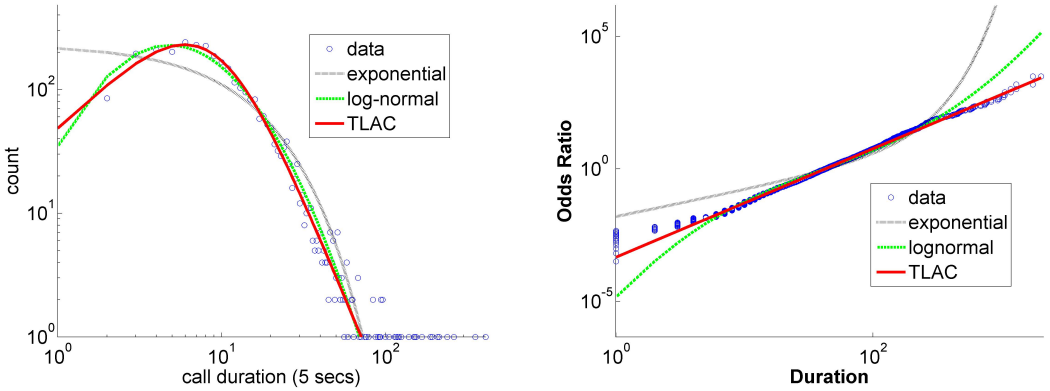
In our context, Equation 4.8 means that the ratio between the number of calls that will die by time t and the ones that will survive grows with a power of ρ. Moreover, given that the median $\hat{t}$ of the CDD is given when $OR(t) = 1$ and $OR(t) < 1$ when $t < \hat{t}$, the probability of a call to end grows with t when $t < \hat{t}$ and then decrease forever. We call this phenomenon the "lazy contractor" effect, which represents the time a lazy contractor takes to complete a job. If the job is easier and does require less effort than the ordinary regular job, he finishes it fast. However, for jobs that are harder and that demand more work than the ordinary regular job, the contractor also gets more lazier and takes even more time to complete it, i.e., the longer a job is taking to be completed, the longer it will take. The ρ and the β are the parameters of the TLAC model, with $\rho = 1/\sigma$.

We conclude this section and, therefore, the first part of the solution to Problem 1, by explaining the intuition behind the parameters of the TLAC model. The parameter ρ is the *efficiency* coefficient, which measures how efficient is the contractor. The higher the ρ, the more efficient is the contractor and the faster he will complete the job. On the other hand, the location parameter β is the *weakness* coefficient, which gives the duration $\hat{t}$ of the typical regular job a contractor with a determined efficiency coefficient ρ can take without being lazy, where $\hat{t} = exp(-\beta/\rho)$. This means that the lower the β, the harder are the jobs that the contractor is used to handle.

Goodness of Fit

In this section, we tackle the second requirement of Problem 1 by showing the goodness of fit of our TLAC model. First, we show in Figure 4.18-a, the PDF of the CDD for a high talkative user, with 3091 calls, and with the values put in buckets of 5 seconds to ease the visualization. We also show the best fittings using Maximum Likelihood Estimation (MLE) for the exponential and the log-normal distributions and also for our proposed TLAC model. Visually, it is clear that the best fittings are the ones from the log-normal distribution and the TLAC distribution, with the exponential distribution not being able to explain either the head and the tail of the CDD.

However, by examining the OR plot in Figure 4.18-b, we clearly see the the TLAC model provide the best fitting for the real data. As verified for the exponential distribution in the PDF, in the OR case, the log-normal also could not explain either the head and the tail of the CDD. We also point out that we can see relevant differences between the TLAC model and the real data only for the first call durations, that happen because these regions represent only a very small fraction of the data. The results showed in Figure 4.18 once more validate our proposal that the TLAC is a good model for CDDs.



(a) PDF (b) Odds Ratio

Figure 4.18: Comparison of models for the distribution of the phone calls duration of a high talkative user, with 3091 calls. TLAC in red, log-normal in green and exponential in black. Visually, for the PDF both the TLAC and the log-normal distribution provide good fits to the CDD but, for the OR, the TLAC clearly provide the best fit.

Given our initial analysis, we may state that the TLAC seems to be a good fit for the CDDs and also serve as an intuitive explanation for how the durations of the calls are generated. However, in order to conclude our answer for Problem 1, we must verify its generality power and also compare it to the log-normal and exponential generality power as well. Thus, we verify which one of the distributions can better fit the CDD of all the users of our dataset that have $n > 30$ phone calls. We calculated, for every user, the best fit according to the MLE for the TLAC , the log-normal and exponential distributions and we performed a Kolmogorov-Smirnov goodness of fit test [Massey, 1951], with 5% of significance level, to verify if the user's CDD is either one of these distributions. From now on, every time we mention that a distribution was correctly fitted, we are implying that we succesfully performed a Kolmogorov-Smirnov goodness of fit test.

In Figure 4.19, we show the percentage of CDDs that could be fitted by a log-normal, a TLAC and an exponential distribution. As we can see, the TLAC distribution can explain the highest fraction of the CDDs and the exponential distribution, the lowest. We observe that the TLAC distribution correctly fit almost 100% of the CDDs for users with $n < 1000$. From this point, the quality of the fittings starts to decay, but significantly later than the log-normal distribution. We emphasize that the great majority of users have $n < 1000$, what indicates that some of these talkative users'

CDD are probably driven by non natural activities, such as spams, telemarketing or other strong commercial-driven intents. This result, allied to the fact that the TLAC distribution could model more than 96% of the users, make it reasonable to answer Problem 1 claiming that the TLAC distribution is the standard model for CDDs in our dataset.

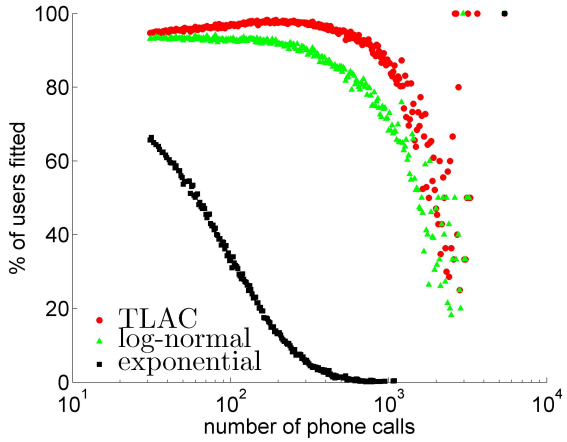
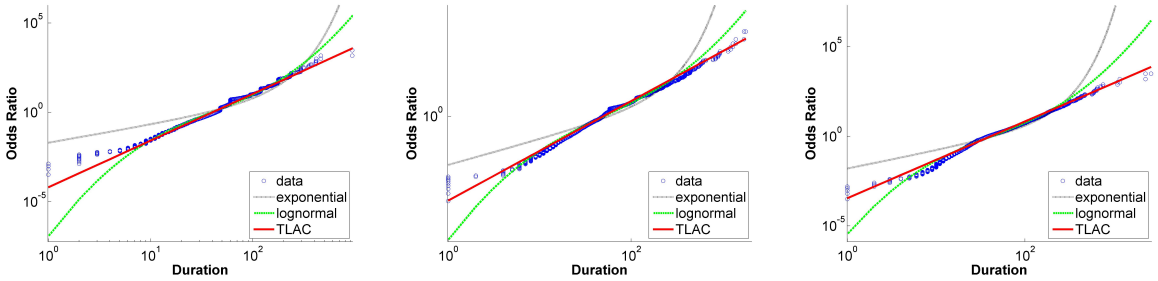


Figure 4.19: Percentage of users' CDDs that were correctly fitted vs. the user's number of calls c. The TLAC distribution is the one that provided better fittings for the whole population of customers with $c > 30$. It correctly fitted more than 96% of the users, only significantly failing to fit users with $c > 10^3$, probably spammers, telemarketers or other non-normal behavior user.

Finally, we further explore Problem 1 by looking at the OR of the talkative users that were not correctly fitted by the TLAC model. In Figure 4.20, we show the OR for three of these users and, as we observe, even these customers have a visually good fitting to the TLAC model. These results corroborate even more with the generality power of TLAC . Despite the fact that the irregularities of these customers' CDDs unable them to be correctly fitted by the TLAC model, it is clear that the TLAC can represent their CDDs significantly well.



(a) (b) (c)

Figure 4.20: Odds ratio of 3 talkative customers that were not correctly fitted by TLAC .

4.3.3 TLAC Over Time

We know it is trivial to visualize the distribution of users with a determined summarized attribute, such as number of phone calls per month or aggregate calls duration. However, if we want to visualize the distribution and evolution of a temporal feature of the user such as his CDD, things start to get more complicated. Thus, in this section, we tackle the following problem:

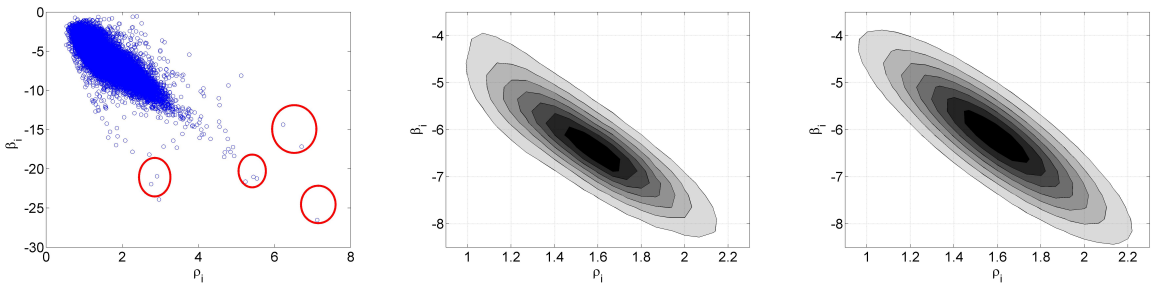
Problem 2 *EVOLUTION. Given the ρ_i and β_i parameters of N customers ($i = 1, 2..., N$), describe how they collectively evolve over time.*

We propose two approaches to solve Problem 2. In the next two sections, we describe the *MetaDist* solution and the *Focal Point* approach, respectively.

Group Behavior and Meta-Fitting

Since we know that the great majority of users' CDD can be modeled by the TLAC model, in order to solve Problem 2, we need to figure out how each user i is distributed according to their parameters ρ_i and β_i of the TLAC model. If the meta-distribution of the parameters ρ_i and β_i is well defined, then we can model the collective call behavior of the users and see its evolution over time. From now on, we will call the meta-distribution of the parameters ρ_i and β_i the *MetaDist* distribution.

In Figure 4.21-a, we show the scatter plot of the parameters ρ_i and β_i of the CDD of each user i for the first month of our dataset. We can not observe any latent pattern due to the overplotting but, however, we can spot outliers. Moreover, by plotting the ρ_i and β_i parameters using isocontours, as shown in Figure 4.21-b, we automatically smooth the visualization by disconsidering low populated regions. While darker colors mean a higher concentration of pairs ρ_i and β_i, white color mean that there are no users with CDDs with these values of ρ_i and β_i.



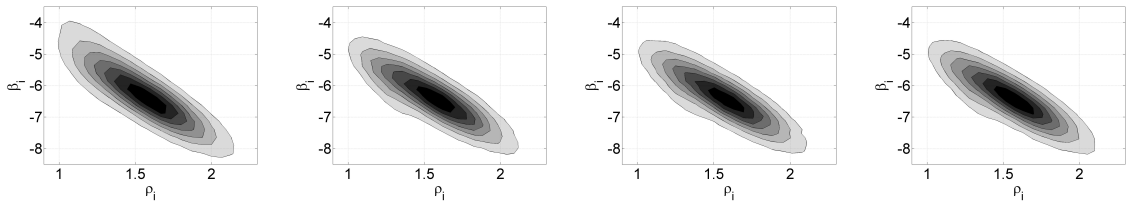
(a) Rough scatter plot (b) Isocontours of the real data (c) Bivariate Gaussian fitting.

Figure 4.21: Scatter plot of the parameters ρ_i and β_i of the CDD of each user i for the first month of our dataset. In (a) we can not see any particular pattern, but we can spot outliers. By plotting the isocontours (b), we can observe how well a bivariate Gaussian (c) fits the real distribution of the ρ_i and β_i of the CDDs ('meta-fitting')

Surprisingly, we observe that the isocontours of Figure 4.21-b are very similar to the ones of a bivariate Gaussian. In order to verify this, we extracted from the *MetaDist* distribution the means P and B of the parameters ρ_i and β_i, respectively, and also the covariance matrix Σ. We use these values to generate the isocontours of a bivariate Gaussian distribution and we plotted it in Figure 4.21-c. We observe that the isocontours of the generated bivariate Gaussian distribution are similar to the ones from the *MetaDist* distribution, which indicates that both distributions are also similar. Thus, we conjecture that a bivariate Gaussian distribution fits the real distribution of ρ and βs, making the *MetaDist* a good model to represent the population of users with a determined calls duration behavior.

Given that the *MetaDist* is a good model for the group behavior of the customers in our dataset, we can now visualize and measure how they evolve over time. In Figure 4.22 we show the evolution of the *MetaDist* over the four months of our dataset. The first observation we can make is that the bivariate Gaussian shape stands well during the whole analyzed period, what validates the robustness of the *MetaDist* . Moreover, a primary view indicates that the meta-parameters also have not changed significantly over the months. This can be confirmed by the first 5 rows of Table 4.4, which describes the value of the meta-parameters P, B and $\Sigma(\sigma^2_{\rho_i}, \sigma^2_{\beta_i}, cov(\rho_i, \beta_i))$ for the four analyzed months. This indicates that the phone company already reached a stable state before its customers concerning its prices, plans and services. In fact, the only noticeable difference occurs between the first month and the others. We observe that the meta-parameters of the first month have a slightly higher variance than the others, what indicates that this is probably an atypical month for the residents of the country in which our phone records were collected. But in spite of that, in general, the meta-parameters do not change through time. Then, we can state the following observation:

Observation 4.3.1 *TYPICAL BEHAVIOR. The typical human behavior is to have a efficiency coefficient* $\rho \approx 1.59$ *and a weakness coefficient* $\beta \approx -6.25$*. Thus, the median duration for a typical mobile phone user is* 51 *seconds and the mode is* 20 *seconds.*



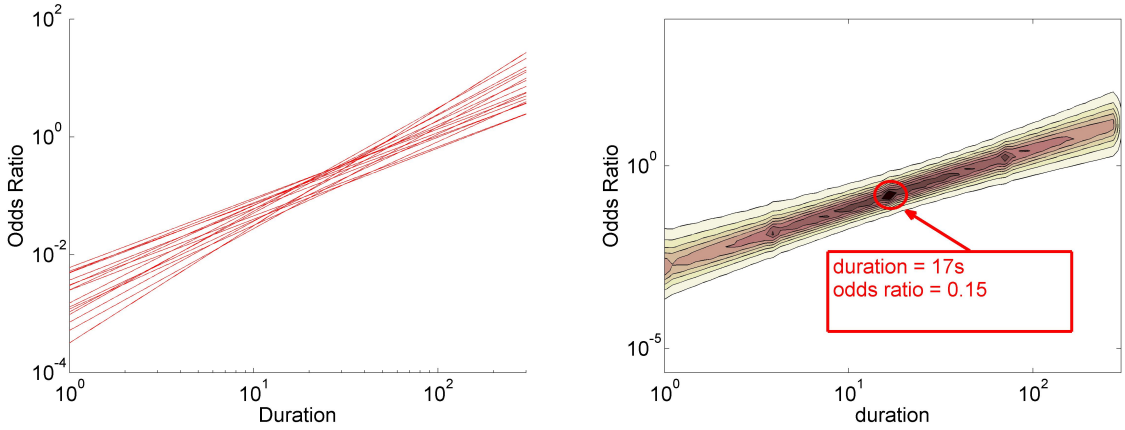
(a) Month 1 (b) Month 2. (c) Month 3 (d) Month 4.

Figure 4.22: Evolution of the *MetaDist* over the four months of our dataset. Note that the collective behavior of the customers is practically stable over time.

Focal Point

An interesting observation we can derive from the *MetaDist* showed in Figure 4.21 is that there exists a significant negative correlation between the parameters ρ_i and β_i. This negative correla-

tion, more precisely of -0.86, lead us to the fact that the OR lines, i.e., the TLAC odds ratio plots of the customers of our dataset, when plotted together, should cross over a determined region. In order to verify this, we plotted in Figure 4.23-a the OR lines for some customers of our dataset. As we can observe, it appears that these lines are all crossing in the same region, when the duration is approximately 20 seconds and the odds ratio approximately 0.1. Then, in Figure 4.23-b, we plotted together the OR lines of $20,000$ randomly picked customers and derived from them the isocontours to show the most populated areas. As we can observe, there is a highly populated point when the duration is 17 seconds and the OR is 0.15. By analyzing the whole month dataset, we verified that more than 50% of the users have OR lines that cross this point. From now on, we call this point the *Focal Point* .



(a) Direct plot. (b) Isocontours of the plot.

Figure 4.23: The TLAC lines of several customers plotted together. We can observe that, given the negative correlation of the parameters ρ_i and β_i, that the lines tend to cross in one point (a). We plot the isocontours of the lines together and approximately 50% of the customers have TLAC lines that pass on the high density point (duration=$17s$, OR=0.15) (b).

Formally, the *Focal Point* is a point on the OR plot with two coordinates: a coordinate $FP_{duration}$ in the duration axis and a coordinate FP_{OR} in the OR axis. When a set of customers have their OR plots crossing at a *Focal Point* with coordinates $(FP_{duration}, FP_{OR})$, it means that for all these customers the $\frac{FP_{OR}}{1+FP_{OR}}$th percentile of their CDD is on $FP_{duration}$ seconds. Thus, in the 2 bottom lines of Table 4.4, we describe the *Focal Point* coordinates for the four months of our analysis and, surprisingly, the *Focal Point* is stationary. Thus, we can make the following observation:

Observation 4.3.2 *UNIVERSAL PERCENTILE. The vast majority of mobile phone users has the same* 10*th percentile, that is on* 17 *seconds.*

Observation 4.3.2 suggests that one of the risks for a call to end acts in the same way for everyone. We conjecture that, given the 17 seconds durations, this is the risk of a call to reach the voice mail of the destination's mobile phone, i.e., the callee could not answer the call. The duration of

this call involves listening to the voice mail record and leaving a message, what is coherent with the 17 seconds mark. It would be interesting to empirically verify the percentage of phone calls that reaches the voice mail and compare with the *Focal Point* result.

-	1st month	2nd month	3rd month	4th month
P	1.59	1.58	1.59	1.59
B	-6.16	-6.28	-6.32	-6.30
$\sigma^2_{\rho_i}$	0.095	0.086	0.084	0.083
$\sigma^2_{\beta_i}$	1.24	0.98	0.95	0.94
$cov(\rho_i, \beta_i)$	-0.30	-0.24	-0.24	-0.23
$FP_{duration}(s)$	17	17	17	17
FP_{OR}	0.15	0.12	0.11	0.11

Table 4.4: Evolution of the meta-parameters (rows 1-5) and the *Focal Point* (rows 6-7) during the four months of our dataset.

4.3.4 TLAC at Work

In the previous section, we showed the collective behavior of millions of mobile phone users is stationary over time. We described two approaches to do that, one based on the *MetaDist* and the other based on the *Focal Point* . The initial conclusions of both approaches are same. First, the collective behavior of our dataset is stable, i.e., it does not change significantly over time. Second, we could see a slight difference between the first month and the others, indicating that this month is an atypical month in the year. We believe that these two approaches can succinctly and accurately aid the mobile phone companies to monitor the collective behavior of their customers over time.

Moreover, since we could successfully model more than 96% of the CDDs as a TLAC , a natural application of our models would be for anomaly detection and user classification. A mobile phone user that does not have a CDD that can be explained by the TLAC distribution is a potential user to be observed, since he has a distinct call behavior from the majority of the other users. To illustrate this, we show in Figure 4.24 a talkative node with a CDD that can not be modeled by a TLAC distribution. We observe that this node, indeed, has an atypical behavior, with his CDD having a noisy behavior from 10 to 100 seconds and also an impressive number of phone calls with duration around 1 hour (or 5×700 seconds). Moreover, another way to spot outliers is to check which users have a significant distance from the main cluster of the *MetaDist* . As we showed in Figure 4.21-a, this can be easily done even visually.

Another application that emerges naturally for our models is the summarization of data. By modeling the users' CDD into TLAC distributions, we are able to summarize, for each user i, hundreds or thousands of phone calls into just two values, the parameters ρ_i and β_i of the TLAC model. In our specific case, we could summarize over $0.1TB$ of phone calls data into less than $80MB$ of data. In this way, it is completely feasible to analyze several months, or even years

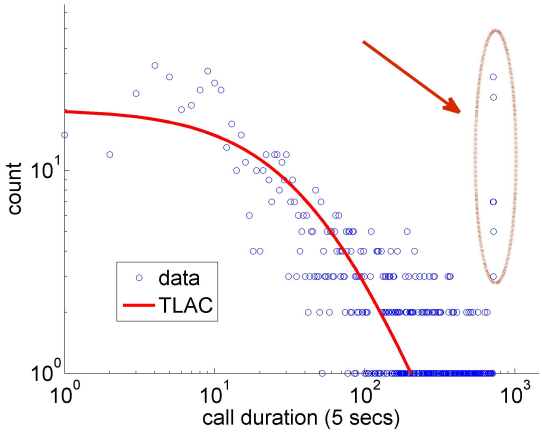


Figure 4.24: Outlier whose CDD can not be modeled by the TLAC distribution.

of temporal phone calls data and verify how the behavior of the users is evolving through time. Also, all the proposed models in this work can be directly applied on the design of generators that produce synthetic data, allowing researchers that do not have access to real data to generate their own.

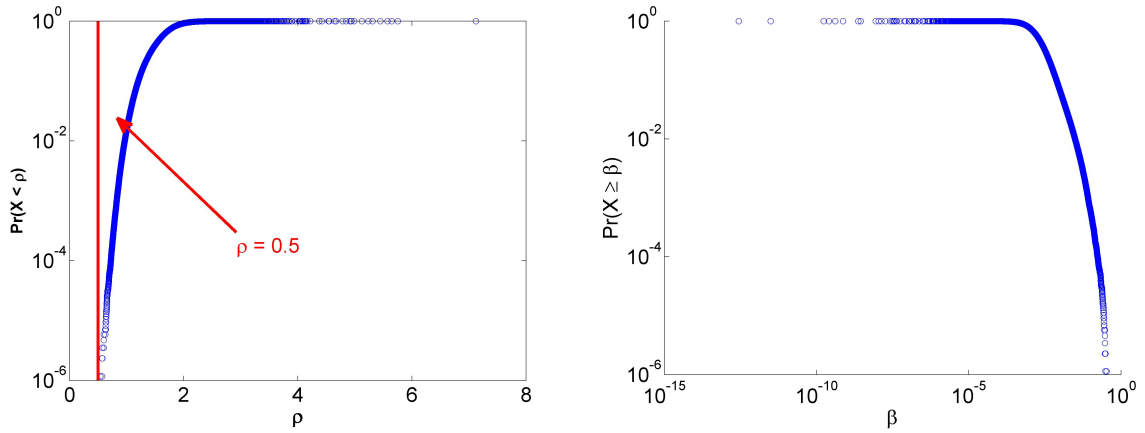
4.3.5 Discussion

Generality of TLAC

As we mentioned earlier, one of the major strengths of the TLAC model is its generalization power. We showed that even for distributions that oscillate between log-normal and log-logistic, or that have irregular spikes that unable them to be correctly fitted by TLAC , TLAC can represent them significantly well. Besides this, the simplicity of the TLAC model allow us to directly understand its form when its parameters are changed and verify its boundaries. For instance, in the case of the CDD, e^{β} gives the odds ratio when duration is 1 second. Thus, when $e^{\beta} > 1$, most of the calls have a lower duration than 1 second, which makes the CDD converges to a power law, i.e., the initial spike is truncated. Moreover, as $\alpha \to 0$, the odds ratio tends to be the constant e^{β}, what causes the variance to be infinity. By observing Figure 4.25 and concerning human calling behavior, we conjecture that β is upper bounded by 1 and ρ is lower bounded by 0.5. These values are coherent with the global intuition on human calling behavior.

Additional Correlations

Given that the vast majority of users' CDDs can be represented by the TLAC model, it would be interesting if we could predict their parameters ρ_i and β_i based on one of their summarized attributes. One could imagine that a user that makes a large number of phone calls per month might have a distinct CDD than a user that makes only a few. Moreover, we could also think



(a) CDF for ρ (b) CCDF for β

Figure 4.25: Cumulative distributions for ρ and β. We can observe that ρ is lower bounded by 0.5 and β is upper bounded by 1. These values are coherent with the global intuition on human calling behavior.

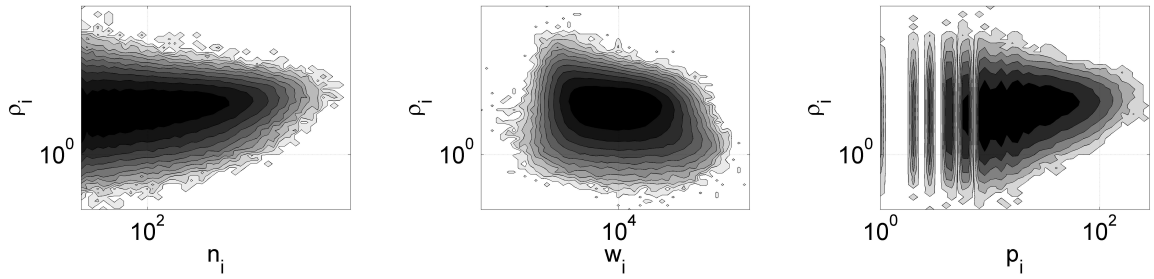
that a user that has many friends and talk to them by the phone regularly may also have a distinct CDD from a user that only talks to his family on the phone. In Figures 4.26 and 4.27, we show, respectively, the isocontours of the behavior of the ρ_i and β_i parameters for users with different values of number of phone calls n_i, aggregate duration w_i and number of partners p_i, i.e., the distinct number of persons that the user called in a month. With the exception made for the ρ_i against w_i, we observe that the variance decreases as the value of the summarized attribute increases. This suggests that the CDD of high or long talkative users, as well as users with many partners, is easier to predict. Moreover, as we can observe in the figures and also in Table 4.5, there is no significant correlation between the TLAC parameters and the summarized attributes of the users. Thus, we make the following observation:

Observation 4.3.3 *INVARIANT BEHAVIOR. The ρ_i and β_i parameters of user i behave as invariant with respect to (a) number of phone calls n_i, (b) aggregate duration w_i and (c) number of partners p_i.*

Attribute	Correlation with ρ	Correlation with β
number of phone calls	0.14	-0.18
aggregate duration	-0.21	0.01
number of partners	0.18	-0.18

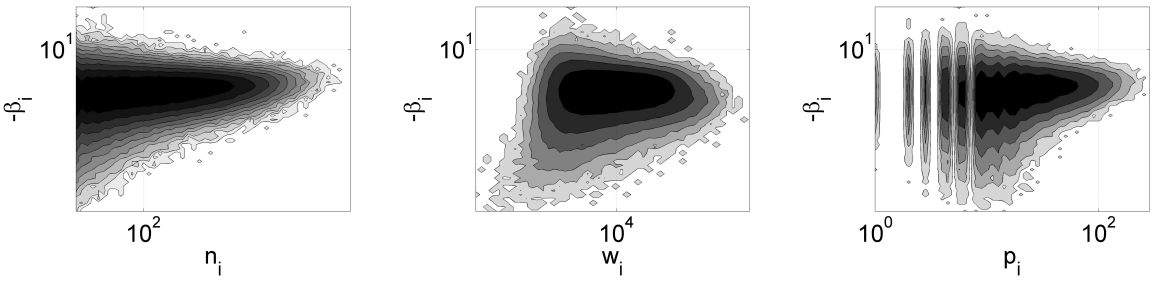
Table 4.5: Correlations between summarized attributes and ρ and β.

Finally, since there is no significant correlation between the users' CDD parameters ρ_i and β_i with their summarized attributes, we emphasize that these parameters should be considered when characterizing user behavior in phone call networks. Moreover, besides characterizing individual customers, the TLAC model can also be directly applied to the relationship between users,



(a) number of phone calls (b) aggregate duration (c) number of friends

Figure 4.26: Isocontours of the users' CDD efficiency coefficient ρ and their summarized attributes.



(a) number of phone calls (b) aggregate duration (c) number of friends

Figure 4.27: Isocontours of the users' CDD weakness coefficient β and their summarized attributes.

analyzing how two persons call each other. One could use, for instance, the ρ parameter as the weight of the edges of the social network generated from phone call records.

4.4 Summary of contributions

In this chapter we presented our work on the analysis of billions of human communication (phone call, SMS, and instant message) records of millions of users over months' of activity. Our research questions focus on understanding human communication patterns. This body of work can be grouped into three main tracks: (1) study of social circles; (2) study of reciprocal relations; and (3) study of call durations.

In particular, our main contributions are summarized as the following:

- *Social circles and related patterns:* We studied the maximal cliques in our communication networks and discovered new power-law-like patterns. More specifically, we found that the number of maximal cliques a node participates in follows a power-law relation with its degree; this translates to "popularity" growing superlinearly with the number of contacts. Related to degree distribution, thus, there exist many nodes which belong to only a few cliques while there are only a few nodes that belong to many cliques. We also found that the weights on the edges of triangles follow power-laws. Moreover, the discovered patterns are stable and persistent over time.
- *Reciprocity and 3PL model:* We found that joint distribution $\Pr(w_{ij}, w_{ji})$ of the weights on mutual edges follow a bivariate pattern for all three types of weights; number of phone calls, duration of phone calls and number of SMSs. We proposed the Triple Power Law (3PL) function to model this distribution. Our goodness of fit tests showed that 3PL provides better fits than two other well-known bivariate distributions for skewed data, the Bivariate Pareto and the Bivariate Yule.
  In addition, we took a weighted approach to quantify the *degree* of reciprocity. We observed that reciprocity is higher (1) for mutual pairs with larger local network overlap, that is, for people with more common friends; and (2) for mutual pairs with larger degree-similarity, that is, for people with similar number of contacts.
- *Call durations and TLAC model:* We proposed the TLAC distribution, which fits very well the vast majority of individual phone call durations, *much better* than log-normal and exponential. We introduced *MetaDist* which shows that the *collection* of TLAC parameters follow a striking bivariate Gaussian; *MetaDist* also remains the same over time, with very small fluctuations.

Part II

Generative Models of Networks

Chapter 5

Preliminaries

5.1 Introduction

How can we build a model that would generate graphs that look like real data? The goal here is to develop a graph generation scheme such that the output graph obeys as many of the patterns observed in real-world graphs as possible.

As we discussed in previous chapters, many fascinating properties of real-world graphs have been discovered, such as small and shrinking diameter [Albert et al., 1999; Leskovec et al., 2005b], as well as numerous power-laws [Akoglu et al., 2008; Chakrabarti et al., 2004b; Faloutsos et al., 1999; Kleinberg et al., 1999; Leskovec et al., 2005b; Mcglohon et al., 2008; Newman, 2005; Siganos et al., 2003; Tsourakakis, 2008]. As a result of such interesting patterns being discovered, and for many other reasons which we will discuss next, how to find a model that would produce synthetic but realistic graphs is a natural question to ask. There are several applications and advantages of modeling real-world graphs:

- *Simulation studies*: if we want to run tests for, say a spam detection algorithm, and want to observe how the algorithm behaves on graphs with different sizes and structural properties, we can use graph generators to produce such graphs by changing the parameters. This is also true when it is difficult to collect any kind of real data.
- *Sampling/Extrapolation*: we can generate a smaller graph for example for visualization purposes or in case the original graph is too big to run tests on it; or conversely to generate a larger graph for instance to make future prediction and answer what-if questions.
- *Summarization/Compression*: model parameters can be used to summarize and compress a given graph as well as to measure similarity to other graphs.
- *Motivation to understand pattern generating processes*: graph generators give intuition and shed light upon what kind of processes can (or cannot) yield the emergence of certain patterns. Moreover, modeling addresses the question of what patterns real networks exhibit that needs to be matched and provides motivation to figure out such properties.

Ideally, a desired graph generator should be:

1. *realistic*: it would produce graphs that obey all the discovered "laws" of real-world graphs with appropriate values.
2. *simple*: it would be easy to understand and it would intuitively lead to the emergence of macroscopic patterns.
3. *parsimonious*: it would require only a few number of parameters.
4. *flexible*: it would be able to generate the cross-product of weighted/unweighted, directed /undirected and unipartite/bipartite graphs.
5. *fast*: the generation process would ideally take linear time with respect to the number of edges in the output graph.

Earlier work on real-world networks focused on *static* snapshots and their structural properties. As a result, earlier graph generators that try to mimic real graphs are limited to capturing only these structural properties. Even then, those generators focus on modeling a *single* property of networks, and fail to mimic others. For example, the 'small-world' model [Watts and Strogatz, 1998] tries to capture the 'small diameter' phenomenon, or the 'preferential attachment' model [Barabási and Albert, 1999] seeks to generate skewed degree distributions.

In the first chapter of this part, we focus on building generators, capturing *dynamic* and *weighted*, as well as static structural properties. We first introduce the *Recursive Tensor Model* (RTM) that generates weighted, time-evolving graphs. RTM is a generalization of Kronecker graph generators [Leskovec et al., 2005a] to the time-evolving setting. While it is mathematically tractable, it comes with some disadvantages of Kronecker models; namely, multinomial/lognormal (instead of power-law) distributions and fixed number of nodes (see related work 5.2 for details). Next, we introduce the *Random Typing Graphs* (RTG) that uses a process of 'random typing', to generate source and destination node identifiers. It overcomes the two shortcomings of RTM and it meets all the above desired properties. In fact, we show that it can generate graphs that obey all *eleven* patterns that real graphs were observed to typically exhibit to date.

In the second chapter of this part, we focus on building a generator, capturing human communication patterns. We introduce the *Pay and Call* (PaC) model, which is a utility-driven generator that models the way in which humans decide when and whom to contact. Our guiding principle is that humans balance a trade-off between the cost of the communication (in time and money), and its benefit (in valuable information and emotional support).

The following two chapters in this part are based on work as cited below.

- Chapter 6 Models of graph topology [Akoglu and Faloutsos, 2009; Akoglu et al., 2008]
- Chapter 7 Model of human communications [Du et al., 2009].

5.2 Related Work

Modeling real-world graphs successfully is an elusive task. A vast majority of earlier graph generators have focused on modeling a small number of common properties, but fail to mimic others. The very first generative model was proposed by Erdos & Renyi [Erdos and Renyi, 1960].

The model begins with a fixed number of nodes, and adds edges, where any pair of nodes has the same and independent probability of being linked by an edge. While it has some interesting provable properties, it fails to produce a number of realistic properties, most notably the heavy-tailed degree distribution. Another striking generator is the *'preferential attachment'* model [Barabási and Albert, 1999], where at each time step nodes are added and 'prefer' to link to high-degree nodes. This in turn leads to small diameter and heavy-tailed degree distributions; however, this and related models lack the *shrinking diameter* property. There exists a group of other generators such as the *'small-world'* [Watts and Strogatz, 1998], *'winners don't take all'* [Pennock et al., 2002], *'forest fire'* [Leskovec et al., 2005b], and *'butterfly'* [Mcglohon et al., 2008] models. In addition, recursive models using Kronecker multiplication have proved useful for generating self-similar properties of graphs [Leskovec et al., 2005a]. [Chakrabarti and Faloutsos, 2006] provides a detailed survey on graph generators. In general, these methods are limited in trying to model some static graph property while neglecting others as well as *dynamic* properties or cannot be generalized to produce *weighted* graphs.

Kronecker graph generators [Leskovec et al., 2005a] are successful in the sense that they match several of the properties of real graphs and they have proved useful for generating self-similar properties of graphs. However, they have two disadvantages: The first is that they generate multinomial/lognormal distributions for their degree and eigenvalue distribution, instead of a power-law one. The second disadvantage is that it is not easy to grow the graph incrementally: They have a fixed, predetermined number of nodes (say, N^k, where N is the number of nodes of the generator graph, and k is the number of iterations); where adding more edges than expected does *not* create additional nodes.

Random dot product graphs [Kraetzl M., 2005; Young and Scheinerman, 2007] assign each vertex a random vector in some d-dimensional space and an edge is put between two vertexes with probability equal to the dot product of the endpoints. This model does not generate weighted graphs and by definition only produces undirected graphs. It also seems to require the computation of the dot product for each pair of nodes which takes *quadratic* time.

A different family of models, often referred to as *games of network formation* and that are mainly from the fields of economics and game theory, are utility-based. In such models, there exist agents that try to optimize a predefined utility function and the network structure takes shape from their collective strategic behavior. [Laoutaris et al., 2008] proposes a network formation game, where links have costs and lengths, and players have preference weights on the other players, to study the properties of pure Nash equilibria [Nash, 1951] in different settings. [Albers et al., 2006], [Demaine et al., 2009], and [Fabrikant et al., 2003] study a similar game where players do not have fixed budgets and the cost function is defined in terms of the sum of the number of edges. [Even-Dar et al., 2007] proposes a network creation game where nodes act as buyers and sellers such that the resulting graphs are bipartite. This class of models, however, is usually hard to analyze.

In this body of work, in contrast to previous work, we explicitly focus on generators for time-evolving as well as weighted graphs. Moreover, our models are often mathematically tractable.

Chapter 6

Models of network topology

PROBLEM STATEMENT: *How could we have a realistic generative model that will produce synthetic graphs that look like real, i.e. graphs that obey all the patterns we know so far, as well as the newly discovered ones for dynamic and weighted graphs?*

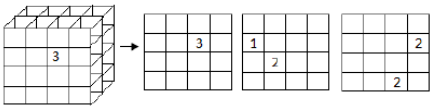
In this chapter, we present two generative models: (1) Recursive Tensor Model, which uses Kronecker tensor product to produce time-evolving weighted graphs, and (2) Random Typing Graphs, based on a simple random typing procedure that mimics human behavior well.

6.1 RTM: Recursive Tensor Model

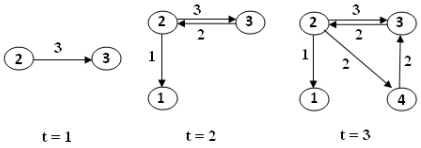
The high level idea behind our Recursive Tensor Model is to use recursion, in conjunction with tensors (n-dimensional extension of matrices). Recursion and self-similarity naturally leads to modular network behavior ("communities-within-communities"), power laws [Leskovec et al., 2005a], as well as bursty traffic [Wang et al., 2002]. Earlier work used self-similarity to generate static snapshots of unweighted graphs [Chakrabarti et al., 2004b]. Here, we show how to build a generator that will also match dynamic and weighted properties.

The main idea is to use recursion not only on the adjacency matrix, but also on the *time* dimension. Specifically, we start with a small tensor $\mathcal{I}$ that has 3 sides ('modes'): (a) senders (b) recipients and (c) time. We call the graph represented by a tensor a 't-graph' that evolves over time (see Fig. 6.1 (a-b)). Then, we recursively substitute every cell (i, j, t) of the original tensor $\mathcal{I}$, with a copy of itself, and multiply it with the value $a_{i,j,t}$ (see Fig. 6.1 (c) for illustration and Definition 2 for full details). Thanks to the self-similarity of the construct, we expect the resulting tensor to have all the properties we want.

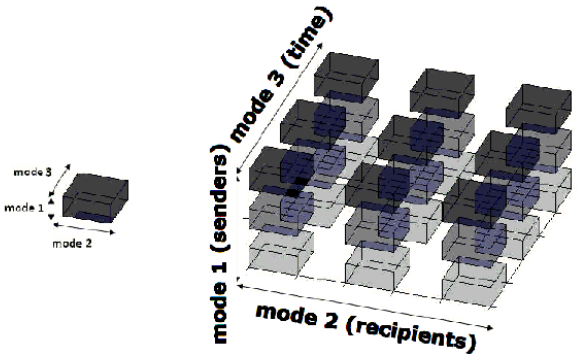
First, we give the details of the construction. Secondly, we provide proofs to show that our model will generate graphs with desired properties. Finally, we give our experimental results.



(a) $(4 \times 4 \times 3)$ tensor $\mathcal{T} \rightarrow$ t-slices



(b) corresponding t-graph over time



(c) Kronecker product of $\mathcal{T}$ by itself

Figure 6.1: (a) An example for the initial tensor $\mathcal{T}$ of size $(4 \times 4 \times 3)$. The 't-slices' represent the changes on the adjacency matrix at every other time step. (b) The corresponding graph represented by the tensor in part (a). It changes according to the 't-slices' over time. (c) Kronecker product of $\mathcal{T}$ by itself produces a self-similar tensor.

6.1.1 Model Description

Throughout this section, we use the following conventions: uppercase bold letters for matrices, lowercase bold letters for vectors, lowercase letters for scalars, and calligraphic uppercase letters for tensors. A list of symbols used is listed in Table 6.1.

Symbol	Description
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	Tensors used to illustrate recursive tensor product
$a_{i,j,k}$	Entry of a tensor
$\mathcal{I}$	Initial tensor in RTM model
$\mathcal{G}_{\mathcal{A}}$	t-graph (time-evolving graph) represented by tensor $\mathcal{A}$
$\mathbf{D}_t$	t^{th} slice of final tensor $\mathcal{D}$ in RTM
s_t	Total weight of $\mathbf{D}_t$
e_t	Number of edges of $\mathbf{D}_t$
$W_{\mathcal{D}}$	Total weight of a tensor $\mathcal{D}$, or $\sum_t s_t$
$\mathbf{s}_{\mathcal{D},r}$	Temporal profile of $\mathcal{D}$ at resolution r
$\mathbf{p}_{\mathcal{D},r}$	Normalized temporal profile of $\mathcal{D}$ at resolution r

Table 6.1: Table of symbols used in RTM notation.

For the construction, we choose an initial $(N \times N \times \tau)$ tensor $\mathcal{I}$ with nonzero cells (i, j, t) indicating an edge from node i to node j at time tick t. We initialize the cells so that the initial t-graph(t- for time-evolving) $\mathcal{G}_{\mathcal{I}}$ represented by $\mathcal{I}$ looks like a miniature real-world graph. We provide details for how to initialize $\mathcal{I}$ in Section 5.3.

Note that RTM works for both directed and bipartite t-graphs. A bipartite t-graph can be represented by an $(N \times M \times \tau)$ tensor. For simplicity, we focus on unipartite graphs in our work.

We propose to use *Recursive Tensor Multiplication* to produce a time-evolving graph. Our method extends Kronecker product [1] of two matrices by adding a third 'mode'. Kronecker product of two matrices is defined as follows: Given two matrices **A** and **B** of sizes $(N \times M)$ and $(N' \times M')$, respectively, the Kronecker product of **A** and **B**, namely matrix **C** of dimension $(N * N') \times (M * M')$ is given by

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} \quad = \quad \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,M}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,M}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1}\mathbf{B} & a_{N,2}\mathbf{B} & \dots & a_{N,M}\mathbf{B} \end{pmatrix}$$

Definition 2 (Recursive Tensor Multiplication) *Given two tensors $\mathcal{A}$ of size $(N \times M \times \tau)$ and $\mathcal{B}$ of size $(N' \times M' \times \tau')$, Recursive Tensor Multiplication $\mathcal{C}$ of $\mathcal{A}$ and $\mathcal{B}$ is obtained by replacing each cell $a_{i,j,t}$ of tensor $\mathcal{A}$ with $a_{i,j,t} * \mathcal{B}$. The resulting tensor $\mathcal{C}$ is of size $(N * N') \times (M * M') \times (\tau * \tau')$ such that*

$$c_{((i-1)*N+i'),((j-1)*M+j'),((k-1)*\tau+k')} = a_{i,j,k} * b_{i',j',k'}.$$

An example of the Recursive Tensor Multiplication of a $(3 \times 3 \times 3)$ tensor by itself is given in Fig. 6.1 (c).

To generate a growing graph over time, we get the *'Recursive Tensor Multiplication'* of the initial $(N \times N \times \tau)$ tensor $\mathcal{I}$ by itself k times as:

$$\mathcal{I}^k = \mathcal{D} = \underbrace{\mathcal{I} \textcircled{t} \mathcal{I} \textcircled{t} \dots \textcircled{t} \mathcal{I}}_{k \; times}$$

and then we take the final tensor $\mathcal{D}$ to represent our data. The data spans τ^k number of time ticks with N^k nodes. At every time step t ($t = \{1, 2, ..., \tau^k\}$), we get the t-slice (see Definition 3 below) $\mathbf{D}_t$ of $\mathcal{D}$, and for each nonzero cell $a_{i,j}$ of $\mathbf{D}_t$, we add an edge between node i and node j with weight $a_{i,j}$. If the edge already exists, we increase the weight $w_{i,j}$ by the same amount.

Initializing $\mathcal{I}$

In order to take advantage of the self-similarity property of our construct, we want the initial graph $\mathcal{G}_\mathcal{I}$ to be a realistic graph itself. Basically, one can use any graph generator in the literature that is known to produce realistic graphs [Erdos and Renyi, 1960; Leskovec et al., 2005b; Watts and Strogatz, 1998] to generate $\mathcal{G}_\mathcal{I}$.

To our knowledge, RTM is the first *weighted* graph generator; thus we also take weights into consideration as follows. We force the initial graph to obey the WPL at all time ticks. That is, when a link occurs between two nodes, we put weight on the edge, so that number of edges $E(t)$ and total weight $W(t)$ over time follow a power law, with a user-specified exponent α.

[1] Unfortunately, Kronecker product **C** of two matrices **A** and **B** is also called Kronecker Tensor multiplication, despite **A**, **B**, **C** are matrices. To disambiguate, we use the name RTM where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ are in fact tensors.

Having given the details of the construction, next we give proofs for the characteristics that RTM will generate. Before that, we define the terms we use throughout the proofs.

Definition 3 (t-slice of a tensor $\mathcal{T}$) *Given a tensor $\mathcal{T}$ of size $(N \times M \times \tau)$, t-slice of $\mathcal{T}$ is a matrix $\boldsymbol{T}_t$ such that*

$$\boldsymbol{T}_t \equiv \mathcal{T}(i, j, t), \quad \forall i, \forall j, \quad 1 \leq i \leq N, 1 \leq j \leq M$$

Definition 4 ((Normalized) temporal (t-) profile of $\mathcal{T}$) *Given a tensor $\mathcal{T}$ of size $(N \times M \times \tau)$, let s_t denote the total weight of its t-slice. Then, the t-profile of $\mathcal{T}$ is a $(1 \times \tau)$ vector, such that*

$$\mathbf{s}_{\mathcal{T},0} \equiv (s_1, s_2, \ldots, s_\tau)$$

Total weight $W_\mathcal{T}$ of $\mathcal{T}$ can be written as $\sum_{t=1}^{\tau} s_t$. Then, normalized t-profile of $\mathcal{T}$ is a $(1 \times \tau)$ vector, such that

$$\mathbf{p}_{\mathcal{T},0} \equiv (\frac{s_1}{W_\mathcal{T}}, \frac{s_2}{W_\mathcal{T}}, \ldots, \frac{s_\tau}{W_\mathcal{T}})$$

6.1.2 Theorems and Proofs

Recursive Tensor graphs can be shown to exhibit several real-world graph properties. In particular, if we choose the initial graph to be a miniature of a real graph, after recursive iterations of RTM, the resulting graph will follow similar properties as of the initial graph due to self-similarity of the construction.

Theorem 1 (Self-similar and Bursty Edge/Weight Additions) *Let edge/weight additions for $\mathcal{I}$ with $\mathbf{p}_{\mathcal{I},0}$ be self-similar and bursty for which the slope of the entropy plot is*

$$slope = H(\mathbf{p}_{\mathcal{I},0}) = -\sum_{i=1}^{\tau} \mathbf{p}_{\mathcal{I},0}(i) log_2(\mathbf{p}_{\mathcal{I},0}(i)),$$

After k iterations of RTM, edge/weight arrivals over time for $\mathcal{D}$ are also self-similar and bursty. The slope of the entropy plot over all *aggregation levels r of $\mathcal{D}$ is equal to*

$$slope = H(\mathbf{p}_{\mathcal{D},r}) = H(\mathbf{p}_{\mathcal{I},0}), \quad \forall r$$

where $H(\mathbf{p}_{\mathcal{D},r})$ is the slope of the entropy plot at aggregation level r. Furthermore, the slope does not change with the value of k, that is, burstiness is independent of scale.

Proof We will prove for weight additions and similar arguments apply for edge additions. After k iterations of *RTM*, total weight of $\mathcal{D}$ becomes

$$W_\mathcal{D} = W_\mathcal{I}^k = (s_1 + s_2 + \ldots + s_\tau)^k$$

At aggregation level (resolution) 1, we group slices by τ^{k-1} into τ groups. Then, $W_\mathcal{D}$ can be written as

$$W_\mathcal{D} = s_1 * W_\mathcal{I}^{k-1} + s_2 * W_\mathcal{I}^{k-1} + \ldots + s_\tau * W_\mathcal{I}^{k-1}$$

Then for $1 \leq t \leq \tau$,

$$\mathbf{p}_{\mathcal{D},1}(t) = \frac{s_t * W_{\mathcal{I}}^{k-1}}{W_{\mathcal{I}}^{k}} = \frac{s_t}{W_{\mathcal{I}}} = \mathbf{p}_{\mathcal{I},0}(t)$$

At aggregation level 2, we group slices by τ^{k-2} into τ^2 groups. Then,

$$\begin{aligned} W_{\mathcal{D}} &= s_1 * (s_1 * W_{\mathcal{I}}^{k-2} + s_2 * W_{\mathcal{I}}^{k-2} + \ldots + s_\tau * W_{\mathcal{I}}^{k-2}) \\ &+ s_2 * (s_1 * W_{\mathcal{I}}^{k-2} + s_2 * W_{\mathcal{I}}^{k-2} + \ldots + s_\tau * W_{\mathcal{I}}^{k-2}) \\ &+ \qquad\qquad \ldots \\ &+ s_\tau * (s_1 * W_{\mathcal{I}}^{k-2} + s_2 * W_{\mathcal{I}}^{k-2} + \ldots + s_\tau * W_{\mathcal{I}}^{k-2}) \end{aligned}$$

For any slice i at level 1 and t at level 2, $1 \leq t \leq \tau^2$,

$$\mathbf{p}_{\mathcal{D},2}(t) = \frac{s_i * s_t * W_{\mathcal{I}}^{k-2}}{s_i * W_{\mathcal{I}}^{k-1}} = \frac{s_t}{W_{\mathcal{I}}} = \mathbf{p}_{\mathcal{I},0}(t)$$

Finally, at aggregation level k, we group slices by τ^0 into τ^k groups as

$$\begin{aligned} W_{\mathcal{D}} &= (s_1)^{k-1} * (s_1 + s_2 + \ldots + s_\tau) \\ &+ (s_1)^{k-2} * s_2 * (s_1 + s_2 + \ldots + s_\tau) \\ &+ \qquad\qquad \ldots \\ &+ (s_\tau)^{k-1} * (s_1 + s_2 + \ldots + s_\tau) \end{aligned}$$

For all combinations of $(k-1)$ slices at levels from 1 to $(k-1)$, let c_j denote the corresponding coefficients, $1 \leq j \leq \tau^{k-1}$, and for any slice t at level k, $1 \leq t \leq \tau^k$

$$\mathbf{p}_{\mathcal{D},k}(t) = \frac{c_j * s_t}{c_j * W_{\mathcal{I}}} = \frac{s_t}{W_{\mathcal{I}}} = \mathbf{p}_{\mathcal{I},0}(t)$$

We showed that normalized t-profile, $\mathbf{p}_{\mathcal{D},r}$, of $\mathcal{D}$ remains the same at all aggregation levels r and is equal to that of the initial tensor $\mathcal{I}$. So, we conclude that bias of burstiness for $\mathcal{D}$ is the same as that of $\mathcal{I}$ for all values of k, since $H(\mathbf{p}_{\mathcal{D},r})$ would not change for $\forall r$. ∎

As an example, starting with a $(10 \times 10 \times 2)$ $\mathcal{I}$, where $\mathbf{p}_{\mathcal{I},0}(1) : \mathbf{p}_{\mathcal{I},0}(2) = 0.175 : 0.825$; after $k = 3$ iterations, the slope of the entropy plot is obtained to be 0.669, which is equal to

$$H(\mathbf{p}_{\mathcal{I},0}) = .175 * log_2(.175) + .825 * log_2(.825)$$

See Fig. 6.3 (c).

Theorem 2 (Weight Power Law (WPL)) *If the initial graph $\mathcal{G}_{\mathcal{I}}$ exhibits the WPL at* all *time ticks, that is, number of edges $E(t)$ and total weight $W(t)$ over time follow a power law with exponent α, $\mathcal{G}_{\mathcal{D}}$ shows the same property at time ticks $1, \tau^1, \tau^2, \ldots, \tau^k$ with exactly the same exponent α.*

Proof We are given the condition that $\mathcal{G}_{\mathcal{I}}$ follows the *WPL* at *all* time ticks, that is,

$$e_1^{\alpha} = s_1, (e_1 + e_2)^{\alpha} = (s_1 + s_2), \ldots, (\sum_{t=1}^{\tau} e_t)^{\alpha} = (\sum_{1}^{\tau} s_t).$$

After k iterations of RTM, the resulting graph has $E^k = (\sum_{t=1}^{\tau} e_t)^k$ edges and $W_{\mathcal{I}}^k = (\sum_{t=1}^{\tau} s_t)^k$ total weight. At $t = \tau^k$,

$$E^{\alpha} = W_{\mathcal{I}} \Rightarrow (E^{\alpha})^k = W_{\mathcal{I}}^k \Rightarrow (E^k)^{\alpha} = W_{\mathcal{I}}^k$$

At aggregation level 1, E^k can be written as,

$$E^k = e_1 * E^{k-1} + e_2 * E^{k-1} + \ldots + e_{\tau} * E^{k-1}$$

Same argument holds for $W_{\mathcal{I}}^k$. So, at $t = \tau^{k-1}$, number of edges is $(e_1 * E^{k-1})$ and total weight is $(s_1 * W_{\mathcal{I}}^{k-1})$. And,

$$(e_1 * E^{k-1})^{\alpha} = (e_1)^{\alpha} * (E^{\alpha})^{k-1} = s_1 * W_{\mathcal{I}}^{k-1}$$

In general, at every aggregation level r, at $t = \tau^r$, the graph follows the *WPL* as

$$(e_1^r * E^{k-r})^{\alpha} = (e_1^{\alpha})^r * (E^{\alpha})^{k-r} = s_1^r * W_{\mathcal{I}}^{k-r}$$

■

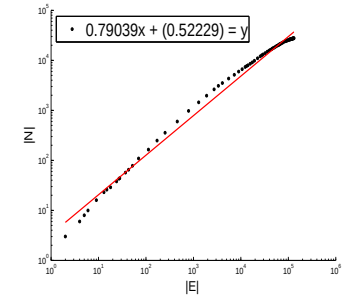
We observe that when we interpolate total weight $W(t)$ versus number of edges $E(t)$ at all time ticks $t \in \{1, 2, \ldots, \tau^k\}$ for the final graph $\mathcal{G}_{\mathcal{D}}$, the resulting exponent remains very close to α. In Fig. 6.3 (b), the user-specified WPL exponent is 1.5, which is equal to the slope when points at $t = \{1, 2, 4, 8\}$ are used to fit a line ($k = 3$). When all points are used, the slope is 1.47.

6.1.3 Model Validation and Analysis

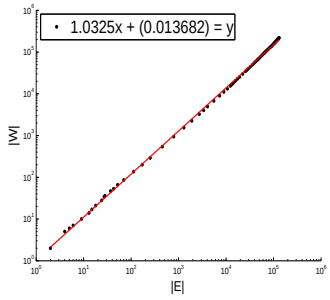
Having created the initial tensor, we take the Recursive Tensor Multiplication of $\mathcal{I}$ by itself k times. The final tensor $\mathcal{D}$ is a $(N^k \times N^k \times \tau^k)$ tensor spanning τ^k time ticks. At every time tick t, we take the t-slice $\mathbf{D}_t$ of $\mathcal{D}$. Next, we simply introduce an edge from node i to node j with weight $a_{i,j}$ for every nonzero entry of $\mathbf{D}_t$. If node i or node j did not exist, we introduce new node(s). If both existed, we only increase the edge weight by $a_{i,j}$.

We did several experiments for different values of N, τ and k. Our model produced realistic graphs for a wide range of parameters, the results being independent of the number of iterations k. In fact, k can be chosen as large as the size of the graph that one needs to generate.

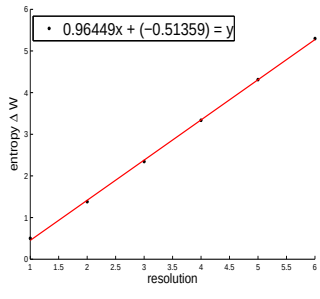
As a comparison with real-world data, we give the plots showing reported laws for *BlogNet* in Fig. 6.2. The plots our model generated for $N = 10$, $\tau = 2$ and $k = 3$ are shown in Fig. 6.3. In particular, we show (a) the Densification Power Law (DPL); (b) the Weight Power Law (WPL); (c) bursty weight additions; (d) the λ_1 Power Law (LPL), (e) the $\lambda_{1,w}$ Power Law and finally, (f) the Edge Weight Power Law (EWPL). See Table 2.2 and Chapter 3 for details on these laws. Other desired characteristics such as small and shrinking diameter, the gelling point and the Degree Power Law for degree distribution of nodes are also matched, but omitted here for brevity.



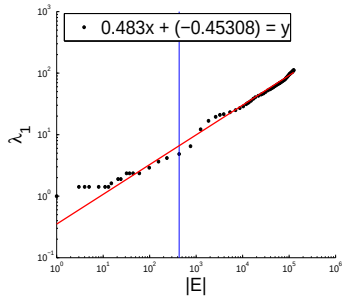
(a) Densification Power Law



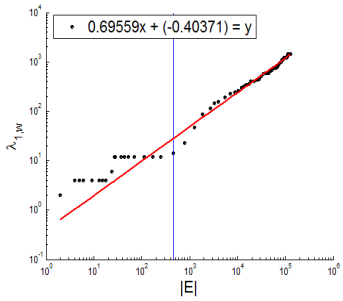
(b) Weight Power Law



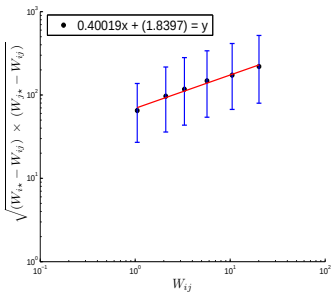
(c) ΔW entropy



(d) λ_1 Power Law

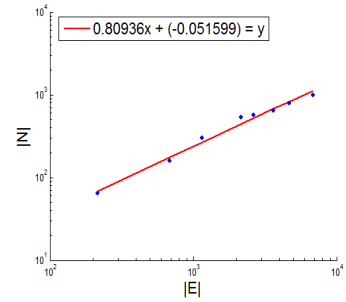


(e) $\lambda_{1,w}$ Power Law

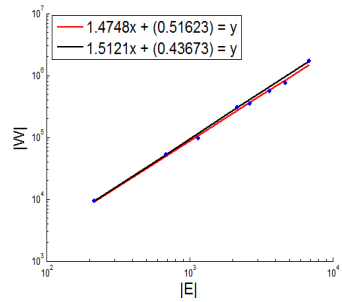


(f) Edge Weight Power Law

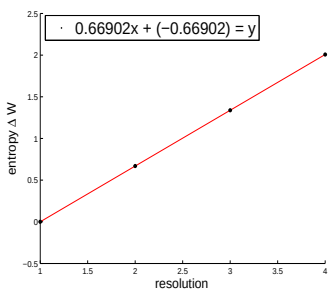
Figure 6.2: Plots showing related laws that real-world graphs obey for *BlogNet*. First row shows previous laws while second row shows observations of this work.



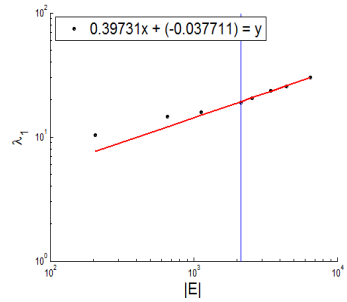
(a) Densification Power Law



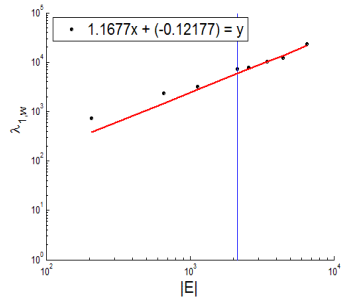
(b) Weight Power Law



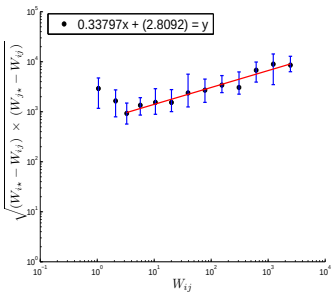
(c) ΔW entropy



(d) λ_1 Power Law



(e) $\lambda_{1,w}$ Power Law



(f) Edge Weight Power Law

Figure 6.3: Plots showing related laws our RTM generator produced. Notice that they are very similar in all the listed properties to those of *BlogNet*.

6.2 RTG: Random Typing Graphs

Zipf introduced probably the earliest power law [Zipf, 1932], stating that, in many natural languages, the rank r and the frequency f_r of vocabulary words follow a power-law $f_r \propto 1/r$. [Mandelbrot, 1953] argued that Zipf's law is the result of optimizing the average amount of information per unit transmission cost. [Miller, 1957] showed that a random process also leads to Zipf-like power laws. He suggested the following experiment: "A monkey types randomly on a keyboard with k characters and a space bar. A space is hit with probability q; all other characters are hit with equal probability, $\frac{(1-q)}{k}$. A space is used to separate words". The distribution of the resulting words of this random typing process follow a power-law. Conrad and Mitzenmacher [Conrad and Mitzenmacher, 2004] showed that this relation still holds when the keys are hit with *unequal* probability.

Our RTG model generalizes the above model of natural human behavior, using "random typing". We build the model in *three* steps, incrementally. In the next two steps, we introduce the base version of the proposed model to give an insight. However, as will become clear, it has *two* shortcomings. In particular, the base model does not capture (1) homophily, the tendency to associate and bond with similar others- people tend to be acquainted with others similar in age, class, geographical area, etc. and (2) community structure, the existence of groups of nodes that are more densely connected internally than with the rest of the graph.

6.2.1 Initial RTG with Independent Equiprobable keys

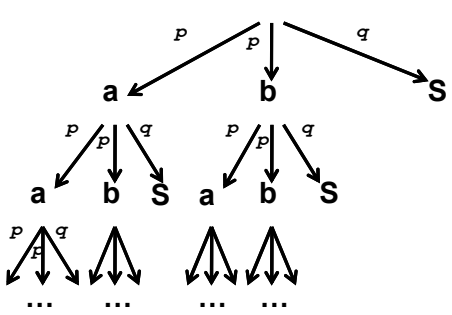
As in Miller's experimental setting, we propose each unique word typed by the monkey to represent a node in the output graph (one can think of each unique word as the label of the corresponding node). To form links between nodes, we mark the sequence of words as 'source' and 'destination', alternatingly. That is, we divide the sequence of words into groups of two and link the first node to the second node in each pair. If two nodes are already linked, the weight of the edge is simply increased by 1. Therefore, if W words are typed, the total weight of the output graph is $W/2$. See Figure 6.4 for an example illustration. Intuitively, random typing introduces new nodes to the graph as more words are typed, because the possibility of generating longer words increases with increasing number of words typed.

Due to its simple structure, this model is very easy to implement and is indeed mathematically tractable. If W words are typed on a keyboard with k keys and a space bar, the probability p of hitting a key being the same for all keys and the probability of hitting the space bar being denoted as q=$(1 - kp)$:

Lemma 2 *The expected number of nodes N in the output graph G of the RTG-IE model is*

$$N \propto W^{-log_p k}.$$

Proof Given the number of words W, we want to find the expected number of nodes N that the RTG-IE graph consists of. This question can be reformulated as follows: "Given W words



Time	Source	Destination	Weight
T1	ab	a	1
T2	bba	ab	1
T3	b	ab	1
T4	a	b	1
T5	ab	a	1

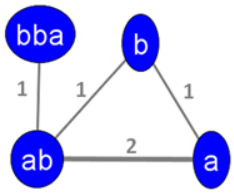


Figure 6.4: Illustration of the RTG-IE. Upper left: how words are (recursively) generated on a keyboard with two equiprobable keys, 'a' and 'b', and a space bar; lower left: a keyboard is used to randomly type words, separated by the space character; upper right: how words are organized in pairs to create source and destination nodes in the graph over time; lower right: the output graph; each node label corresponds to a unique word, while labels on edges denote weights.

typed by a monkey on a keyboard with k keys and a space bar, what is the size of the vocabulary V?" The number of unique words V is basically equal to the number of nodes N in the output graph.

Let w denote a single word generated by the defined random process. Then, w can recursively be written as follows: "$w : c_i w|S$", where c_i is the character that corresponds to key i, $1 \leq i \leq k$, and S is the space character. So, V as a function of model parameters can be formulated as:

$$\begin{aligned} V(W) &= V(c_1, Wp) + V(c_2, Wp) + \ldots + V(c_k, Wp) + V(S) \\ &= k * V(Wp) + V(S) = \quad k * V(Wp) + \begin{cases} 1, 1-(1-q)^W \\ 0, (1-q)^W \end{cases} \end{aligned}$$

where q denotes the probability of hitting the space bar, i.e. $q = 1 - kp$. Given the fact that W is often large, and $(1-q) < 1$, it is almost always the case that w=S is generated; but since this adds only a constant factor, we can ignore it in the rest of the computation. That is,

$$V(W) \approx k * V(Wp) = k * (k * V(Wp^2)) = k^n * V(1)$$

where $n = log_p(1/W) = -log_p W$. By definition, when W=1, that is, in case only one word is generated, the vocabulary size is 1, i.e. V(1)=1. Therefore,

$$V(W) = N \propto k^n = k^{-log_p W} = W^{-log_p k}.$$

■

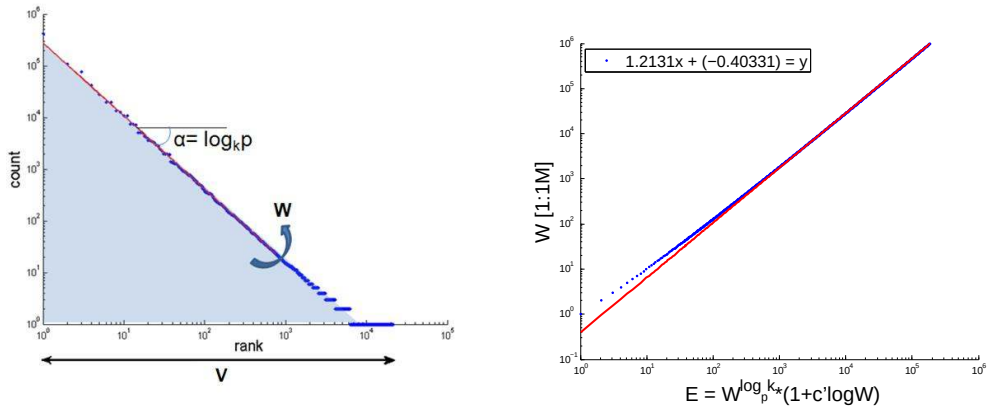


Figure 6.5: (a) Rank vs count of vocabulary words typed randomly on a keyboard with k equiprobable keys (with probability p) and a space bar (with probability q), follow a power law with exponent $\alpha = log_k p$. Approximately, the area under the curve gives the total number of words typed. (b) The relationship between number of edges E and total weight W behaves like a power-law (k=2, p=0.4).

The above proof shown using recursion is in agreement with the early result of [Miller, 1957], who showed that in the monkey-typing experiment with k equiprobable keys (with probability p) and a space bar (with probability q), the rank-frequency distribution of words follow a power law. In particular,

$$f(r) \propto r^{-1+log_k(1-q)-1} = r^{log_k p}.$$

In this case, the number of ranks corresponds to the number of unique words, that is, the vocabulary size V. And, the sum of the counts of occurrences of all words in the vocabulary should give W, the number of words typed. The total count can be approximated by the area under the curve on the rank-count plot. See Figure 6.5 (a). Next, we give a second proof of Lemma 2 using Miller's result.

Proof 2 Let $\alpha = log_k p$ and $C(r)$ denote the number of times that the word with rank r is typed. Then, $C(r) = cr^{\alpha}$, where $C(r)_{min} = C(V) = cV^{\alpha}$ and the constant $c = C(V)V^{-\alpha}$. Then we can write W as

$$\begin{aligned} W &= C(V)V^{-\alpha}\left(\sum_{r=1}^{V} r^{\alpha}\right) \approx C(V)V^{-\alpha}\left(\int_{r=1}^{V} r^{\alpha} dr\right) = C(V)V^{-\alpha}\left(\frac{r^{\alpha+1}}{\alpha+1}\Big|_{r=1}^{V}\right) \\ &= C(V)V^{-\alpha}\left(\frac{1}{-\alpha-1} - \frac{1}{(-\alpha-1)V^{-\alpha-1}}\right) \approx c'V^{-\alpha}. \end{aligned}$$

where $c' = \frac{C(V)}{-\alpha-1}$, where $\alpha < -1$ and $C(V)$ is very small (usually 1). Therefore,

$$V = N \propto W^{-\frac{1}{\alpha}} = W^{-log_p k}.$$

∎

Lemma 3 *The expected number of edges E in the output graph G of the RTG-IE model is*

$$E \approx W^{-log_p k} * (1 + c' log W), \quad for\ c' = \frac{q^{-log_p k}}{-log p} > 0.$$

Proof Given the number of words W, we want to find the expected number of edges E that the RTG-IE graph consists of. The number of edges E is the same as the unique number of *pairs* of words. We can think of a pair of words as a single word e, the generation of which is stopped after the *second* hit to the space bar. So, e always contains a single space character. Recursively, "$e : c_i e|Sw$", where "$w : c_i w|S$". So, E can be formulated as:

$$E(W) = k * E(Wp) + V(Wq) \tag{6.1}$$

$$V(Wq) = k * V(Wqp) + \begin{cases} 1, 1 - (1-q)^{Wq} \\ 0, (1-q)^{Wq} \end{cases} \tag{6.2}$$

From Lemma 2, Equation (6.2) can be approximately written as $V(Wq) = (Wq)^{-log_p k}$. Then, Equ.(1) becomes $E(W) = k * E(Wp) + cW^{\alpha}$, where $c = q^{-log_p k}$ and $\alpha = -log_p k$. Given that E(W=1)=1, we can solve the recursion as follows:

$$\begin{aligned} E(W) &\approx k * (k * E(Wp^2) + c(Wp)^{\alpha}) + cW^{\alpha} \\ &= k * (k * (k * V(Wp^3) + c(Wp^2)^{\alpha}) + c(Wp)^{\alpha}) + cW^{\alpha} \\ &= k^n * V(1) + k^{n-1} * c(Wp^{n-1}) + k^{n-2} * c(Wp^{n-2})^{\alpha} + \ldots + cW^{\alpha} \\ &= k^n * V(1) + cW^{\alpha}((kp^{\alpha})^{n-1} + (kp^{\alpha})^{n-2} + \ldots + 1) \end{aligned}$$

where $n = log_p(1/W) = -log_p W$. Since $kp^{\alpha} = kp^{-log_p k} = 1$,

$$E(W) \approx k^n * V(1) + n * cW^{\alpha} = k^{-log_p W} + c\frac{-log\frac{1}{W}}{-log p} W^{-log_p k} = W^{-log_p k}(1 + c' log W)$$

where $c' = \frac{c}{-log p} = \frac{q^{-log_p k}}{-log p} > 0$. ∎

The above function of E in terms of W and other model parameters looks like a power-law for a wide range of W. See Figure 6.5 (b).

Lemma 4 *The in(out)-degree d_n of a node in the output graph G of the RTG-IE model is power law related to its total in(out)-weight W_n, that is,*

$$W_n \propto d_n^{-log_k p}$$

with expected exponent $-log_k p > 1$.

Proof We will show that $W_n \propto d_n^{-log_k p}$ for out-edges, and a similar argument holds for in-edges. Given that the experiment is repeated W times, let W_n denote the number of times a unique word is typed as a source. Each such unique word corresponds to a node in the final graph and W_n is basically its out-weight, since the node appears as a source node. Then, the out-degree d_n of a node is simply the number of unique words typed as a destination. From Lemma 2,

$$W_n \propto d_n^{-log_k p}, \;\; for \;\; -log_k p > 1.$$

■

Even though most of the properties listed at the beginning of this section are matched, there are two problems with this model: (1) the degree distribution follows a power-law only for small degrees and then shows multinomial characteristics (see Figure 6.6 (top)), and (2) it does not generate homophily and community structure, because it is possible for every node to get connected to every other node, rather than to 'similar' nodes in the graph.

6.2.2 Initial RTG with Independent Un-equiprobable keys

We can spread the degrees so that nodes with the same-length but otherwise distinct labels would have different degrees by making keys have *unequal* probabilities. This procedure introduces smoothing in the distribution of degrees, which remedies the first problem introduced by the RTG-IE model. In addition, thanks to [Conrad and Mitzenmacher, 2004], we are still guaranteed to obtain the desired power-law characteristics as before (see Figure 6.6 (bottom)).

6.2.3 Proposed RTG: Model Description

What the previous model fails to capture is the homophily and community structure. In a real network, we would expect nodes to get connected to similar nodes (homophily), and form groups and possibly groups within groups (modular structure). In our model, for example on a keyboard with two keys 'a' and 'b', we would like nodes with many 'a's in their labels to be connected to similar nodes, as opposed to nodes labeled with many 'b's. However, in both RTG-IE and RTG-IU it is possible for every node to connect to every other node. In fact, this yields a tightly connected core of nodes with rather short labels.

Our proposal to fix this is to envision a two-dimensional keyboard that generates source and destination labels in one shot, as shown in Figure 6.7. The previous model generates a word for source, and, completely independently, another word for destination. In the example with two keys, we can envision this process as picking one of the nine keys in Figure 6.7 (a), using the independence assumption: the probability for each key is the product of the probability of the corresponding row times the probability of the corresponding column: p_l for letter l, and q for space ('S'). After a key is selected, its row character is appended to the source label, and the column character to the destination label. This process repeats recursively as in Figure 6.7 (b), until the space character is hit on the first dimension in which case the source label is terminated and also on the second dimension in which case the destination label is terminated.

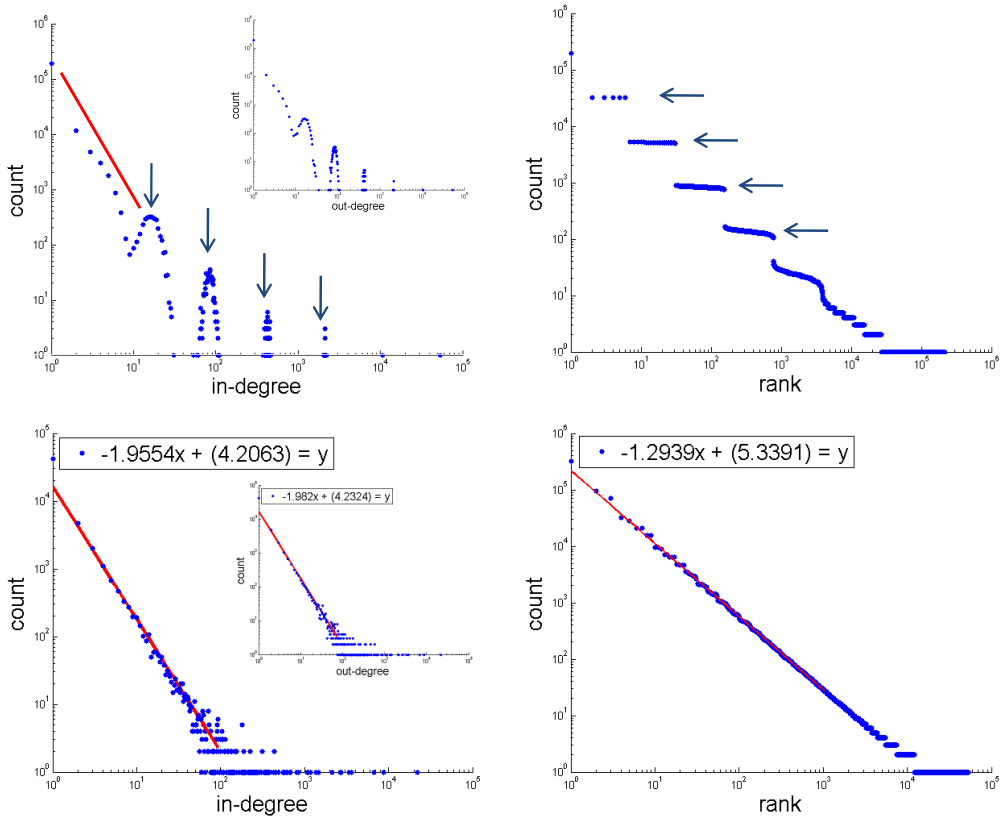
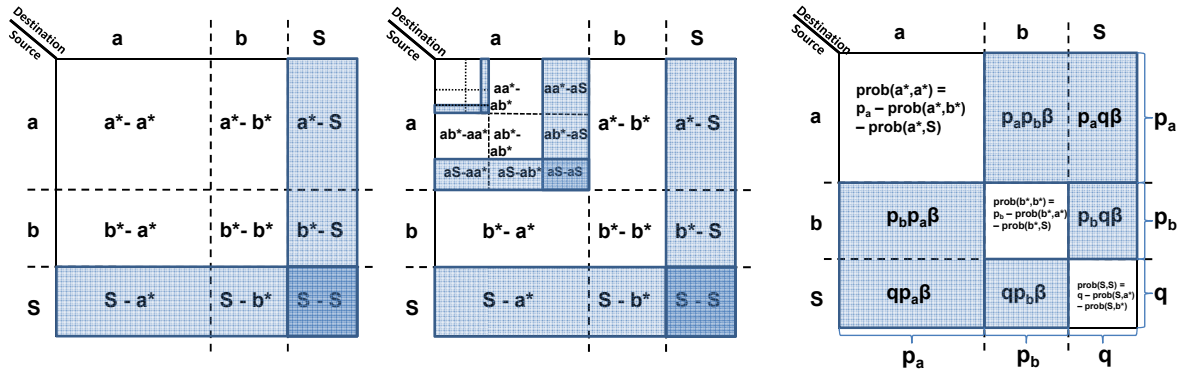


Figure 6.6: Top row: Results of RTG-IE ($k = 5$, $p = 0.16$, $W = 1M$). The problem with this model is that in(out)-degrees form multinomial clusters (left). This is because nodes with labels of the same length are expected to have the same degree. This can be observed on the rank-frequency plot (right) where we see many words with the same frequency. Notice the 'staircase effect'. Bottom row: Results of RTG-IU ($k = 5$, $p = [0.03, 0.05, 0.1, 0.22, 0.30]$, $W = 1M$). Unequal probabilities introduce smoothing on the frequency of words that are of the same length (right). As a result, degree distribution follows a power-law with expected heavy tails (left).

In order to model homophily and communities, rather than assigning cross-product probabilities to keys on the 2-d keyboard, we introduce an imbalance factor β, which will decrease the chance of a-to-b edges, and increase the chance for a-to-a and b-to-b edges, as shown in Figure 6.7 (c). Thus, for the example that we have, the formulas for the probabilities of the nine keys become:

$$prob(a,b) = prob(b,a) = p_a p_b \beta \text{ , } prob(a,a) = p_a - (prob(a,b) + prob(a,S)),$$
$$prob(S,a) = prob(a,S) = q p_a \beta \text{ , } prob(b,b) = p_b - (prob(b,a) + prob(b,S)),$$
$$prob(S,b) = prob(b,S) = q p_b \beta \text{ , } prob(S,S) = q - (prob(S,a) + prob(S,b)).$$

By boosting the probabilities of the diagonal keys and down-rating the probabilities of the off-diagonal keys, we are guaranteed that nodes with similar labels will have higher chance to get connected. The pseudo-code of the generator is given in Algorithm 1.



(a) first level (b) recursion (c) communities

Figure 6.7: The RTG model: random typing on a 2-d keyboard, generating edges (source-destination pairs). See Algorithm 1. (a) an example 2-d keyboard (nine keys), hitting a key generates the row(column) character for source(destination), shaded keys terminate source and/or destination words. (b) illustrates recursive nature. (c) the imbalance factor β favors diagonal keys and leads to homophily.

Algorithm 1: RTG Model

Input: number of keys k, probability q to hit 'space', #iterations W, imbalance factor β
Output: edge-list L for output graph $\mathcal{G}$

Initialize $(k+1)$-by-$(k+1)$ matrix P with cross-product probabilities
// in order to ensure homophily and community structure
Multiply off-diagonal probabilities by β, $0 < \beta < 1$
Boost diagonal probabilities s.t. sum of row (column) probabilities remain the same.
Initialize edge list L
foreach *1 to* W **do**
    L1, L2 $\leftarrow$ `SelectNodeLabels` (P, k)
    Append L1, L2 to L

Next,we describe how we handle time so that edge/weight additions are bursty and self-similar. We also discuss the generalizations of the model in order to produce all types of uni/bipartite, (un)weighted, and (un)directed graphs.

Burstiness and Self-similarity

Most real-world traffic as well as edge/weight additions to real-world graphs have been found to be self-similar and bursty [Crovella and Bestavros, 1996; Gomez and Santonja, 1998; Gribble et al., 1998]. Therefore, in this section we give a brief overview of how to aggregate time so that edge and weight additions, that is ΔE and ΔW, are bursty and self-similar.

Notice that when we link two nodes at each step, we add 1 to the total weight W. So, if every

Procedure SelectNodeLabels(P, k)

Input: Probability matrix P, number of keys k
Output: Source label L1 and destination label L2

```
Initialize L1 and L2 to empty string
while not terminated L1 and not terminated L2 do
    Draw key (i, j) with probability P(i, j)
    if i ≤ k, j ≤ k then
        Append character 'i' to L1 and 'j' to L2 if not terminated
    else if i ≤ k, j=k + 1 then
        Append character 'i' to L1 if not terminated
        Terminate L2
    else if i=k + 1, j ≤ k then
        Append character 'j' to L2 if not terminated
        Terminate L1
    else
        Terminate L1 and L2
return L1 and L2
```

step is represented as a single time-tick, the weight additions are uniform. However, to generate bursty traffic, we need to have a *bias factor b*> 0.5, such that b-fraction of the additions happen in one half and the remaining in the other half. We use the b-model [Wang et al., 2002], which generates such self-similar and bursty traffic. Specifically, starting with a uniform interval, we recursively subdivide weight additions to each half, quarter, and so on, according to the bias b. To create randomness, at each step we randomly swap the order of fractions b and $(1 - b)$.

Generalizations

We can easily generalize RTG to model all type of graphs. To generate undirected graphs, we can simply assume edges from source to destination to be undirected as the formation of source and destination labels is the same and symmetric. For unweighted graphs, we can simply ignore *duplicate* edges, that is, edges that connect already linked nodes. Finally, for bipartite graphs, we can use *two* different sets of keys such that on the 2-d keyboard, source dimension contains keys from the first set, and the destination dimension from the other set. This assures source and destination labels to be completely different, as desired.

6.2.4 Model Validation and Analysis

The question we wish to answer here is how well RTG is able to model real-world graphs. The datasets we used are:

BlogNet: a social network of blogs based on citations (undirected, unipartite and unweighted with N=27,726; E=126,227; over 80 time ticks).

CampOrg: the U.S. electoral campaign donations network from organizations to candidates (directed, bipartite and weighted with N=23,191; E=877,721; and W=4,383,105,580 over 29 time ticks). Weights on edges indicate donated dollar amounts.

In order to evaluate community structure, we use the modularity measure in [Newman and Girvan, 2004]. Figure 13.5 (left) shows that modularity increases with smaller imbalance factor β. Without any imbalance, β=1, modularity is as low as 0.35, which indicates that no significant modularity exists. In Figure 13.5 (right), we also show the running time of RTG with respect to the number of duplicate edges (that is, number of iterations W). Notice the *linear* growth with increasing W.

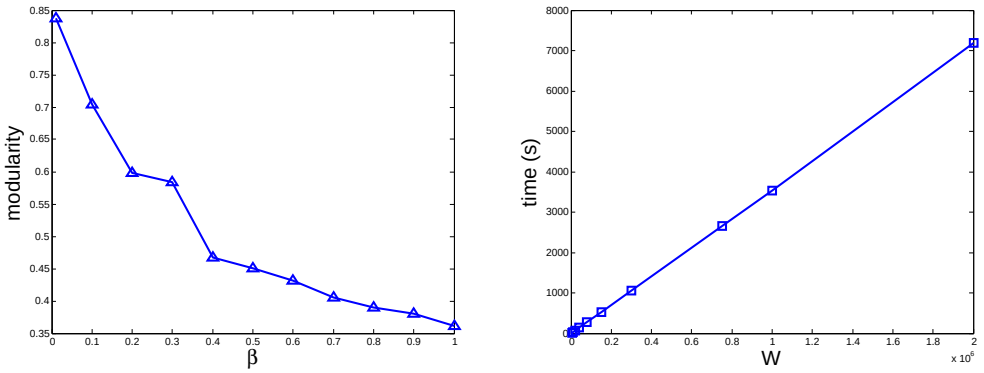
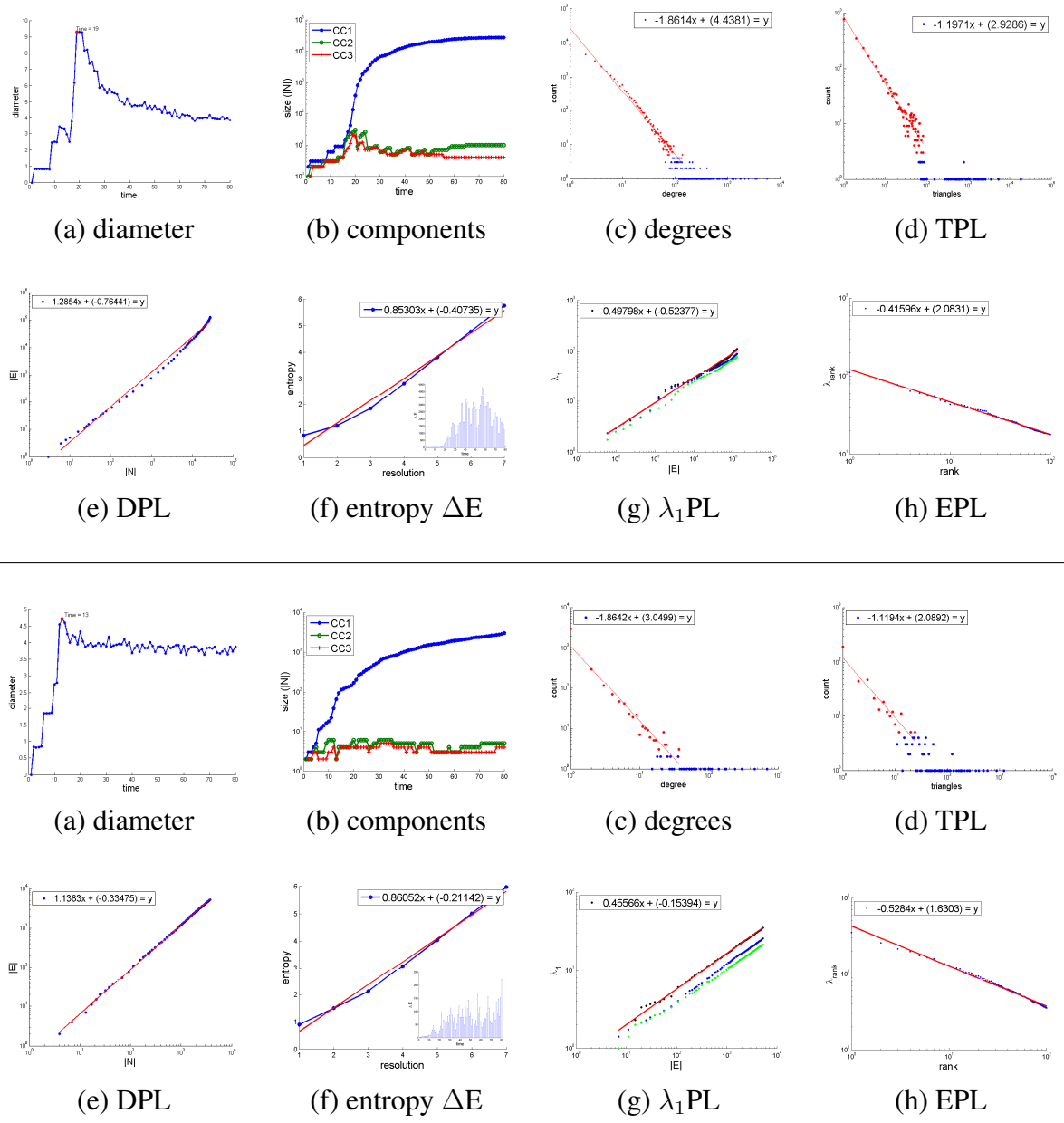


Figure 6.8: (left) Modularity score vs. imbalance factor β, modularity increases with decreasing β. For β=1, the score is very low indicating no significant modularity. (right) Computation time vs. number of iterations W, time grows *linearly* with W.

In Figures 6.9 and 6.10, we show the related patterns for *BlogNet* and *CampOrg* as well as synthetic results, respectively. In order to model these networks, we ran experiments for different parameter values k, q, W, and β. Here, we show the closest results that RTG generated, though fitting the parameters is a challenging future direction. We observe that RTG is able to match the *long* wish-list of static and dynamic properties we presented earlier for the two real graphs.



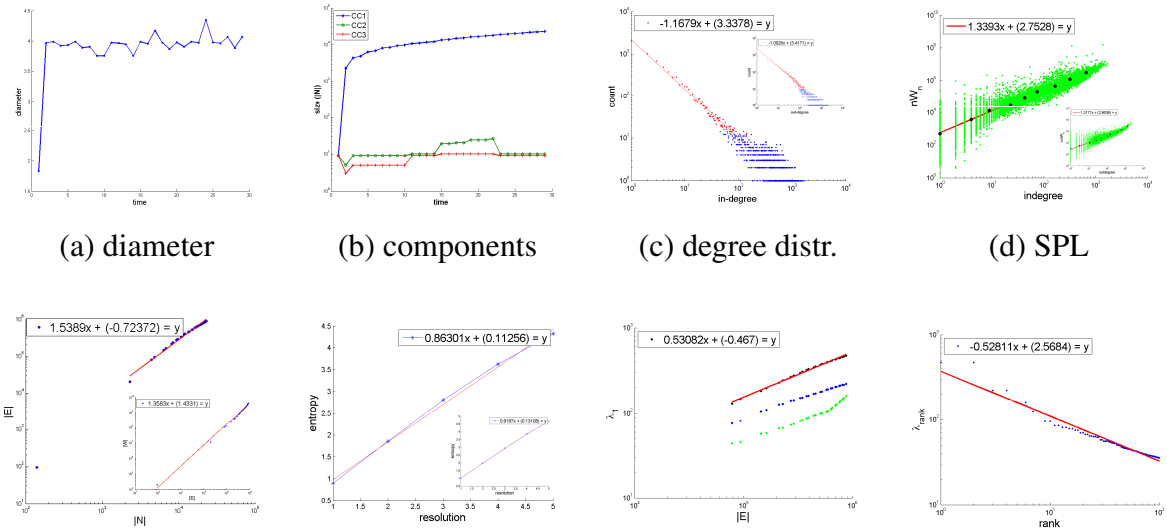
(a) diameter (b) components (c) degrees (d) TPL

(e) DPL (f) entropy ΔE (g) λ_1PL (h) EPL

(a) diameter (b) components (c) degrees (d) TPL

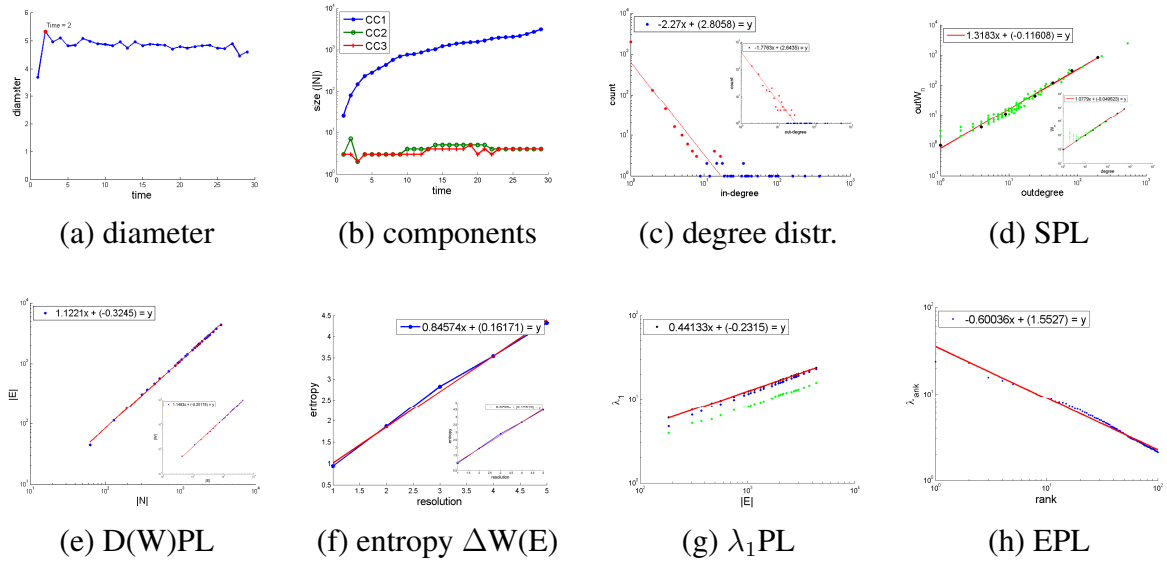
(e) DPL (f) entropy ΔE (g) λ_1PL (h) EPL

Figure 6.9: Top two rows: properties of *BlogNet*: (a) small and shrinking diameter; (b) largest 3 connected components; (c) degree distribution; (d) triangles Δ vs number of nodes with Δ triangles; (e) densification; (f) bursty edge additions; (g) largest 3 eigenvalues wrt E; (h) rank spectrum of the adjacency matrix. Bottom two rows: results of RTG. Notice the similar qualitative behavior for all *eight* laws. See Table 2.2 and Chapter 3 for details on these laws.



(a) diameter (b) components (c) degree distr. (d) SPL

(e) D(W)PL (f) entropy ΔW(E) (g) λ_1PL (h) EPL



(a) diameter (b) components (c) degree distr. (d) SPL

(e) D(W)PL (f) entropy ΔW(E) (g) λ_1PL (h) EPL

Figure 6.10: Top two rows: properties of *CampOrg*; as opposed to *BlogNet*, *CampOrg* is *weighted*. So, different from above we show: (d) node weight vs in(inset: out)degree; (e) total weight vs number of edges(inset); (f) bursty weight additions(inset); Bottom two rows: results of RTG. Notice the similar qualitative behavior for all *nine* laws. See Table 2.2 and Chapter 3 for details on these laws.

6.3 Summary of contributions

In this chapter, we focused on the problem of how to build a generative model that could produce realistic-looking synthetic networks. We presented two graph generators, namely RTM and RTG[1], that mimic topological properties of real-world networks. Compared to previous work on graph generators, these models are two of the first models that capture the dynamic and weighted properties, in addition to static unweighted properties, that real networks exhibit.

Our first model *Recursive Tensor Model* (RTM) is a simple, recursive generator based on Kronecker tensor multiplication. We rigorously proved that RTM produces several desired characteristics, such as bursty weight additions. We also experimentally validated that it mimics a *long* list of the laws described in previous chapters. On the other hand, RTM has some shortcomings. In particular, it requires a fixed, predetermined number of nodes as initial input. The number of nodes in the final output graph grows as powers of this initial number over iterations due to the nature of the Kronecker product. In other words, new nodes joining the network do not emerge naturally. A second issue with RTM is that it generates multinomial/lognormal (instead of power-law) distributions.

Our second model *Random Typing Graphs* (RTG) is based on a simple 'random typing' procedure which is shown to mimic natural behavior very well. The random typing involves hitting keys on a keyboard randomly and generating words, i.e. node labels, separated with space characters. This model overcomes the issues associated with RTM as discussed above. In particular, setting the probabilities of keys being hit unequally ensures that it generates power-laws. In addition, the nodes arrive to the network naturally over time– longer typing generates new words seamlessly. More importantly, it meets all the five desirable properties given in the introduction. Particularly, the RTG model is

1. realistic; generating graphs that obey all *eleven* properties that real graphs obey –no other generator has been shown to achieve that,
2. simple and intuitive; yet it generates the emergent, macroscopic patterns that we see in real-world graphs,
3. parsimonious; requiring only a handful of parameters,
4. flexible; capable of generating weighted/unweighted, directed/undirected, and unipartite/ bipartite graphs, and any combination of the above, and
5. fast; being linear on the number of iterations (on par with the number of duplicate edges in the output graph).

[1]Source code of our algorithm can be found at `www.cs.cmu.edu/~lakoglu/#code`

Chapter 7

Models of human communications

PROBLEM STATEMENT: *How could we design a realistic and intuitive generative model that will naturally reproduce real human-to-human communication behavior?*

In this chapter, we introduce a novel generator to model the way which humans decide when and whom to contact. We argue that such a model should be *utility driven*, as opposed to earlier models (preferential attachment [Barabási and Albert, 1999], forest-fire [Leskovec et al., 2005b], butterfly [Mcglohon et al., 2008], etc.) which are mainly randomness-guided generators. Our guiding principle is that humans balance a trade-off between the cost of the communication (in time and money), and its benefit (in valuable information and emotional support).

7.1 PaC: Pay and Call Model

Every communication, such as phone-call, SMS, and e-mail, has a cost in terms of money, time, and equipment. On the other hand, it has a benefit, otherwise humans would not do it. The benefits can be psychological and emotional (talking to friends makes us happy), or monetary (stock tip), or desirable in other ways. For ease of presentation, we refer to the benefit as if it is measured by *emotional dollars*. The point of this thought experiment is to set up a utility-driven model for the social contacts of humans, which should be more realistic and more informative than the ones using randomness.

Therefore, we assume that people are rational agents, which means telemarketers are excluded from the model, and we design our generator to guide the behavior of each agent according to a well-defined utility function. Ideally, the fundamental macro-phenomena of a social network should then emerge from the simple local behavior of each agent/human.

7.1.1 Model Description

Following the above discussion, we now present our utility-driven model PaC as a ***P**ay **a**nd **C**all* game. Assume a setting where a set $\mathcal{A}$ of n distinct agents create links to one another through phone calls. In every round of the game, each agent's strategy is to choose among other peers to whom he will make calls and build links. Links are undirected. Once agent $a_i \in \mathcal{A}$ calls $a_j \in \mathcal{A}$, there will be a link between them. The total number of phone-calls that a_i and a_j give to each other is treated as the *weight* on the undirected link between them. The PaC model essentially includes the following four ingredients:

- It adopts the agent-based modeling approach. Each agent has a *friendliness* value, an *exponential lifetime*, a certain amount of *capital*, and the *expected* payoffs from talking to strangers.
- The goal of each agent is to invest his limited capital into phone-calls and maximize the potential payoffs from each conversation.
- The per-minute gain of a conversation will be gradually *saturated*, and finally both of the callers and callees will lose interest, and stop the conversation.
- Each agent a_i can ask his partners for recommendations. Every partner recommends the profitable agents from his own partners, so a_i benefits from talking to the most profitable agent within the recommendations.

Friendliness and Exponential Lifetime

Each agent has a *friendliness* value $F_i \in (0, 1)$ to show his personality. F_i approaching to 1 means the agent is very open and friendly, and F_i close to 0 means he is very shy and introverted. a_i has a probability P_l, uniformly chosen from 0 to 1, to stay in the game, and has the probability $1 - P_l$ to leave the game, so that we can simulate the mixture of different ages in a real scenario. Once an old agent leaves, all his links will be removed, and a new agent replaces his position with the *friendliness* and P_l initialized to new values.

Utility-Driven Phone-Calls and Saturation

An agent's payoffs are the difference between the benefits and costs. The benefits are defined based on the following considerations. Two open agents usually can benefit emotionally from a happy conversation. When an open agent meets a shy agent, they may benefit less from their conversation. Finally, two shy agents might gain little in the end. In addition, after two agents have been talking for a while, they may gradually lose interest, and gain less emotionally as time goes by. For agent a_i and a_j, they can achieve $\sqrt{F_i \times F_j} \times \alpha^{m-1}$ emotional dollars per minute from a conversation, where $\alpha \in (0, 1)$ is called the *saturation* factor to represent the loss of interest, and m is the number of minutes for which they have been talking.

For an m-minutes long conversation, the total benefits are defined as

$$
\begin{aligned}
benefits &= \sqrt{F_i \times F_j} \times \sum (1 + \alpha + \alpha^2 + ... + \alpha^{m-1}) \\
&= \sqrt{F_i \times F_j} \times \frac{1 - \alpha^m}{1 - \alpha}
\end{aligned} \tag{7.1}
$$

The costs are the expenses of phone-calls, which include C_{ini} and C_{pm}. C_{ini} is the cost to initiate a phone-call, and C_{pm} is the per-minute fee. The total costs for an m-minutes call will be $C_{ini} + m \times C_{pm}$, so our utility function is defined as

$$
payoffs = benefits - C_{ini} - m \times C_{pm} \tag{7.2}
$$

and each agent starts and maintains a conversation until the payoffs as given by Equation 7.2 reach the maximum value or the agent has used all his money.

Expected Payoffs on Strangers

At first, each agent is given an initial *capital* which is enough to make one call only. Since none of the agents have ever talked before, agent a_i first uniformly calls a stranger a_j, and keeps the conversation until either the payoffs by Equation 7.2 begin to decrease or s/he spends all his money in the call ($a_i.capital \leq 0$). When the call is finished, a_i and a_j will achieve the payoffs P_j from the conversation. A link is built between a_i and a_j with weight 1, and a_i will remember the payoffs P_j earned by talking to a_j. Because a_j was first a stranger to a_i before they met, a_i also updates his *expected* payoffs from talking to strangers as

$$
S_{exp} = \frac{\sum P_j}{1 + S} \tag{7.3}
$$

where S is the total number of times talking to strangers, and P_i is the payoffs achieved at each time. S_{exp} is initialized to 0 in the beginning. In each round of the game, agent a_i is only allowed to call a_j for one time. If a_i still has some money left (note that the payoffs earned in the current round can only be used in the next round), he will continue to interact with other strangers.

Recommendations

Once agent a_i has some partners, he will first prioritize his partners according to the remembered payoffs, and talk to them respectively. If the payoffs of the currently chosen partner is less than a_i's expected payoffs from strangers ($S_{exp} \geq 0$), a_i will stop talking to partners and choose to call strangers again. He first asks his partners for recommendations. Every partner will tell a_i how much money he actually earned by talking to his own partners last time. a_i can then pick the most profitable agent out of the partners of his partners. If all the recommended agents are already his partners, a_i will uniformly choose a stranger from the rest.

In summary, the PaC model is formulated as pseudo-code in Algorithm 2.

Algorithm 2: PaC Model

Input: C_{ini}, C_{pm}, α
foreach $a_i \in \mathcal{A}$ **do**
  if a_i *stays with probability* $a_i.P_l$ *and* $a_i.capital \geq C_{ini} + C_{pm}$ **then**
    if $N(a_i) = \emptyset$ **then**
      `Talk2Strangers(`a_i`)`
    else
      `Talk2Partners(`a_i`)`
  if a_i *quits with probability* $1 - a_i.P_l$ **then**
    Replace a_i with a newly born agent

Procedure Talk2Strangers(a_i)

Input: current agent a_i
$total \leftarrow 0$
if $N(a_i) \neq \emptyset$ *and* a_i *finds the most profitable agent he never talked to from the recommendations* **then**
  $a_j \leftarrow$ the most profitable agent
else
  $a_j \leftarrow$`GetRandom(`$\mathcal{A}$`)`
while $a_i.capital \geq C_{ini} + C_{pm}$ *and* $a_i.S_{exp} \geq 0$ **do**
  maximize $payoffs$ with constraint $a_i.capital \geq 0$ by Equation 7.2
  add a_j to $N(a_i)$, add a_i to $N(a_j)$
  $a_j.capital \leftarrow a_j.capital + payoffs$
  $total \leftarrow total + payoffs$
  update $a_i.S_{exp}$ and $a_j.S_{exp}$ by Equation 7.3
$a_i.capital \leftarrow a_i.capital + total$

Procedure Talk2Partners(a_i)

Input: current agent a_i
$total \leftarrow 0$
prioritize $N(a_i)$ according to the descending order of the remembered payoffs
for $k \leftarrow 1$ **to** $|N(a_i)|$ **do**
  $P_k \leftarrow a_i$'s remembered payoffs from a_k
  if $P_k \geq a_i.S_{exp}$ **then**
    maximize $payoffs$ with constraint $a_i.capital \geq 0$ by Equation 7.2
    increase the weight on the link between a_i and a_k by 1
    $a_j.capital \leftarrow a_j.capital + payoffs$
    $total \leftarrow total + payoffs$
  else
    `Talk2Strangers(`a_i`)`
    break
  if $a_i.capital \leq 0$ **then** break
$a_i.capital \leftarrow a_i.capital + total$

7.1.2 Model Validation and Analysis

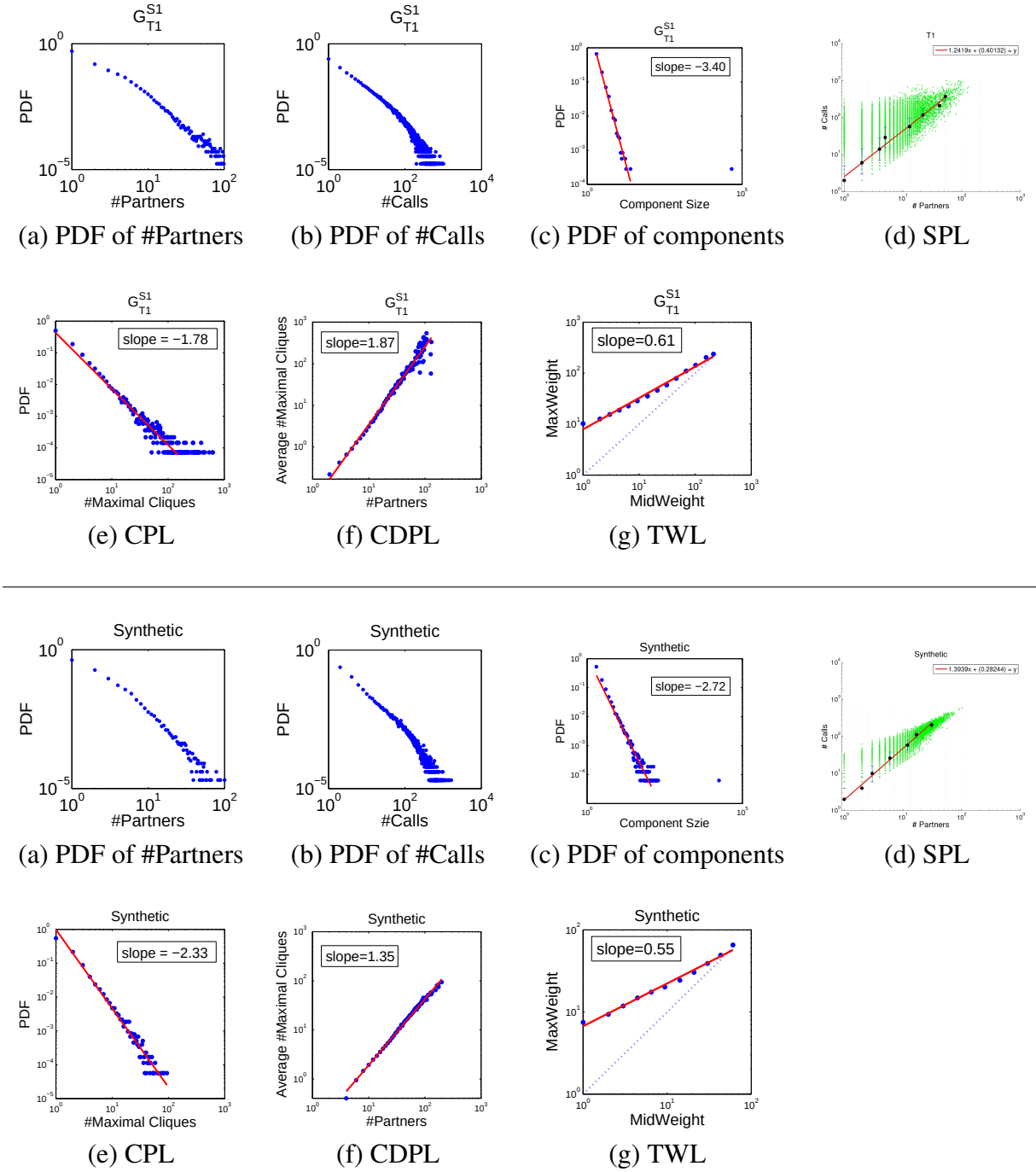
Our goal here is to show that our model is able to generate degree, weight and clique distributions that mimic a real graph like our communication networks. Notice that we only want to show qualitative match of the properties. Exact fitting is outside the scope of this work. We decided to test our model with respect to all the usual patterns, and specifically the degree distribution, weight distribution, as well as the snapshot power law. We also want to qualitatively check against our new clique-related patterns, the CDPL, CPL, and the TWL. We simulated the model 35 times for 100,000 nodes, with $C_{ini} = 0.1$, $C_{pm} = 0.4$ and $\alpha = 0.9$. For each agent a_i, F_i and P_l are randomly chosen from 0 to 1.

Figure 7.1 shows the results of these checkpoints. The top rows are for the actual graph $\mathcal{G}_{T1}^{S1}$, and the bottom rows are for a synthetic graph, generated by our PaC model. In all cases, notice that PaC gives skewed distributions that are remarkably close to the real ones. For example, except for the giant connected component which is an isolated point distant from the rest, the size distribution of the connected-components conforms to a power-law. The exponents take values within the range observed in real world networks with a least-square fit of $R^2 > 0.95$.

From earlier research [Mitzenmacher, 2003; Reed and Jorgensen, 2004], we understand how heavy-tailed distributions such as power-law, lognormal and DPLN could arise for the degree distribution and the node weight distribution. According to [Mitzenmacher, 2003], lognormal distributions can be naturally generated by *multiplicative processes*: For a biological example, at each step j, an organism may grow or shrink by a certain percentage according to a random variable F_j. If X_j denotes the current size of the organism, $X_j = F_j X_{j-1}$ where F_j is independent of X_{j-1}. Consider $\ln X_j = \ln X_0 + \sum_{k=1}^{j} \ln F_k$. If $F_k, 1 \leq k \leq j$, are independent lognormal distributions, then X_j is always lognormal. If F_k are not lognormal, but are independent and identically distributed with finite mean and variance, by Central Limit Theorem, $\sum_{k=1}^{j} \ln F_k$ converges to a normal distribution, and X_j will asymptotically approach a lognormal distribution. If X_j is lower bounded by a minimum value, then the distribution will become a power-law. If we sample the series from X_0 to X_j by a geometrically distributed random time k, we will have a geometric mixture of lognormal distributions. This will turn out to be a DPLN distribution with two power-laws at both tails.

Following the *PowerTrack* method in [Reed and Jorgensen, 2004], we empirically analyze the generative process of our PaC model by taking two snapshots $\mathcal{G}_{T_i}$ and $\mathcal{G}_{T_j}$ at time step T_i and T_j with $j - i \geq 1$. Among the common agents between $\mathcal{G}_{T_i}$ and $\mathcal{G}_{T_j}$, we calculate the ratio X_{T_j}/X_{T_i}, where X_t represents either the degree or the weight for each node. In Figure 7.2, the distributions of the ratio for both of the degree and weight appear to be parabolic in logarithmic scales. This provides good evidence that a lognormal multiplicative process similar to that described in [Mitzenmacher, 2003] is involved in the temporal evolution of our model.

Another important issue is that we also need to test the independence between partners and their ratios, and the same for the calls. Here, the correlation coefficients, which are necessary but not sufficient for independence [Reed and Jorgensen, 2004], are very small: -0.02 and -0.04 for partners and calls respectively. Finally, in each round of the game, every agent has the proba-



(a) PDF of #Partners (b) PDF of #Calls (c) PDF of components (d) SPL

(e) CPL (f) CDPL (g) TWL

(a) PDF of #Partners (b) PDF of #Calls (c) PDF of components (d) SPL

(e) CPL (f) CDPL (g) TWL

Figure 7.1: Qualitative comparison between the real graph (top two rows) and our synthetic graph (bottom two rows). PaC gives skewed distributions like the real ones.

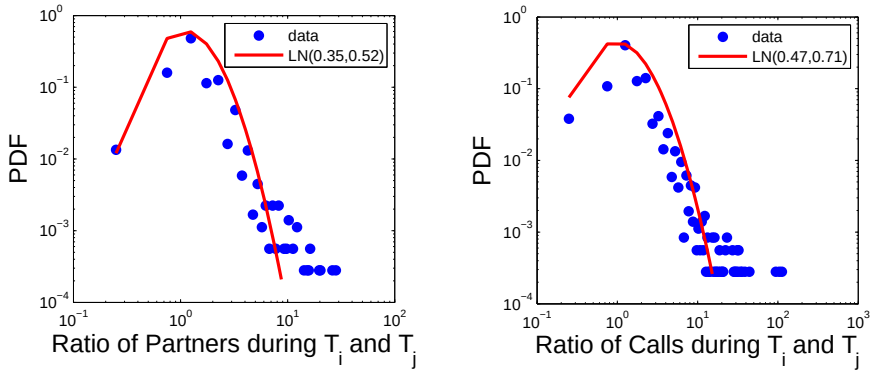


Figure 7.2: The ratio of partners (left), and calls (right) between two different snapshots of PaC follow the lognormal distribution. The parabolic line is fitted in red.

bility P_l to stay or leave the game, which essentially leads to a geometric lifetime. Therefore, although we do not explicitly assume any prior distribution about the ratio (the File Model in [Mitzenmacher, 2003] explicitly assumes a lognormal distribution for F_k), the PaC model can still mimic the DPLN degree distribution and the node weight distribution which are identical with the real communication networks.

By comparing with the existing graph generators, we see that many models like the preferential-attachment guided models usually ignore the weight information, and only generate the giant connected component. In contrast, our model is able to reproduce the networks that have not only the patterns holding in un-weighted networks, but also the patterns followed by the weighted communication networks.

7.2 Summary of contributions

Many preferential-attachment [Barabási and Albert, 1999] guided models assume that a newly-added node is more likely to be linked to the most popular node of the current graph. However, in real world scenarios, incoming nodes are typically unaware of such global structural knowledge of the network. Moreover, most earlier generators dictate that nodes will choose contacts at random; in contrast, we argue that people choose contacts to maximize some utility.

In this work we designed an intuitive graph generator, called PaC, for modeling human communication behavior. In this model, each node (a) uses only local information, and (b) uses no randomness, but instead tries to maximize a well-defined utility function. The major advantage of PaC over older generators is that it can answer *what if* scenarios. For example, if the connection price of each phone-call goes up, will this decrease the average number of friendship edges, what about a change in the price-per-minute, what if there is a flat rate, and so on. Based on the utility function of PaC, we can explore what the impact of these settings would be on the structure and evolution of the network.

Part III

Anomaly Detection

Chapter 8

Preliminaries

8.1 Introduction

Given a large real-world graph, possibly with weighted edges and attributed nodes, which nodes should we consider as "strange"? What are the time points at which the graph structure changes significantly? Applications of such settings abound: For example, in network intrusion detection, we have computers sending packets to each other, and we want to know which nodes misbehave (e.g., spammers, port-scanners). In a who-calls-whom network [Seshadri et al., 2008], strange behavior may indicate defecting customers, or telemarketers, or even faulty equipment dropping connections too often. In a social network, like Facebook and LinkedIn, again we want to spot users whose behavior deviates from the usual behavior [Leskovec et al., 2008], such as people adding friends indiscriminately, in "popularity contests".

The list of applications continues: Anomalous behavior could signify irregularities, like credit card fraud, calling card fraud, campaign donation irregularities, accounting inefficiencies or fraud [Bay et al., 2006], extremely cross-disciplinary authors in an author-paper graph [Sun et al., 2005], suspicious cargo shipments [Eberle and Holder, 2006], electronic auction fraud [Chau et al., 2006; Pandit et al., 2007], and many others. In addition to revealing suspicious, illegal and/or dangerous behavior, anomaly detection is useful for spotting rare events, as well as for the thankless, but absolutely vital task of data cleansing [Chen et al., 2005; Dasu and Johnson, 2003]. Moreover, anomaly detection is intimately related with the pattern and law discovery: unless the majority of our nodes closely obey a pattern (say, a power law), only then can we confidently consider as outliers the few nodes that deviate [Broder et al., 2000].

Most existing anomaly and outlier detection algorithms focus on clouds of multi-dimensional points, and similarly most algorithms for change and event detection are designed for traditional time series data. Our main contribution in this part of the thesis is to develop such methods for *graph* data. We tackle the above problem in various settings; for graphs possibly with weighted edges, attributed nodes, and for graphs changing over time. Furthermore, we make our detection techniques accessible to human analysts, by providing means for sensemaking.

In Chapter 9, we focus on the *problem* of spotting strange nodes in *plain* graphs, with weighted edges. We refer to a graph for which we only know its structure, that is the nodes and the links among them, as a plain graph. We start with answering questions such as: what features should we extract to characterize each node?, what patterns and laws do the nodes obey? Based on those patterns we observe, we develop ODDBALL, a scalable, un-supervised method for anomalous node detection. The *main idea* is to focus on the local neighborhood (the *egonet*), that is, a sphere, or a ball (hence the name ODDBALL) around each node. We show that egonets obey some surprising patterns, which gives us confidence to declare as outliers the ones that deviate. We apply ODDBALL[1] to numerous real graphs (*DBLP*, political donations, etc.) and we show that it indeed spots nodes that a human would agree are strange and/or extreme.

In Chapter 10 and 11, we address the *problem* of finding clusters and outliers in *attributed* graphs. We refer to a graph in which the nodes are associated with a set of attributes (or features) as an attributed graph. We consider both binary and categorical attributes in those two chapters, respectively. First we introduce PICS [1], to find cohesive groups of nodes which have similar connectivity patterns as well as attribute coherence. Second we introduce COMPREX, for identifying anomalies using pattern-based compression. The *main idea* behind these work is to build a compression model that describes the *norm* of the data succinctly, and subsequently flag those points that are dissimilar to the norm—those with high compression cost—as anomalies.

In Chapter 12, we look at the *problem* of detecting change-points in a time-varying graph at which many nodes deviate from their normal 'behavior'. In a nut-shell, the *main idea* of our method is as follows. We first extract time sequences of several network features for all nodes in the graph. Then, we derive a 'behavior' vector for the nodes over consecutive time windows and compare each one to a summary of previous 'behavior' vectors. We flag a time window as anomalous and report that an *event* has occurred if its 'behavior' is found to be significantly different than its past.

In Chapter 13, we develop a novel method for *sensemaking* for graph anomalies. The methods like ODDBALL, PICS, and COMPREX we introduce in earlier chapters help us find anomalous nodes in a given graph. Our new method, called DOT2DOT [1], helps us to understand how these nodes are correlated within the graph and thus to further summarize the anomalies. In particular, it groups the flagged anomalous nodes into groups where the nodes in the same group are 'close' to each other in the graph while nodes across groups are 'sufficiently' far apart. For the 'close-by' nodes in the same group, DOT2DOT finds a small candidate graph around them for visualization and it highlights good connection pathways among them. The *main idea* is to build a compression framework and employ information-theoretic ideas to select the best set of connection subgraphs that would yield the minimum compression cost for the given subset of nodes in the graph.

All the proposed methods are designed to work in an *unsupervised* fashion. That is, they do not assume that there exist any labeled anomalous data to learn from. Furthermore, the algorithms are developed so that their running time scale *linear*ly with respect to the input graph size, as well as they operate *parameter-free*, i.e. require no user-specified parameters (similarity thresholds, number of clusters, etc.).

[1]Source code of ODDBALL, PICS, and DOT2DOT are at `www.cs.cmu.edu/~lakoglu/#code`

8.2 Definitions

In this section, we introduce basic terminology and present the Minimum Description Length (MDL) principle. MDL serves as the main model selection principle we use for our methods in this part of the thesis.

Egonets of a Graph

Borrowing terminology from social network analysis (SNA), an "ego" is an individual node. A graph has as many egos as it has nodes. Egos can be people, groups, organizations, or whole societies, depending on what the nodes of the graph represent.

The "k-step neighborhood" of node i is the collection of node i, all its k-step-away nodes, and all the connections among all of these nodes – formally, this is the "*induced subgraph*". In SNA, the 1-step neighborhood of a node is specifically known as its "*egonet*". It contains the "ego" (center of interest) and all the nodes (neighbors) and edges directly around the ego. Formally,

Definition 5 (*k*-step neighborhood) *Given a graph $\mathcal{G}$, node i in $\mathcal{G}$, and a non-negative integer k, let $\mathcal{N}_{(i,k)} = \{j \in V(\mathcal{G}) : d(i,j) \leq k\}$ denote the k-step away neighbors of node i, where $d(i,j)$ is defined as the minimum path length from node i to node j in $\mathcal{G}$. Then, the* k*-step neighborhood of node i, denoted as $\mathcal{G}_{(i,k)}(V,E)$, is defined as the induced subgraph, where $V(\mathcal{G}_{(i,k)}) = \mathcal{N}_{(i,k)}$ and $E(\mathcal{G}_{(i,k)}) = \{(i,j) \in E(\mathcal{G}) : i,j \in \mathcal{N}_{(i,k)}\}$.*

Minimum Description Length principle

The Minimum Description Length (MDL) principle [Grünwald, 2007] is a practical version of Kolmogorov Complexity [Li and Vitányi, 1993], and can be regarded as a model selection criterion based on lossless compression principles. Both embrace the slogan *Induction by Compression*. For MDL, this can be roughly described as follows:

Given a set of models, or hypotheses, $\mathcal{H}$, the best model $H \in \mathcal{H}$ is the one that minimizes

$$L(H) + L(D \mid H) \ ,$$

in which $L(H)$ is the length in bits of the description of H, and $L(D \mid H)$ is the length of the data when encoded with model H. That is, the MDL-optimal model for a database D encodes D most succinctly among all possible models; it provides the best possible lossless compression.

MDL provides us a systematic approach for selecting the model that best balances the complexity of the model and its fit to the data. While models that are overly complex may provide an arbitrarily good fit to the data and thus have low $L(D \mid H)$, they overfit the data, and are penalized with a relatively high $L(H)$. Overly simple models, on the other hand, have very low $L(H)$, but as they fail to identify important structure in D, their corresponding $L(D \mid H)$ tends to be relatively high. As such, the MDL-optimal model provides the best balance between model complexity and goodness of fit.

The above definition is called two-part MDL, or *crude* MDL—as opposed to *refined* MDL, where model and data are encoded together [Grünwald, 2007]. We use two-part MDL because we are specifically interested in the model. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases. Note that MDL requires the compression to be *lossless* in order to allow for fair comparison between different $H \in \mathcal{H}$.

To use MDL, we have to define what our models $\mathcal{H}$ are, how a $H \in \mathcal{H}$ describes the data at hand, and how we encode this all in bits. For example in the following chapters, we will use block-encoding or code-table encoding as our models. Note that in MDL we are only concerned with code lengths, not actual code words.

8.3 Related Work

Related work of this part include outlier detection in clouds of multi-dimensional data points, anomaly detection in graph data, event detection in time-series data, data summarization, graph clustering, and connection subgraphs, which we discuss in the given order next.

Outlier Detection in Multi-dimensional Data

Outlier detection has attracted wide interest, being a difficult problem, despite its apparent simplicity. Even the definition of the outlier is hard to give: For instance, [Hawkins, 1980] defines an outlier as "an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism." Similar, but not identical, definitions have been given by others: [Barnett and Lewis, 1994], states that an outlier is one that appears to deviate markedly from the other members of the sample in which it occurs. [Johnson and Wichern, 1998]. defines an outlier to be an observation in a data set that appears to be inconsistent with the remainder of that set of data. These definitions are formalized by [Knorr and Ng, 1998]:

Definition 6 (DB(pct,d_{min})) *An object p in a data set D is a $DB(pct, dmin)$-outlier if at least percentage pct of the objects in D lies greater than distance d_{min} from p. That is, the cardinality of the set $\{q \in D | d(p,q) > d_{min}\}$ is greater than pct% of the size of D, where $d(p,q)$ denotes the distance between object q and object p in D.*

Outlier detection methods form two classes, *parametric* and *non-parametric*. The former class includes statistical methods that assume there exists a standard underlying distribution of the observations that fit the data [Barnett and Lewis, 1994; Hawkins, 1980]. and those observations that deviate from the model assumptions are flagged to be outliers. These methods exhibit problems for high dimensional data sets and for settings where the underlying data distribution is unknown [Papadimitriou et al., 2003]. The latter class includes *distance-based* and *density-based* data mining methods. These methods typically define as an outlier the (n-D) point that is too far away from the rest, and thus lives in a low-density area [Knorr and Ng, 1998]. Typical methods include LOF [Breunig et al., 2000b], LOCI [Papadimitriou et al., 2003], and recently

LSOD [Wang et al., 2011]. These methods not only flag a point as an outlier but they also give outlierness scores; thus, they can sort the points according to their "strangeness".

Many other *density-based* methods that perform well in detecting outliers in very large datasets of high dimension are proposed in [Aggarwal and Yu, 2001; Arning et al., 1996; Chaudhary et al., 2002; Guha et al., 2001; Kaufman and Rousseeuw, 1990; Ng and Han, 1994; Zhang et al., 1996]. Feature bagging [Lazarevic and Kumar, 2005] also proves to be useful to tackle high dimensionality, where features are randomly grouped into multiple sets of different sizes and outlier detection algorithms are performed on each different set after which the scores are combined. Finally, most clustering algorithms [Guha et al., 2001; Kaufman and Rousseeuw, 1990; Ng and Han, 1994; Zhang et al., 1996] reveal outliers as a by-product.

While most work on outlier detection has focused on numerical datasets, there also exist some work on unsupervised anomaly detection in categorical data. [Bronstein et al., 2001] learns a structure and the parameters of a Bayesian network, and use the log-likelihood values as the anomalousness score of each record. [Das and Schneider, 2007; Wong et al., 2003] address the problem of finding anomaly *pattern*s. They build a Bayes net that represents the baseline distribution, and then score all one and two component rules with unusual proportions compared to the baseline, with the aim of detecting rules that summarize significant patterns of anomalies. [Smets and Vreeken, 2011] takes a pattern-based compression approach and employs KRIMP [Siebes et al., 2006] as its compressor for anomaly detection.

For a comprehensive survey on outlier detection techniques, see [Chandola et al., 2009].

Anomaly Detection in Graph Data

Compared to outlier detection, anomaly detection in graph-based data has only recently gained attention. Current work in this area can be divided into two areas, discovering anomalies in static snapshots of graphs and in time series of graphs. Spotting anomalies in time-evolving graphs, which we include in the next sub-section, largely focuses on detecting anomalous points in time during the evolution of a graph by using different graph editing distances.

[Noble and Cook, 2003] detect anomalous sub-graphs using variants of the Minimum Description Length principle. [Eberle and Holder, 2007] also use the MDL principle as well as other probabilistic measures to detect several types of anomalies (e.g. unexpected/missing nodes/edges). [Liu et al., 2005] detect non-crashing bugs in software using frequent execution flow graphs combined with classification. They extend this work, using graph properties such as connectedness and clustering coefficient to show that normal graphs and intentionally altered graphs significantly differ in such properties [Eberle and Holder, 2006]. [Chakrabarti, 2004] uses MDL to spot outlier edges that lie across clusters. [Gao et al., 2010] spots anomalous nodes that do not well belong to the community they reside, which they call community outliers. [Sun et al., 2005] use proximity and random walks, to assess the normality of nodes in bipartite graphs. OutRank [Ghoting et al., 2004] and LOADED [Moonesinghe and Tan, 2008] use similarity graphs of objects to detect outliers. [Shetty and Adibi, 2005] use an entropy-based approach, flagging important nodes that increase the entropy of a graph when they are removed.

Event Detection in Time-series Data

Anomaly detection has been studied widely in many settings such as 'anomalous point detection' on clouds of multi-dimensional points, spatio-temporal 'anomalous pattern detection', and 'graph anomaly detection', with many motivating applications such as network intrusion detection [Sequeira and Zaki, 2002], detection of medical insurance claim fraud, credit card fraud, electronic auction fraud [Bolton and David, 2002; Chau et al., 2006], fault detection in engineering systems [Fujimaki et al., 2005] as well as many others.

Another class of anomaly detection methods can be grouped under 'change-point detection' on a sequence of time series of data, which address the problem of spotting points in time at which properties of the time-series data change significantly [Basseville and Nikiforov, 1993; Brodsky and Darkhovsky, 1993; Gustafsson, 2000; Kawahara et al., 2007; Kifer et al., 2004; Yamanishi and ichi Takeuchi, 2002]. This problem is also referred as event detection [Guralnik and Srivastava, 1999].

Although the change-point detection problem has been actively studied in the statistics and the data mining communities over the last several decades, there has been much less focus on change-point detection particularly in graph data. [Idé and Kashima, 2004] developed an eigen-vector based algorithm to detect faults in multi-tier Web-based systems represented as a time sequence of graphs. Another set of research [Bunke and Shearer, 1998; Shoubridge et al., 2002] derives distance functions between a pair of graphs, compute distances between consecutive graphs in a given sequence, and finally apply traditional anomaly detection methods on the time series of distance values. [Sun et al., 2007] propose a parameter-free algorithm to discover communities in streams of graph and flag points in time as discontinuity points when the community structure changes significantly.

Data Description and Summarization

Our COMPREX method describes data using sets of patterns, so research on pattern set mining is related. KRIMP algorithm by [Siebes et al., 2006; Vreeken et al., 2011] employs the MDL principle [Grünwald, 2007] to define the best set of patterns as those patterns that compress the data best. KRIMP heuristically approximates the optimal pattern set by considering a collection of itemsets in a fixed order, greedily selecting itemsets that contribute to better compression. The code tables it discovers have been shown to characterize the data distribution very well, providing highly competitive performance on a variety of data mining tasks [Smets and Vreeken, 2011; Vreeken et al., 2011].

The PACK algorithm [Tatti and Vreeken, 2008] also follows a bottom-up MDL approach, using a decision tree per attribute to encode binary data 0/1 symmetrically. By translating these trees into downward-closed families of itemsets, patterns can be extracted. Although good compression results are obtained, by the downward-closed requirement typically large groups of itemsets are returned. Further examples of pattern set mining methods that describe data include Tiling [Geerts et al., 2004], Noisy Tiling [Kontonasios and De Bie, 2010], and Boolean ma-

trix factorization [Miettinen et al., 2008]. As these do not define a probability (or code length) distribution for individual rows, it is not trivial to apply them for anomaly detection. Attribute clustering [Mampaey and Vreeken, 2010] provides a high-level summarization of the data, where the authors employ MDL to find the optimal grouping of binary attributes. These groups are described using crude code tables, consisting of all attribute-value combinations in the data.

Graph Clustering

Graph partitioning has been well studied in the literature. The top-performing methods include METIS [Karypis and Kumar, 1998] and spectral clustering [Ng et al., 2001]. However, these as well as many other graph partitioning methods [Andersen et al., 2006; Flake et al., 2000; Girvan and Newman, 2002] work with the connectivity structure of the graph and cannot be directly for attributed graphs. More importantly, they require the number of partitions and a measure of imbalance between any two partitions as input. These are usually hard for the user to specify, especially for large graphs, and require experimentation to get good results. For example, spectral partitioning requires the choice from several measures such as ratio cut [Chan et al., 1993], normalized cut [Shi and Malik, 2000], or min-max cut [Ding et al., 2001].

The idea of using (lossy) compression for graph clustering is introduced in [Dhillon et al., 2003]. The information-theoretic co-clustering algorithm simultaneously clusters rows and the columns of a normalized contingency table which is treated as a two-dimensional joint probability distribution. The algorithm, however, requires the number of row and column clusters as input.

In terms of parameter-free graph clustering algorithms, Autopart [Chakrabarti, 2004] and cross-associations [Chakrabarti et al., 2004a], and their extension to time-evolving [Sun et al., 2007] and k-partite graphs [He et al., 2009] are the most representative. These methods use the *minimum description length (MDL)* principle [Grünwald, 2007] to automatically choose the number of clusters. However, they do not apply to attributed graphs as they operate on the adjacency matrix of the graph, and consider only the connectivity structure of the graph.

Compared to the wide range of work on graph clustering, there has been much less focus on clustering attributed graphs. [Hanisch et al., 2002] transforms the graph and the attributes to a combined distance metric and then applies flat clustering. This and other similar methods [Tian et al., 2008] achieve homogeneity of attributes for the nodes in the same cluster, however they tend to yield low intra-cluster connectivity. [Zhou et al., 2009, 2010] transforms the attributes to additional nodes in the graph, where original nodes are linked to attribute nodes that they exhibit. Again, all these methods require the number of clusters to be exclusively specified.

Recently, [Günnemann et al., 2010; Moser et al., 2009] propose methods to extract cohesive subgraphs from an attributed graph rather than partitioning the entire graph. The subgraphs exhibit high density and homogeneity in a subset of their attributes. In these methods other types of parameters need to be set by the user; these include the subspace dimensionality and density thresholds, as well as the minimum number of nodes in each cluster.

The spectral relational clustering algorithm [Long et al., 2006] performs collective factorization

of related matrices for multi-type relational data clustering. Although the method is applicable to graph data with attributes, the user intervention there is two-fold; besides the number of clusters for each type of nodes, reasonable weights for different type of relations or attributes also need to be specified. In addition, it is not easy to do spectral clustering on directed graphs with attributes (there exists spectral clustering for plain directed graphs, although it is much more complicated than those for un-directed graphs).

Connection Subgraphs and Visualization

Our DOT2DOT method finds connection subgraphs among top anomalous nodes for better characterization and visualization. The connection subgraphs mining problem is introduced by [Faloutsos et al., 2004]. As they formulated, a connection subgraph is a relatively small, connected subgraph that well captures the relationship (set of interesting paths) between two nodes in a given graph. Later, [Ramakrishnan et al., 2005] and [Sevon and Eronen, 2008] extended [Faloutsos et al., 2004] for querying labeled graphs in which nodes and edges belong to certain types or relations. On similar lines, [Jin et al., 2007] and [Shahaf and Guestrin, 2010] propose techniques to discover a coherent chain of evidence trails that connect two given concepts in text documents. All these methods mentioned so far, however, are appropriate for only two query nodes; thus, they do not generalize for more than two input nodes.

The Center-Piece Subgraph Problem introduced by [Tong and Faloutsos, 2006; Tong et al., 2007] finds the most 'centered' node that has strong connections to most or all of the query nodes, and keeps adding important paths from the center-piece node to active nodes until an upper bound on the number of nodes in the output graph is reached. [Koren et al., 2006] introduce the Cycle-Free Effective Conductance measure for graph proximity and propose methods to extract the subgraph with at most a certain size which retains most of the proximity between query nodes. Proximity and connection subgraphs are also exploited for graph visualization and summarization [Chau et al., 2008, 2011; Jr. et al., 2006]. For a survey, see [Alsudairy et al., 2011].

Another set of related work focuses on expanding communities around a given set of seed nodes [Andersen and Lang, 2006; Andersen et al., 2006; Riedy et al., 2011; Spielman and Teng, 2008]. These methods find a subgraph containing the seed nodes and differ in the specific graph measure like its modularity, conductance, or clustering coefficient that they optimize.

We find it worth emphasizing that graph clustering [Flake et al., 2000; Girvan and Newman, 2002; Karypis and Kumar, 1998], although related, target a different problem. While their goal is to cluster the *whole* graph, we try to partition and connect a small *subset* of nodes, making use of the graph structure.

[Bhalotia et al., 2002] develop methods to find effective connection trees among keywords for browsing and querying. While conceptually similar, our problem definition and formulation have important differences, such as incorporating encoding length. [Leskovec et al., 2007a] use features extracted from query connection subgraphs together with machine learning techniques to predict quality of query results. The connection subgraph is composed of the induced subgraph returned by the query where the its disconnected components are simply joined by shortest paths.

[Guan et al., 2011] study a new measure called structural correlation to assess how strongly a set of query nodes (e.g. nodes affected by an event) are correlated via the graph structure. Their method provides a correlation score but not a connection subgraph. [Tong et al., 2010] finds the top-k nodes with the highest 'gateway-ness' score with respect to a given source and target node set, such that they collectively lie on most of the short paths from source nodes to target nodes. Their goal, rather than to find strong connections between query nodes, is to find a set of nodes that disconnect source and target nodes to the largest extent when removed from the graph.

Chapter 9

Anomalies in Plain Graphs

PROBLEM STATEMENT: *Given a large, weighted graph, how can we find anomalies? Which rules should be violated, before we label a node as an anomaly?*

Most anomaly detection algorithms focus on clouds of multi-dimensional points, as we described in the survey section. Our goal, on the other hand, is to spot strange nodes in a *graph*, with weighted edges. What patterns and laws do such graphs obey? What features should we extract from each node? In this chapter, we answer all these questions and develop our ODDBALL method. Main contributions of this work are:

1. *Feature extraction*: We propose to focus on neighborhoods, that is, a sphere, or a ball (hence the name ODDBALL) around each node(the *ego*): that is, for each node, we consider the induced subgraph of its neighboring nodes, which is referred to as the *egonet*. Out of the huge number of numerical features one could extract from the egonet of a given node, we give a carefully chosen list, with features that are both fast to compute, as well as effective in revealing outliers.
2. *Egonet patterns*: We show that egonets obey some surprising patterns (like the *Egonet Density Power Law* (*EDPL*), *EWPL*, *ELWPL*, and *ERPL*), which gives us confidence to declare as outliers the ones that deviate. We support our observations by showing that the *ERPL* yields the *EWPL*.
3. *Scalable algorithm*: Based on those patterns, we propose ODDBALL, a scalable, unsupervised method for anomalous node detection.
4. *Application on real data*: We apply ODDBALL to numerous real graphs (*DBLP*, political donations, and other domains) and we show that it indeed spots nodes that a human would agree are strange and/or extreme.

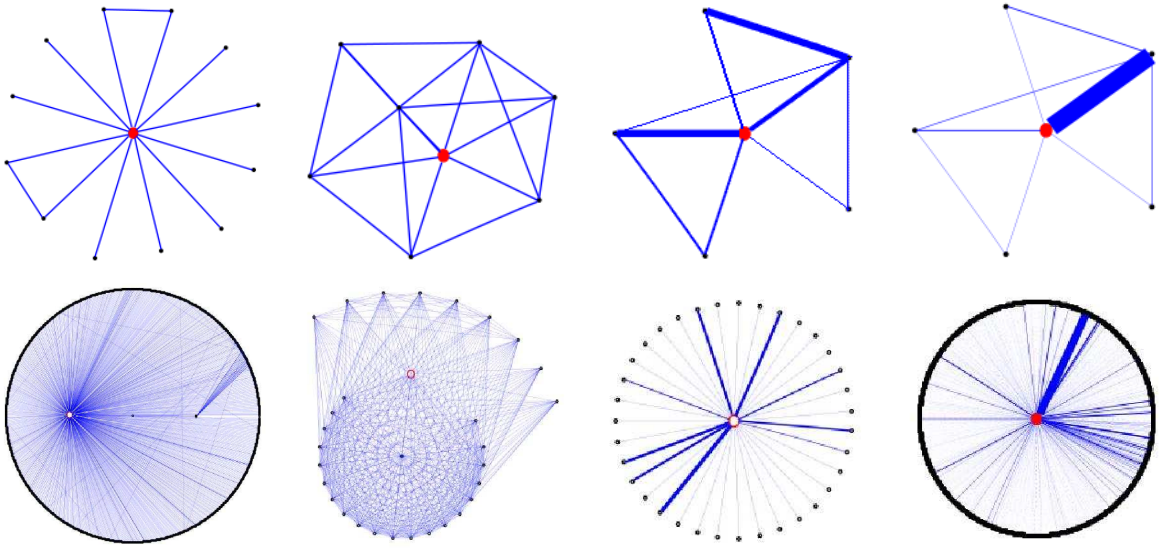
Jumping ahead, the major types of anomalous nodes we can spot are as follows (see Fig.9.1 for examples).

1. *Near-cliques* and *stars*: Detecting those nodes whose neighbors are very well connected

(near-cliques) or not connected (stars) turn out to be "strange": in most social networks, friends of friends are often friends, but either extreme (clique/star) is suspicious.

2. *Heavy vicinities*: If person i has contacted n distinct people in a who-calls-whom network, we would expect that the number of phonecalls (weight) would be proportional to n (say, 3xn or 5xn). Extreme total weight would be suspicious, indicating, e.g., faulty equipment that forces redialing.

3. *Dominant heavy links*: In the who-calls-whom scenario above, a very heavy single link in the 1-step neighborhood of person i is also suspicious, indicating, e.g., a stalker that keeps on calling only one of his/her contacts an excessive count of times.



(a) Near-star (b) Near-clique (c) Heavy vicinity (d) Dominant edge

Figure 9.1: Types of anomalies that ODDBALL detects. Top row: toy sketches of egonets (ego shown in larger, red circle). Bottom row: actual anomalies spotted in real datasets. (a) A near-star in *PostNet*: Instapundit, on Hurricane Katrina relief agencies (`instapundit.com/archives/025235.php`): An extremely long post, with many updates, and numerous links to diverse other posts about donations. (b) A near-clique in *PostNet*: `sizemore.co.uk`, who often linked to its own posts, as well as to its own posts in other blogs. (c) A heavy vicinity in *PostNet*: *blog.searchenginewatch.com/blog* has abnormally high weight wrt the number of edges in its egonet. (d) Dominant edge(s) in *CampOrg*: In FEC 2004, George W. Bush received a huge donation from a single donor committee: Democratic National Committee ($\sim\$87M$) (!) - in fact, this amount was *spent against* him; Next heaviest link ($\sim\$25M$): from Republican National Committee.

Next we give the primary observations and the description of ODDBALL, show experimental results, and list summary of contributions.

9.1 ODDBALL: Method Description

The basic research questions we start with are: (a) *what features should we use to characterize a neighborhood?* and (b) *what does a 'normal' neighborhood look like?* Both questions are open-ended, but we give some answers below.

First, we need to choose the value of k steps to study neighborhoods (see Definition 5). Given that real-world graphs have small diameter [Albert et al., 1999], we need to stay with small values of k, and specifically, we recommend $k = 1$. We report our findings only for $k = 1$, because using $k > 1$ does not provide any more intuitive or revealing information, while it has heavy computational overhead, possibly intractable for very large graphs.

9.1.1 Feature Extraction

The first of our two inter-twined questions is *which statistics/features to extract* from each node neighborhood. There is an infinite set of functions/features that we could use to characterize a neighborhood (number of nodes, one or more eigenvalues, number of triangles, effective radius of the central node, number of neighbors of degree 1, etc etc). Which of all should we use?

Intuitively, we want to select features that (a) are fast-to-compute and (b) will lead us to patterns/laws that most nodes obey, except for a few anomalous nodes. We spend a lot of time experimenting with about a dozen features, trying to see whether the nodes of real graphs obey any patterns with respect to those features. The majority of features lead to no obvious patterns, and thus we do not present them.

The trimmed-down set of features that are successful in spotting patterns, are the following:

1. N_i: number of neighbors (degree) of ego i,
2. E_i: number of edges in egonet i,
3. W_i: total weight of egonet i,
4. $\lambda_{w,i}$: principal eigenvalue of the *weighted* adjacency matrix of egonet i.
5. S_i: number of singleton neighbors of ego i of degree 1.

The next question is how to look for outliers, in such an n-dimensional feature space, with one point for each node of the graph. In our case, $n = 4$, but one might have more features depending on the application and types of anomalies one wants to detect. A quick answer to this would be to use traditional outlier detection methods for clouds of points using all the features.

In our setting, we can *do better*. As we show next, we group features into carefully chosen pairs, where we show that there are patterns of normal behavior (typically, power-laws). We flag those points that significantly deviate from the discovered patterns as anomalous. Among the numerous pairs of features we studied, the successful pairs and the corresponding type of anomaly are the following:

- E vs N: *CliqueStar*: detects near-cliques and stars
- W vs E: *HeavyVicinity*: detects many recurrences of interactions

- λ_w vs W: *DominantPair*: detects single dominating heavy edge (strongly connected pair)
- S vs N: *LoneStar*: detects almost-isolated stars

9.1.2 Laws and Observations

The second of our research questions is *what do normal neighborhoods look like*. Thus, it is important to find patterns ("laws") for neighborhoods of real graphs, and then report the deviations, if any. In this work, we report some new, surprising patterns:

For a given graph $\mathcal{G}$, node $i \in V(\mathcal{G})$, and the egonet $\mathcal{G}_i$ of node i;

Observation 9.1.1 (*EDPL*: Egonet Density Power Law) *the number of nodes N_i and the number of edges E_i of $\mathcal{G}_i$ follow a power law.*

$$E_i \propto N_i^{\alpha}, \;\; 1 \leq \alpha \leq 2.$$

In our experiments the *EDPL* exponent α ranged from 1.10 to 1.66. Fig. 9.3 illustrates this observation, for several of our datasets. Plots show E_i versus N_i for every node (green points); the black circles are the median values for each bucket of points (separated by vertical dotted lines) after applying logarithmic binning on the x-axis as in [Mcglohon et al., 2008]; the red line is the least squares(LS) fit on the median points. The plots also show a blue line of slope 2, that corresponds to cliques, and a black line of slope 1, that corresponds to stars. All the plots are in log-log scales.

Observation 9.1.2 (*EWPL*: Egonet Weight Power Law) *the total weight W_i and the number of edges E_i of $\mathcal{G}_i$ follow a power law.*

$$W_i \propto E_i^{\beta}, \;\; \beta \geq 1.$$

Fig. 9.5 shows the *EWPL* for (only a subset of) our datasets (due to space limit). In our experiments the *EWPL* exponent β ranged up to 1.29. Values of $\beta > 1$ indicate super-linear growth in the total weight with respect to increasing total edge count in the egonet.

Observation 9.1.3 (*ELWPL*: Egonet λ_w Power Law) *the principal eigenvalue $\lambda_{w,i}$ of the weighted adjacency matrix and the total weight W_i of $\mathcal{G}_i$ follow a power law.*

$$\lambda_{w,i} \propto W_i^{\gamma}, \;\; 0.5 \leq \gamma \leq 1.$$

Fig. 9.6 shows the *ELWPL* for a subset of our datasets. In our experiments the *ELWPL* exponent γ ranged from 0.53 to 0.98. γ=0.5 indicates uniform weight distribution whereas γ~1 indicates a dominant heavy edge in the egonet, in which case the weighted eigenvalue closely follows the maximum edge weight. γ=1 if the egonet contains only one edge.

Observation 9.1.4 (*ERPL*: Egonet Rank Power Law) *the rank $R_{i,j}$ and the weight $W_{i,j}$ of edge j in $\mathcal{G}_i$ follow a power law.*

$$W_{i,j} \propto R_{i,j}^{\theta}, \;\; \theta \leq 0.$$

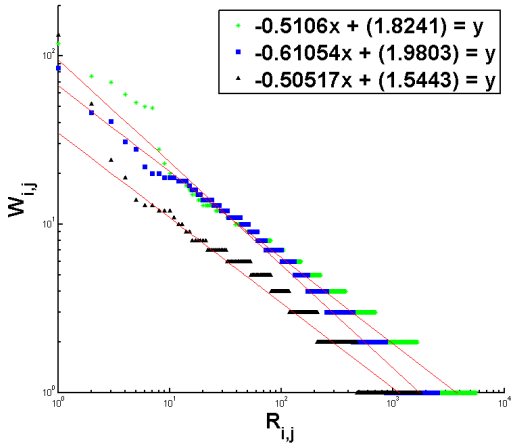


Figure 9.2: Weight $W_{i,j}$ vs. Rank $R_{i,j}$ for each edge j in the egonet of node i. Top three nodes with the highest edge count in their egonet from *BlogNet* are shown. LS line is fit in log-log scales pointing out a power-law relation (*ERPL*).

Here, $R_{i,j}$ is the rank of edge j in the sorted list of edge weights. *ERPL* suggests that edge weights in the egonet have a skewed distribution. This is intuitive; for example in a friendship network, a person could have many not-so-close friends (light links), but only a few close friends (heavy links). In Fig. 9.2 we show the *ERPL* for top three nodes with the highest number of edges in their egonet from *BlogNet* – other datasets have similar results.

Next we show that if the *ERPL* holds, then the *EWPL* also holds. Given an egonet $\mathcal{G}_i$, the total weight W_i and the number of edges E_i of $\mathcal{G}_i$, let $\mathcal{W}_i$ denote the ordered set of weights of the edges, $W_{i,j}$ denote the weight of edge j, and $R_{i,j}$ denote the rank of weight $W_{i,j}$ in set $\mathcal{W}_i$. Then,

Lemma 5 *ERPL implies EWPL, that is: If* $W_{i,j} \propto R_{i,j}^{\theta}$, $\theta \leq 0$, *then*

$$W_i \propto E_i^{\beta} \begin{cases} \beta = 1, \textit{ if } -1 \leq \theta \leq 0 \\ \beta > 1, \textit{ if } \theta < -1 \end{cases}$$

Proof

For brevity, we give the proof for $\theta < -1$ – other cases are similar. If $W_{i,j} = cR_{i,j}^{\theta}$, then $W_{min} = cE_i^{\theta}$ –the least heavy edge l with weight W_{min} is ranked the last, i.e. $R_{i,l} = E_i$. Thus we can write W_i as

$$\begin{aligned} W_i &= W_{min}E_i^{-\theta}\left(\sum_{j=1}^{E_i} j^{\theta}\right) \approx W_{min}E_i^{-\theta}\left(\int_{j=1}^{E_i} j^{\theta}dj\right) \\ &= W_{min}E_i^{-\theta}\left(\frac{j^{\theta+1}}{\theta+1}\Big|_{j=1}^{E_i}\right) = W_{min}E_i^{-\theta}\left(\frac{1}{-\theta-1} - \frac{1}{(-\theta-1)E_i^{-\theta-1}}\right) \end{aligned}$$

For sufficiently large E_i and given $\theta < -1$, the second term in parenthesis goes to 0. Therefore; $W_i \approx c'E_i^{-\theta}$, $c' = \frac{W_{min}}{-\theta-1}$. Since $\theta < -1$, $\beta > 1$. ∎

9.1.3 Anomaly Detection

We can easily use the observations given in part 3.2 in anomaly detection since anomalous nodes would behave away from the normal pattern. Let us define the y-value of a node i as y_i and similarly, let x_i denote the x-value of node i for a particular feature pair $f(x, y)$. Given the power law equation $y = Cx^\theta$ for $f(x, y)$, we define the outlierness score of node i to be

$$\textit{out-line(i)} = \frac{max(y_i, Cx_i^\theta)}{min(y_i, Cx_i^\theta)} * log(|y_i - Cx_i^\theta| + 1)$$

Intuitively, the above measure is the "distance to fitting line". Here we penalize each node with both the *number of times* that y_i deviates from its *expected* value Cx_i^θ given x_i, and with the logarithm of the *amount* of deviation. This way, the minimum outlierness score becomes 0 –for which the actual value y_i is equal to the expected value Cx_i^θ.

This simple and easy-to-compute method not only helps in detecting outliers, but also provides a way to sort the nodes according to their outlierness scores. However, this method is prone to miss some outliers and therefore could yield false negatives for the following reason: Assume that there exist some points that are far away from the remaining points but that are still located close to the fitting line. In our experiments with real data, we observe that this usually happens for high values of x and y. For example, in Fig. 9.3(a), the points marked with left-triangles ($\triangleleft$) are almost on the fitting line even though they are far away from the rest of the points.

We want to flag both types of points as outliers, and thus we propose to combine our heuristic with a density-based outlier detection technique. We used LOF [Breunig et al., 2000b], which also assigns outlierness scores *out-lof(i)* to data points; but any other outlier detection method would do, as long as it gives such a score. To obtain the final outlierness score of a data point i, one might use several methods such as taking a linear function of both scores and ranking the nodes according to the new score, or merging the two ranked lists of nodes, each sorted on a different score. In our work, we simply used the sum of the two normalized(by dividing by the maximum) scores, that is, *out-score(i)* = *out-line(i)*+*out-lof(i)*.

9.2 Experimental Results

9.2.1 Structural Anomalies

CliqueStar and the *LoneStar* are used for finding structural anomalies. Here, we are interested in the communities that neighbors of a node form. In particular, *CliqueStar* detects anomalies having to do with near-cliques and near-stars and *LoneStar* catches those nodes whose egonet is weakly connected to the rest of the graph: most of the neighbors being of degree 1, the egonet is not only star-like, but also is almost isolated from the rest of the network.

CliqueStar

Here, we are interested in the communities that neighbors of a node form. In particular, *CliqueStar* detects anomalies having to do with near-cliques and near-stars. Using *CliqueStar*, we were successful in detecting many anomalies over the unipartite datasets (although it is irrelevant for bipartite graphs since by nature the egonet forms a "star").

In social media data *PostNet* and *BlogNet*, we detected posts or blogs that had either all their neighbors connected (cliques) or mostly disconnected (stars). We show some illustrative examples along with descriptions from *PostNet* in Fig. 9.1. See Fig.9.3a for the detected outliers on the scatter-plot from the same dataset. In *BlogNet*(Fig.9.3b), the method detected several "link blogs", blogs devoted to posting links to a wide array of sources that do not always have similar content. For instance `mfisn.com` links to tech blogs, politics stories, and flash cartoons. `dev.upian.com/hotlinks` also links to a wide range of other posts each day.

In *Enron*(Fig.9.3c), the node with the highest outlier score turns out to be Kenneth Lay, who was the CEO and is best known for his role in the Enron scandal in 2001. Our method reveals that none of his over 1K contacts ever sent emails to each other. In *Oregon* (Fig.9.3d), the top outliers are the three large ISPs (Verizon, Sprint and AT&T).

LoneStar

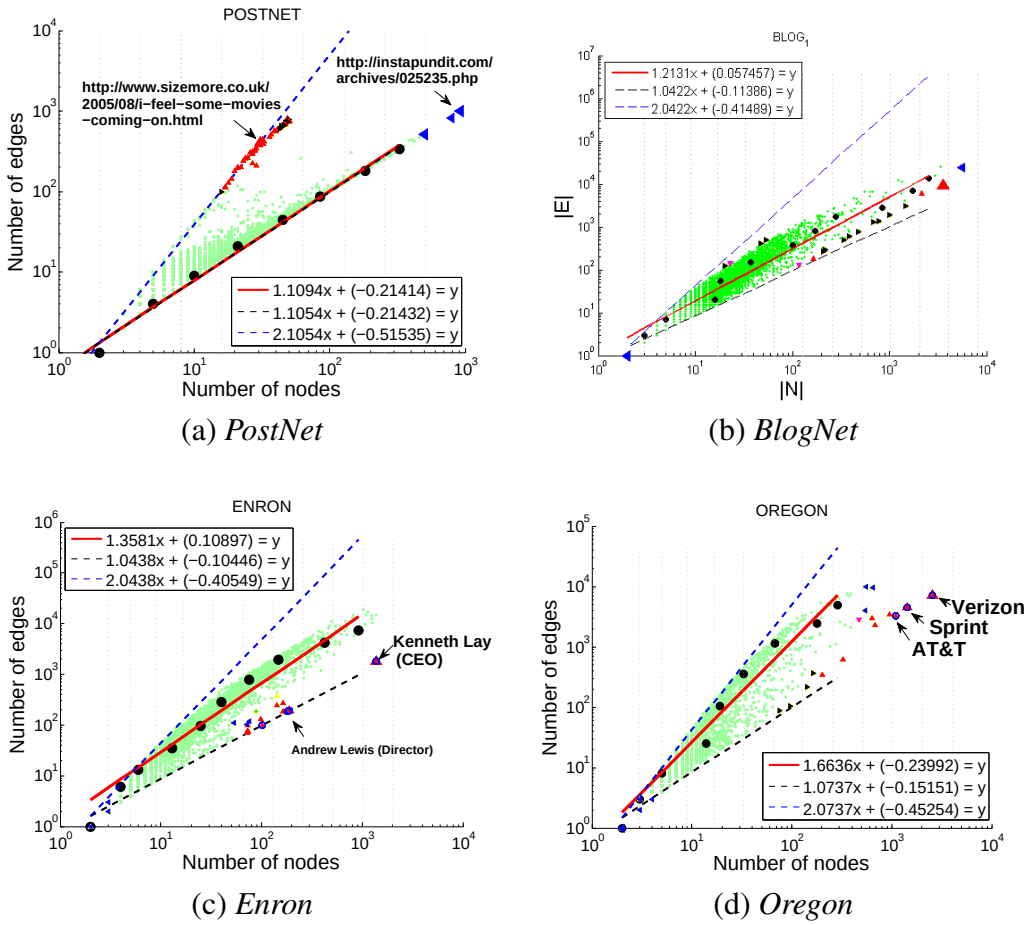
Some results of this method were similar to those of *CliqueStar*, but for different reason: here, we look not simply at whether the neighbors connect to each other, but whether they connect to the rest of the network at large. This gives important implicit information about the 2-step neighborhood. Thus, we are looking for "isolated stars" or "hubs".

In *CampOrg*, the most outlying *Committee* (Fig.9.4a) is the National Committee for an Effective Congress, which donated to candidates who received money from nowhere else. This makes intuitive sense as this committee claims to help "progressive" candidates[1], that may have limited support otherwise. On the other hand, *Candidates* (Fig.9.4b) reveals the top two competing candidates, John F. Kerry and George W. Bush, who had several committees devoted exclusively to their campaigns.

In *CampIndiv*, top outliers for *Committees* (Fig.9.4d) are Bush-Cheney '04 Inc. and John Kerry for President Inc. both of which raised money from a vast number of donors, most of whom did not donate to anywhere else. Another interesting result here is that Kerry Victory 2004 received money from donors who donated to other committees as well, but as we will see later, Kerry Victory 2004 donated money only to a single candidate, John F. Kerry.

In *Auth-Conf* (Fig.9.4c), we see that top outliers include ICRA, ISCAS and SIGUCCS with their authors publishing to no other conferences. This might be due to the fact that these conferences are the only ones related to a particular research area. Winter Simulation Conference, another anomaly here, happens to be a conference of a unique type.

[1] www.ncec.org



(a) *PostNet* (b) *BlogNet*

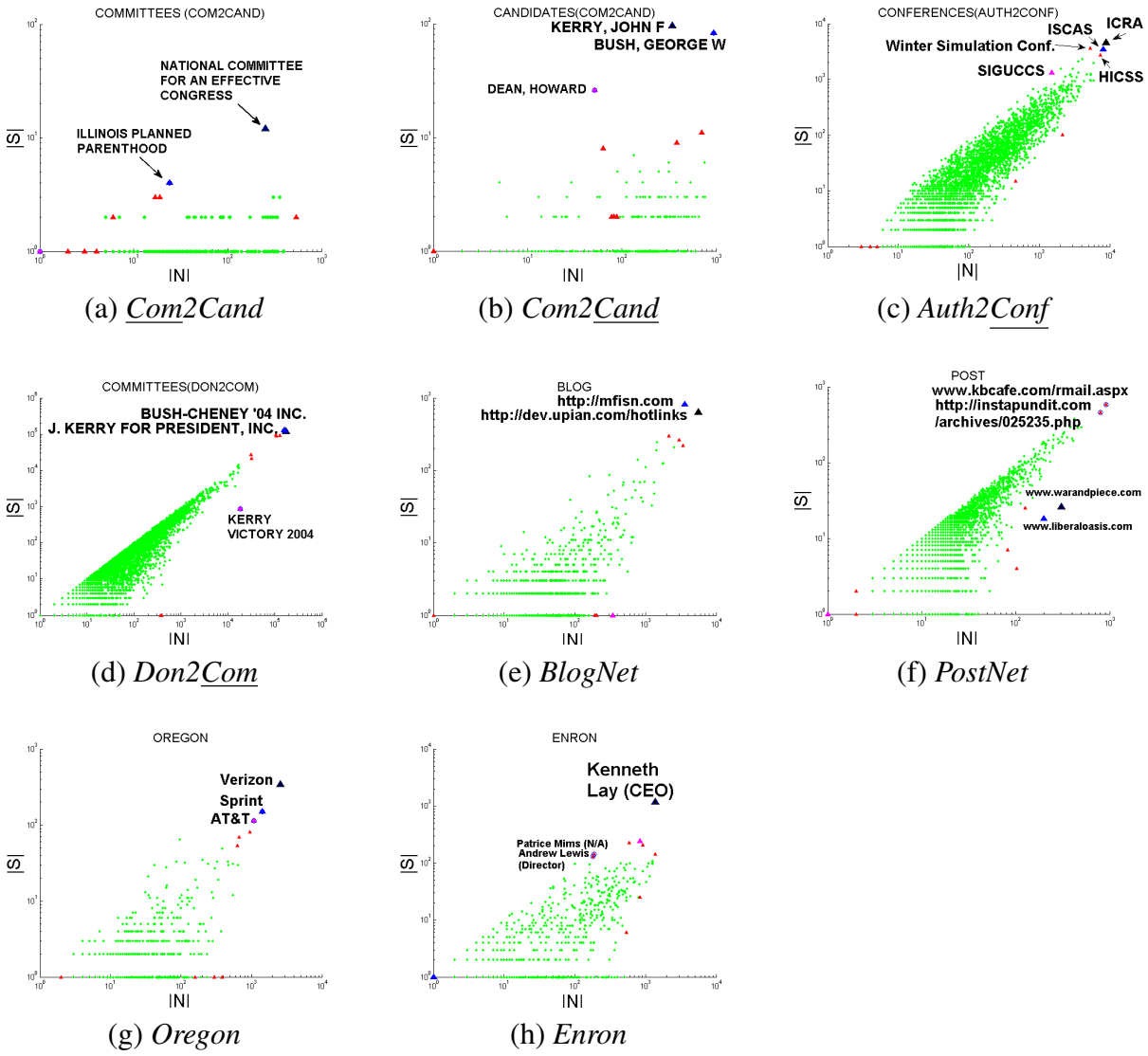
(c) *Enron* (d) *Oregon*

Figure 9.3: Illustration of the Egonet Density Power Law (*EDPL*), and the corresponding anomaly *CliqueStar*, with outliers marked with triangles. Edge count versus node count (log-log scale); red line is the LS fit on the median values (black circles); dashed black and blue lines have slopes 1 and 2 respectively, corresponding to stars, and cliques. Top outlying points are enlarged. Most striking outlier: Ken Lay (CEO of Enron), with a star-like neighborhood. See text for more discussion and Fig.9.1 for example illustration from *PostNet*.

In *BlogNet* (Fig.9.4e), the same "link blogs" as above are detected as "isolated stars": most of their neighbors are not only blogs that do not link to each other, but in fact are linked to nobody else at all. There were similar points in *PostNet*, where the post has many links to otherwise isolated posts. For example, one outlier is KBCafe[2], a homepage to a piece of email software linked several times by a single blog in different posts. See Fig.9.1 for further details.

In *PostNet* (Fig.9.4f), we also found points on the other extreme: posts that were unusually well-connected. Looking further, we found that there were several feeds that claimed that the post permalink was the actual blog home page– this boosted the degree to these so-called "posts", since whenever another post linked to the homepage of this blog, we registered it as linking to

[2] `www.kbcafe.com/rmail.aspx`



(a) *Com2Cand* (b) *Com2Cand* (c) *Auth2Conf*

(d) *Don2Com* (e) *BlogNet* (f) *PostNet*

(g) *Oregon* (h) *Enron*

Figure 9.4: Illustration of the near-isolated star anomaly *LoneStar*. Plots show total number of nodes of degree 1 in the egonet vs. degree of the ego for all nodes(in log-log scale). Most striking anomalies are marked with triangles and labeled, with detailed explanation in text. See Fig.9.1 for an example illustration from *PostNet*.

a single post. These outliers[3] had many neighbors among political blogs which pointed to other blogs, so they were unusually non-isolated. So, in this case, anomalous behavior signaled errors in the feeds.

In *Oregon* (Fig.9.4g) and *Enron* (Fig.9.4h), top 3 detected points were the same as the ones detected with *CliqueStar*. This is intuitive as nodes that form a near-star are expected to have low degrees– indeed degree 1, making the star not only sparse but also isolated.

[3] `www.warandpiece.com` and `www.liberaloasis.com`

9.2.2 Weight Anomalies

HeavyVicinity and the *DominantPair* are used for finding weight anomalies. Here, we are interested in the weights of edges in the egonets of nodes. In particular, *HeavyVicinity* detects anomalies having to do with high weight neighborhoods and *DominantPair* catches those egonets in which a particular edge weight dominates the others.

HeavyVicinity

HeavyVicinity detects nodes that have considerably high total edge weight compared to the number of edges in their egonet. In our datasets, *HeavyVicinity* detected "heavy egonets", with anomalies marked in Fig.9.5. In *PostNet* (Fig.9.5h), often anomalous posts were the ones that linked to another post repeatedly [4] or were listed as the blog's home page[5].

In *BlogNet* (Fig.9.5g), we detected blogs that linked to just a few other sources, either a single post or multiple posts from the same blog. Interesting anomalies are Automotive News Today[6], which linked 241 times to the GM blog[7], but never to any other blog in our dataset. One political blog [8] had 1816 total in- and out-links, to only 30 other blogs. On the other extreme, Bandelier[9] had an abnormally high number of edges, but a low weight upon each. The blog had a single post in the dataset, so it was naturally uncommon for blogs to link to that blog multiple times.

HeavyVicinity revealed interesting observations in bipartite graphs as well, spotting duplicates and irregularities. In *CampOrg* (Fig.9.5a,b), we see that Democratic National Committee gave away a lot of money compared to the number of candidates that it donated to. In addition, (John) Kerry Victory 2004 donated a large amount to a single candidate, whereas Liberty Congressional Political Action Committee donated a very small amount ($5) to a single candidate. Looking at the *Candidates* plot for the same bipartite graph, we also flagged Aaron Russo, the lone recipient of that PAC. (In fact, Aaron Russo is the founder of the Constitution Party which never ran any candidates, and Russo shut it down after 18 months.)

In *CampIndiv* (Fig.9.5c,d), we see that Bush-Cheney '04 Inc. received a lot of money from a single donor. This is strange, as it was also listed as an anomaly in *LoneStar*, having many degree 1 donors. Looking at the data, we notice that that committee is listed twice with two different IDs.

On the other hand, we notice that the Kerry Committee received less money than would be expected looking at the number of checks it received in total. Further analysis shows that most of the edges in its egonet are of weight 0, showing that most of the donations to that committee have actually been returned.

[4] `leados.blogs.com/blog/2005/08/overview_of_cia.html`
[5] `blog.searchenginewatch.com/blog`
[6] `www.automotive-news-today.com`
[7] `fastlane.gmblogs.com`
[8] `nocapital.blogspot.com`
[9] `www.bandelier.com`

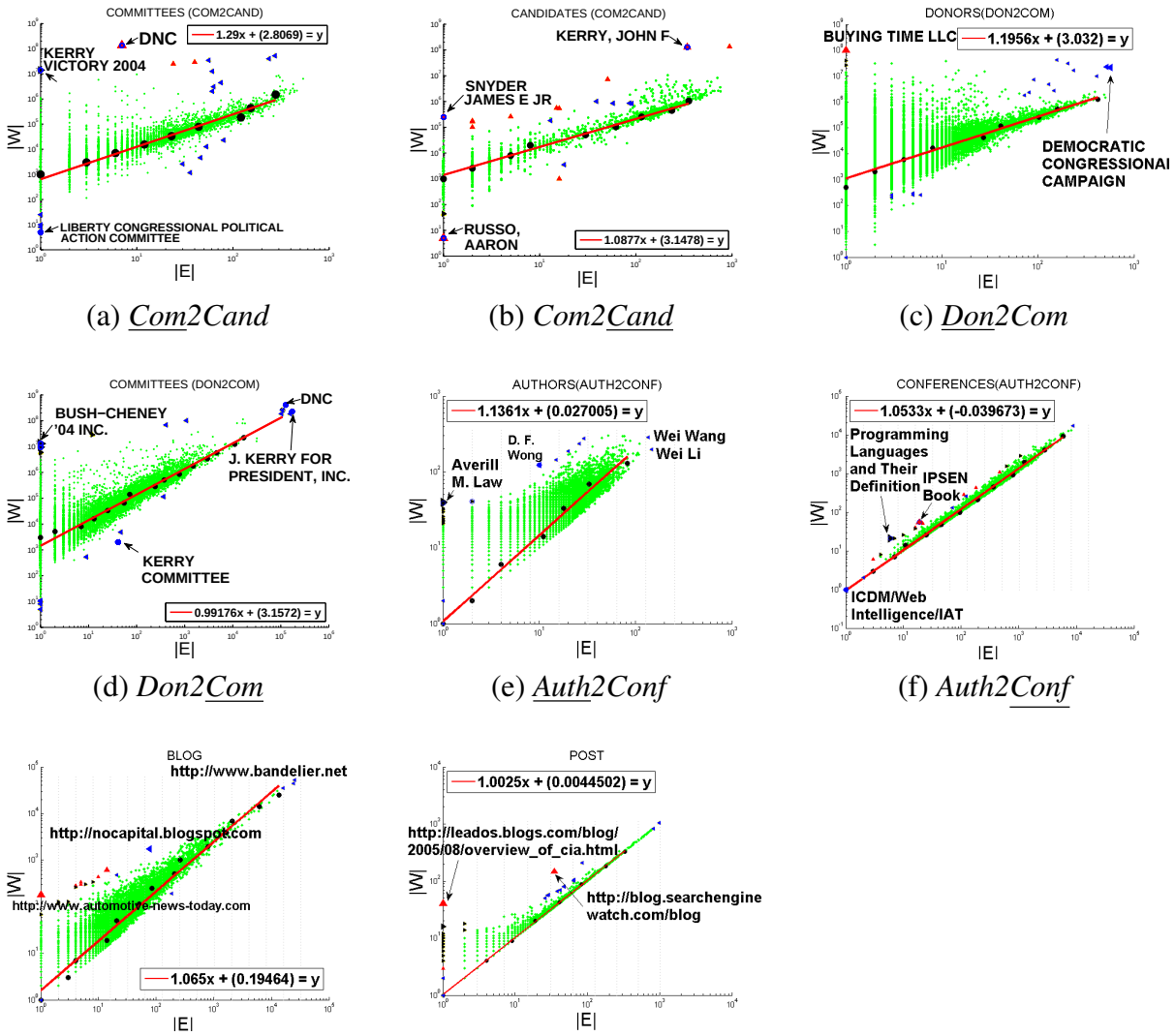
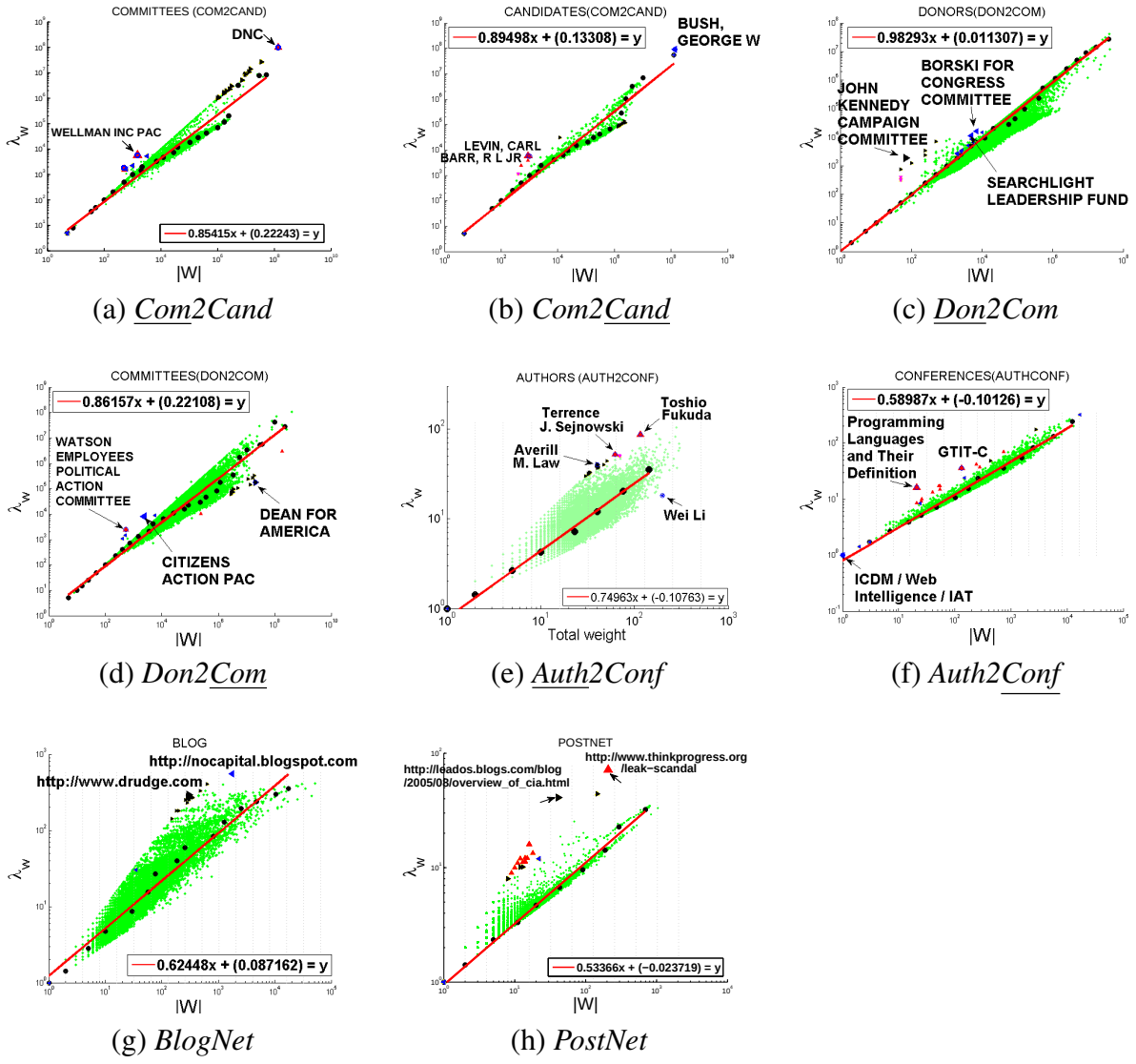


Figure 9.5: Illustration of the Egonet Weight Power Law (*EWPL*) and the weight-edge anomaly *HeavyVicinity*. Plots show total weight vs. total count of edges in the egonet for all nodes(in log-log scales). Detected outliers include Democratic National Committee and John F. Kerry (in FEC campaign donations), and Averill M. Law in DBLP. See text for more discussions, and Fig.9.1 for an illustrative example from *PostNet*.

In *Auth-Conf* (Fig.9.5e,f), Averill M. Law published 40 papers to the Winter Simulation Conference and nowhere else. This might be due to the fact that there exists no other conference that would capture the interest of that author. In fact, under *LoneStar*, we saw that Winter Simulation Conference was one of those conferences with most of its authors with degree 1, pointing to the same possibility that it is a unique conference in a particular area.

Other interesting points are Wei Wang and Wei Li. Those authors have many papers, but they publish them at as many distinct conferences, probably once or twice at each conference.



(a) *Com2Cand* (b) *Com2Cand* (c) *Don2Com*

(d) *Don2Com* (e) *Auth2Conf* (f) *Auth2Conf*

(g) *BlogNet* (h) *PostNet*

Figure 9.6: Illustration of the Egonet λ_w Power Law (*ELWPL*) and dominant heavy link anomaly *DominantPair*. Top anomalies are marked with triangles and labeled. See text for details for each dataset and Fig.9.1 for an illustrative example from *CampOrg*.

DominantPair

DominantPair measures whether there is a single dominant heavy edge in the egonet. In other words, this method detected "bursty" if not exclusive edges. In *PostNet* (Fig.9.6h) nodes such as ThinkProgress's post on a leak scandal [10] and A Freethinker's Paradise post [11] linking several times to the ThinkProgress post were both flagged. In *BlogNet* (Fig. 9.6g), we detected a Drudge

[10] `www.thinkprogress.org/leak-scandal`
[11] `leados.blogs.com/blog/2005/08/overview_of_cia.html`

blogger [12], who had 298 links, all but 4 to another blogger in the same site [13]. Nocapital also appeared here, since it had around 300 links each to two other blogs.

In *CampOrg* (Fig.9.6a,b), Democratic National Committee is one of the top outliers. We would guess that the single large amount of donation was made to John F. Kerry. Counterintuitively, we see that that amount was spent for an opposing advertisement against George W. Bush.

In *CampIndiv* (Fig.9.6c,d), there exists points above the slope 1 line. Further inspection shows that these points correspond to authorities having negative weighted edges due to returns. Points below the fitting line, such as Dean For America on the *Committees* plot, correspond to authorities with even weight distributions on the edges, without a specific dominant heavy link.

DominantPair flagged extremely focused authors (those publish heavily to one conference) in the *DBLP* data, shown in Fig. 9.5(e,f). For instance, Toshio Fukuda has 115 papers in 17 conferences (at the time of data collection), with more than half (87) of his papers in one particular conference (ICRA). Alberto-Sangiovanni Vincentelli published 279 papers to 45 distinct conferences, with 76 of his papers in DAC. On the *Conferences* plot, Programming Languages and Their Definition has 21 papers from 6 authors, with 16 papers by one particular author (Hans Bekic).

9.2.3 Scalability

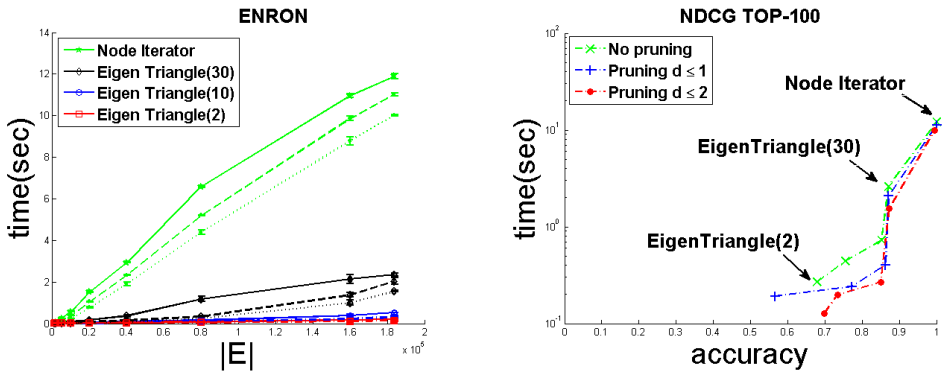
Major computational cost of our method is in feature extraction. In particular, computing those features, such as the total number of edges and total weight, for the egonets is the bottleneck as one needs to find the induced 1-step neighborhood subgraphs for all nodes in the network.

The problem of finding the number of edges in the egonet of a given node can be reduced to the problem of triangle counting. One straightforward *listing* method for local triangle counting is the *Node-Iterator* algorithm. *Node-Iterator* considers each one of the N nodes and examines which pairs of its neighbors are connected. Time complexity of the algorithm is $O(Nd_{max}^2)$. Approximate streaming algorithms for local triangle counting can be applied to reduce the time complexity to $O(E \log N)$ with space complexity $O(N)$ [Becchetti et al., 2008].

In our experiments, we use *Eigen-Triangle*, proposed in [Tsourakakis, 2008], which uses eigenvalues/vectors to approximate the number of paths of length three, i.e. local triangle counts, without performing actual counting. We compare its performance to *Node-Iterator*, which gives exact counts. To improve speed even more, we propose pruning low degree nodes. Notice that removing degree-1 nodes has no effect on the number of triangles in the graph. We also try removing degree-2 nodes; this would remove some triangles but hopefully would not change the relative number of triangles across nodes drastically and still reveal similar outliers.

In Figure 9.7a, we show computation time for *Node-Iterator* (green), and for *Eigen-Triangle* using 2(red), 10(blue), and 30(black) eigenvalues versus graph size in terms of number of edges for Enron ($E \approx 180K$). Solid(–), dashed(−−), and dotted(. . .) lines are for no pruning, after

[12] `drudge.com/user/rcade`
[13] `drudge.com/user/gzlives`



(a) (b)

Figure 9.7: (a) Computation time of computing egonet features vs. number of edges in *Enron*. Effect of pruning degree-1 and degree-2 nodes on computation time of counting triangles. Solid(–): no pruning, dashed(−−): pruning $d \leq 1$, and dotted(. . .): pruning $d \leq 2$ nodes. Computation time increases linearly with increasing number of edges, whereas it decreases with pruning. (b) Time vs. accuracy. Effect of pruning on accuracy of finding top anomalies as in the original ranking before pruning. New rankings are scored using Normalized Cumulative Discounted Gain. Pruning reduces time for both *Node-Iterator* and *Eigen-Triangle* for different number of eigenvalues while keeping accuracy at as high as~1 and~.9, respectively.

pruning $d \leq 1$, and $d \leq 2$ nodes, respectively. We empirically note that computation time grows linearly with increasing graph size and also reduces with pruning. (Experiments ran on a Pentium class workstation, with 16GB of RAM, running Linux Fedora Core. To account for possible variability due to system state, each run is repeated 10 times and execution time results are averaged. Error bars show the variance across repeated runs.)

Pruning low-degree nodes and using *Eigen-Triangle* reduces computation time, however, they only provide approximate answers. In order to quantify their accuracy, we compare the rank list of outliers returned by *Node-Iterator* to the rank list of outliers returned by each approximate method. To measure how rankings changed with approximation compared to the original rankings, we use Normalized Discounted Cumulative Gain(NDCG) which is prevailingly used in Information Retrieval for measuring the effectiveness of search engines. The premise of NDCG is that highly ranked items in the original list appearing lower in the approximated list are penalized logarithmically proportional to their positions in the approximated list.

Figure 9.7b shows time vs. NDCG scores for *Eigen-Triangle* using 2, 5, 10 and 30 eigenvalues, and also *Node-Iterator* for top k anomalies. For brevity, we only show ranking scores for k=100. *, +, and o symbols represent no pruning, pruning $d \leq 1$, and $d \leq 2$ nodes, respectively. We notice that while reducing computation time, pruning low degree nodes as well as using *Eigen-Triangle* keeps the accuracy at as high as~.9 for *Eigen-Triangle*(30), and above 0.55 for *Eigen-Triangle*(2).

9.3 Summary of contributions

In this chapter, we introduced one of few work that focus on anomaly detection in *graph* data, including weighted graphs. Our major contributions are summarized as the following:

- Proposing to work with "egonets", that is, the induced sub-graph of the node of interest and its neighbors; we give a small, carefully designed list of numerical features for egonets.
- Discovery of new patterns that egonets follow, such as patterns in density (Obs.9.1.1: *EDPL*), weights (Obs.9.1.2: *EWPL*), principal eigenvalues (Obs.9.1.3: *ELWPL*), and ranks (Obs.9.1.4: *ERPL*). Proof of Lemma 1, linking the *ERPL* to the *EWPL*.
- ODDBALL[14], a fast, unsupervised method to detect abnormal nodes in weighted graphs. Our method does not require any user-defined constants. It also assigns an "outlierness" *score* to each node. Possible approximations in feature extraction provide speed-up, keeping accuracy at as high as ~ 0.9.
- Experiments on real graphs of over 1 million nodes, where ODDBALL reveals nodes that indeed have strange or extreme behavior.

Our future work will generalize ODDBALL to time-evolving graphs, where the challenge is to find patterns that neighborhood sub-graphs follow, track the behavior of nodes, and to extract features incrementally over time.

[14]Source code of ODDBALL: `www.cs.cmu.edu/~lakoglu/#code`

Chapter 10

Anomalies in Binary-attributed Graphs

PROBLEM STATEMENT: *Given a graph with node attributes, how can we find meaningful patterns such as clusters of nodes and clusters of attributes, as well as bridges and outliers?*

Real graphs often have nodes with attributes, in addition to connectivity information. For example, social networks contain both the friendship relations as well as user attributes such as interests and likes. Other examples include gene interaction networks with gene expression information, phone call networks with customer demographics. Both types of information can be described by a graph in which nodes represent the objects, edges represent the relations between them, and attribute vectors associated with the nodes represent their attributes. Such graph data is often referred to as an *attributed graph*.

Given such an attributed graph how can we find meaningful patterns, clusters of nodes, clusters of attributes, and anomalies? For example, consider the case of YouTube in which graph nodes represent users and YouTube-group memberships represent attributes. Given the who-friends-whom and who-belongs-to-which YouTube groups information, how can we summarize and make sense out of it?

In this chapter, we introduce PICS for mining attributed graphs. Specifically, PICS finds *cohesive clusters* of nodes that have similar connectivity patterns and exhibit high attribute homogeneity. PICS additionally groups the node attributes into attribute-clusters. These patterns further help us to spot anomalies and bridges. A key characteristic of PICS is that it is *parameter-free*; it can recover the number of necessary clusters without any user intervention. For example, our algorithm automatically finds coherent user clusters and YouTube-group clusters as shown in Figure 10.1. As a first observation, the algorithm automatically discovers clusters of users who like the 'anime' genre and form a so-called 'core and periphery' pattern: the core consists of the users in the bottom-right dark square of the blue square labeled as '2'; also notice that the anime fans overwhelmingly belong to a coherent set of YouTube-groups (blue square labeled as 'A2'). As we show in Table 10.2, those groups include *anime4ever*, *narutoholics*, and *crazyforanime*. More discussion on YouTube results is given in experiments, §10.2.2.

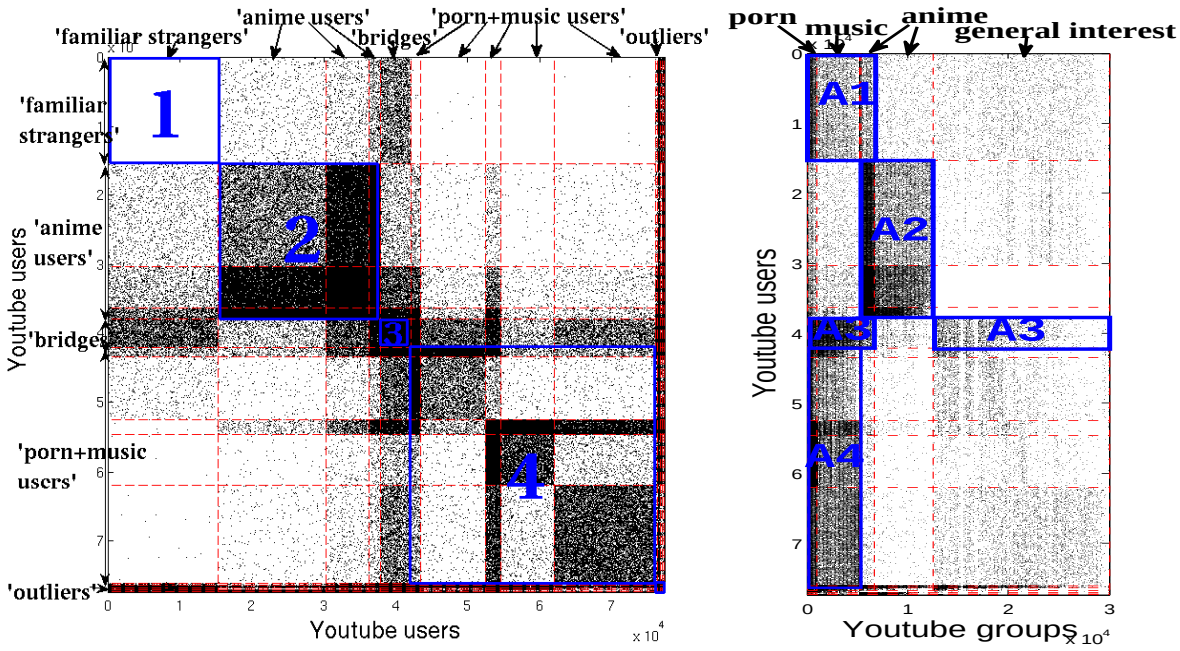


Figure 10.1: PICS on YOUTUBE finds clusters of users with similar connectivity and attribute coherence. Left: the adjacency matrix (users-to-users); right: the attribute matrix (users-to-YouTubeGroups). Both matrices are carefully arranged by PICS, revealing patterns: e.g., the anime fans are heavily connected, and they focus on the same YouTube-groups. See §10.2.2 for more discussion.

In a nut-shell, the contributions of this work are the following:

- *Algorithm design:* We introduce PICS, a novel clustering method to summarize graphs with node attributes. In effect, it groups the nodes with similar connectivity patterns into *cohesive clusters* that have high attribute homogeneity. Besides, it also clusters the attributes into attribute-clusters.

- *Automation:* PICS *does not* require any user-specified input such as the number of clusters, similarity functions or any sort of thresholds.

- *Scalability:* The run time of PICS grows *linearly* with total input graph and attribute size.

- *Effectiveness:* We evaluate our method on a diverse collection of real data sets with tens of thousands of nodes and attributes. Our results show that PICS successfully recovers cohesive node clusters, reveals the bridge and outlier nodes, and groups attributes into meaningful clusters.

Next we highlight the main challenges associated with simply extending traditional clustering algorithms to solve our problem, followed by method description and experiment results.

Why not simple extensions?

E1 Represent the relations of the objects as additional attributes, and then perform flat clustering on this extended feature space. The challenges with this approach are two-fold: First, it leads to a very large number of features and thus is faced with the curse of dimensionality [Beyer et al., 1999]. Second, it yields two separate types of attributes where it is not clear how to weigh those different sets for clustering.

E2 Represent the attributes of the objects as additional nodes in the graph, introduce new edges from the original nodes to the attribute nodes they exhibit, and then perform graph clustering on this extended graph. Again, there are two challenges with this approach: First, the graph grows with new nodes and edges, which may be quite numerous. Additionally, it is not clear how to do clustering in this heterogeneous graph which contains two types of nodes and edges.

E3 Introduce edges between all pairs of nodes where edges are weighted by the existence of connectivity and attribute similarity: This approach requires quadratic computation of pair-wise similarities and is thus untractable for large graphs. It also requires a careful choice of a similarity function.

In summary, clustering attributed graphs by applying traditional clustering approaches presents several nontrivial challenges.

10.1 PICS: Method Description

10.1.1 Problem Description

In this work, we address the problem of *finding cohesive clusters* in an attributed graph. Specifically, given a graph with n nodes and their binary connectivity information, where nodes are associated with f binary attributes (interchangeably, features), our goal is to group the nodes into k, and group the features into l disjoint clusters such that the nodes in the same cluster have "similar connectivity" and also exhibit high "feature coherence" (interchangeably we also use the term feature similarity). Informally, a set of nodes have "similar connectivity" if the set of nodes in the graph they connect to "highly" overlap. Similarly, a set of nodes have high "feature coherence" if the set of features they exhibit "highly" overlap.

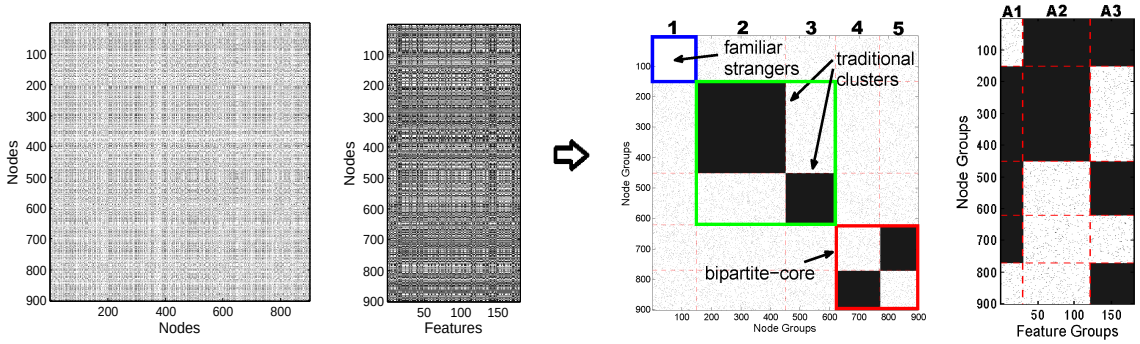
Synthetic Graph Example

To elaborate on the terminology, we give an example in Figure 10.2. PICS detects 5 node-clusters and 3 feature-clusters in this example graph. Notice that the nodes in the same cluster agree on their features to a high extent, i.e. have high feature coherence; for example nodes in node-cluster 1 exhibit features in feature-clusters 2 and 3, nodes in node-cluster 2 exhibit features in feature-clusters 1 and 2, and so on. In addition, notice the nodes in the same cluster having similar connections among themselves as well as to the rest of the graph. For example, nodes in

node-cluster 2 (similarly node-cluster 3) are densely connected among themselves but scarcely to the rest, nodes in node-cluster 4 are densely connected to nodes in node-cluster 5 (and vice versa) and scarcely to the rest, and so on.

Note that the nodes in a cluster that PICS finds may not necessarily be densely connected among themselves. In fact, the nodes in node-cluster 1 in the example graph are not connected to each other at all! They, however, share the same set of features and still have similar connectivity to the graph. Simply put, they are "familiar strangers". Similarly, nodes in node-clusters 4 and 5 are not connected within the clusters but across each other, forming a "bipartite core".

We would like to point out that while the traditional graph clustering algorithms would recover node-clusters 2 and 3, PICS can find additional type of clusters such as familiar strangers and bipartite cores, providing a richer analysis of a given graph. This derives from our more general cluster definition of nodes with "similar", rather than "dense", connectivity.



(a) Original **A** and **F** (b) PICS **A** and **F**

Figure 10.2: PICS on a synthetic dataset with n=900 nodes and f=180 features. Notice 5 node and 3 feature clusters. Nodes in the same cluster exhibit high feature homogeneity and have similar connectivity patterns. Note that similar connectivity does not only imply but also includes dense connectivity; while node-clusters 2 and 3 are densely connected within the clusters, node-clusters 1, 4, and 5 are not. See §10.1.1 for more details.

10.1.2 Problem Formulation

The main questions that arise given the above problem definition are: How should we decide the number of node and feature clusters, i.e., k and l, respectively? How can we assign the nodes and the features to their "proper" clusters? How much overlap of the features is "high" enough? We address these questions without making the users have to set any parameters such as the number of clusters, feature similarity thresholds or make them have to choose from a large collection of similarity functions. In fact, automation is exactly one of the main contributions of our approach.

Our starting point is data compression. Specifically, we want to compress *two*, *inter-related* matrices *simultaneously*. The first matrix is the $n \times n$ binary connectivity matrix $\mathbf{A}$, and the second is the $n \times f$ binary feature matrix $\mathbf{F}$ (note that although we consider binary graphs in our work, similar ideas can also be applied to weighted graphs). Our goal then is to compress (conceptually, summarize) these matrices simultaneously by looking at partitions/clusters/groups (i.e. homogeneous, rectangular "blocks") of both low and high densities. The reason for operating on the matrices simultaneously is simply that the two matrices are inter-related: the node groups should be homogeneous in both the connectivity matrix $\mathbf{A}$ *as well as* in the feature matrix $\mathbf{F}$.

The natural question is, how many rectangular blocks should we have in each matrix $\mathbf{A}$ and $\mathbf{F}$? To compress these matrices efficiently, we need to have several highly homogeneous blocks. On the other hand, having more clusters allows us to obtain more homogeneous blocks (at the very extreme, we can have $n \times n + n \times f$ blocks, each having perfect homogeneity of 0 or 1). The best compression model should achieve a proper trade-off between these two factors of homogeneity (data description complexity) and the number of blocks (model description complexity). To achieve this goal we use the MDL principle [Grünwald, 2007], a model selection criterion based on lossless compression principles –we design a cost criterion that we aim to minimize in which costs are based on the number of bits required to transmit both the "summary" of the structure (model) as well as each rectangular block (data) within the structure.

In our formulation, we address the hard-clustering problem in which each node (and each attribute) is assigned to a single cluster. An interesting research direction is to generalize our formulation for soft-clustering, where nodes could belong to multiple node-clusters.

Notation

Let k denote the desired number of disjoint node-clusters and let l denote the desired number of disjoint feature-clusters. Let $R : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, k\}$ and $C : \{1, 2, \ldots, f\} \rightarrow \{1, 2, \ldots, l\}$ denote the assignments of nodes to node-clusters and features to feature-clusters, respectively. We will refer to (R, C) as a *mapping*. To better understand a mapping, given the node-clusters and feature-clusters, let us rearrange the rows and columns of the connectivity matrix $\mathbf{A}$ such that all rows corresponding to node-cluster-1 are listed first, followed by rows in node-cluster-2, and so on. We also rearrange the columns in the same fashion. Note that the row and column arrangements for $\mathbf{A}$ will be the same. One can imagine that such a rearrangement sub-divides the matrix $\mathbf{A}$ into k^2 two-dimensional, rectangular blocks, which we will refer to as $B^A_{ij}, i, j = 1, \ldots, k$.

Similarly, we can rearrange the rows of the feature matrix $\mathbf{F}$ such that all rows corresponding to node-cluster-1 are listed first, followed by rows in node-cluster-2, and so on. One can realize that the row arrangements of $\mathbf{A}$ and $\mathbf{F}$ matrices will be the same. In a manner different than for $\mathbf{A}$, we will rearrange the columns of the feature matrix $\mathbf{F}$ such that all columns corresponding to feature-cluster-1 are listed first, followed by columns in feature-cluster-2, and so on. As a result, we will obtain rectangular blocks denoted as $B^F_{ij}, i = 1, \ldots, k$ and $j = 1 \ldots, l$ for $\mathbf{F}$. Finally, let the dimensions of B^F_{ij} and B^A_{ij} be (r_i, c_j) and (r_i, r_j), respectively.

10.1.3 Objective Function Formulation

We now describe a two-part cost criterion for the (lossless) compression of the connectivity and feature matrices $\mathbf{A}$ and $\mathbf{F}$. The compression cost can be thought of as the total number of bits required to transmit these matrices over a network channel. The first part is the model description cost that consists of describing the mapping (R, C). The second part is the data description cost that consists of encoding the sub-matrices (i.e., the "blocks" B_{ij}), given the mapping. Intuitively, a good choice of (R, C) would simultaneously compress $\mathbf{A}$ and $\mathbf{F}$ well, and as a result yield a low total description cost.

Next, we describe those two parts in more detail and then give the total encoding cost (i.e. our objective function).

Model Description Cost

This part consists of encoding the number of node and feature clusters as well as the corresponding mapping.

- The number of nodes n and the number of features f (i.e., matrix dimensions) require $\log^\star n + \log^\star f$ bits, where $\log^\star$ is the universal code length for integers [Rissanen, 1983]. This term is independent of any particular mapping.
- The number of node and feature clusters (k, l) require $\log^\star k + \log^\star l$ bits.
- The node and feature cluster assignments with arithmetic coding require $nH(P) + fH(Q)$ bits, where H denotes the Shannon entropy function, P is a multinomial random variable with the probability $p_i = \frac{r_i}{n}$ and r_i is the size of the i-th node cluster, $1 \leq i \leq k$. Similarly, Q is another multinomial random variable with the probability $q_j = \frac{c_j}{f}$ and c_j is the size of the j-th feature cluster, $1 \leq j \leq l$.

Data Description Cost

This part consists of encoding the matrix blocks.

- For each block B^A_{ij}, $i, j = 1, \ldots, k$ and B^F_{ij}, $i = 1, \ldots, k$, $j = 1, \ldots, l$, $n_1(B_{ij})$, that is the number of 1s in the sub-matrix, requires $\log^\star n_1(B_{ij})$ bits.
- Having encoded the summary information about the rectangular blocks, we next encode the actual blocks B_{ij}. We can calculate the density $P_{ij}(1)$ of 1s in B_{ij} using the description code above as $P_{ij}(1) = n_1(B_{ij})/n(B_{ij})$, where $n(B_{ij}) = n_1(B_{ij}) + n_0(B_{ij}) = r_i c_j$ for $\mathbf{F}$ blocks ($r_i r_j$ for $\mathbf{A}$ blocks), and $n_1(B_{ij})$ and $n_0(B_{ij})$ are the number of 1s and 0s in B_{ij}, respectively. Then, the number of bits required to encode each block using arithmetic coding is

$$E(B_{ij}) = -n_1(B_{ij}) \log_2(P_{ij}(1)) - n_0(B_{ij}) \log_2(P_{ij}(0)) = n(B_{ij}) H(P_{ij}(1)).$$

Total Encoding Cost (Length in bits)

We can now write the total encoding cost of the connectivity and feature matrices $\mathbf{A}$ and $\mathbf{F}$, with respect to a particular mapping (R, C):

$$L(\mathbf{A}, \mathbf{F}; R, C) = \log^\star n + \log^\star f + \log^\star k + \log^\star l - \sum_{i=1}^{k} r_i \log_2(\frac{r_i}{n}) - \sum_{j=1}^{l} c_j \log_2(\frac{c_j}{f})$$
$$+ \sum_{i=1}^{k}\sum_{j=1}^{l} \Big(\log^\star n_1(B_{ij}^F) + E(B_{ij}^F) \Big) + \sum_{i=1}^{k}\sum_{j=1}^{k} \Big(\log^\star n_1(B_{ij}^A) + E(B_{ij}^A) \Big).$$

10.1.4 Proposed Algorithm

The total encoding cost $L(\mathbf{A}, \mathbf{F}; R, C)$ can point out the best model with the minimum cost among many. It does not, however, tell us how to find the best model. [Johnson et al., 2004] show that column-reordering for a single matrix in order to reduce the total run length of the matrix is NP-hard, with a reduction from the Hamiltonian path problem. Our objective function is different, however, related in trying to reorder rows and columns to optimize block encoding. Thus, we conjecture that our problem is also hard. As a result, we resort to a monotonic iterative heuristic solution. Our experiments show that the proposed heuristic algorithm PICS performs quite well in practice for the real data sets used. The pseudo-code[1] of PICS is given in Algorithm 1.

Algorithm 1: PICS Algorithm

Input: $n \times n$ link matrix $\mathbf{A}$, $n \times f$ feature matrix $\mathbf{F}$
Output: A heuristic solution towards minimizing total encoding $L(\mathbf{A}, \mathbf{F}; R, C)$: number of row and column groups (k^*, l^*), associated mapping (R^*, C^*)

Set k^0=l^0=1 as we start with a single node and feature cluster.
Set $R^0 := \{1, 2, \ldots, n\} \rightarrow \{1, 1, \ldots, 1\}$
Set $C^0 := \{1, 2, \ldots, f\} \rightarrow \{1, 1, \ldots, 1\}$
Let T denote the outer iteration index. Set $T = 0$.
while *not converged* **do**
    $C^{T+1}, l^{T+1} :=$ `Split-FeatureGroup` (**F**, C^T, l^T)
    $(R^{T+1}, C^{T+1}) :=$ `Shuffle` (**A**, **F**, (R^T, C^{T+1}), (k^T, l^{T+1}))
    $R^{T+1}, k^{T+1} :=$ `Split-NodeGroup` (**A**, **F**, (R^{T+1}, C^{T+1}), (k^T, l^{T+1}))
    $(R^{T+1}, C^{T+1}) :=$ `Shuffle` (**A**, **F**, (R^{T+1}, C^{T+1}), (k^{T+1}, l^{T+1}))
    if $L(\mathbf{A}, \mathbf{F}; R^{T+1}, C^{T+1}) \geq L(\mathbf{A}, \mathbf{F}; R^T, C^T)$ **then**
        return (k^*, l^*)=(k^T, l^T), (R^*, C^*)=(R^T, C^T)
    else Set $T = T + 1$

[1] Source code of PICS: `www.cs.cmu.edu/~lakoglu/#code`

PICS starts with a single node and a single feature cluster (Lines 1-3) and iterates between two steps. At each iteration, it first tries to increase the number of feature clusters l by 1, by *splitting* the feature cluster with the maximum entropy per feature into two clusters (Line 6). Then, it *shuffles* the rows and columns of $\mathbf{A}$ and $\mathbf{F}$ such that the new ordering (mapping) yields a lower total encoding cost for the current number of clusters (k, l) (Line 7). Next, it tries to increase the number of node clusters k by 1, followed by another shuffle step (Lines 8 and 9, respectively). The algorithm halts when the total cost can not be reduced any further.

The implementation details together with further explanation for the procedures `Shuffle`, `Split-FeatureGroup`, and `Split-NodeGroup` follows.

'Split' procedures

Simply put, `Split-FeatureGroup` (similarly `Split-NodeGroup`) increases the number of attribute (node) groups by 1 by first finding the attribute (node) group with the maximum entropy per-column (row) (Line 1), and then moving those attributes (nodes) in that group whose removal reduce the per-column (row) entropy to the new group (Lines 2-6). If no column (row) can be moved, the procedures return the original mapping (Lines 7-8).

Procedure Split-FeatureGroup($\mathbf{F}$, C^T, l^T)

Input: $n \times f$ feature matrix $\mathbf{F}$, C^T, l^T
Output: C^{T+1}, l^{T+1}

1 Split the column group g with the maximum entropy per-column using Equation (10.1)

$$g := \arg\max_{1 \le j \le l} \frac{1}{c_j} \left(\sum_{i=1}^{k} n^F(B_{ij}) H(P_{ij}^F(1)) \right) \tag{10.1}$$

2 **for** *each column y in column group g* **do**
3 **if** *removal of y from g decreases the per-column entropy of g as in Equ.(10.2)*

$$\frac{1}{c_g - 1} \sum_{i=1}^{k} n^F(B'_{ig}) H(P'^F_{ig}(1)) < \frac{1}{c_g} \sum_{i=1}^{k} n^F(B_{ig}) H(P_{ig}^F(1)) \tag{10.2}$$

where B'_{ig} denotes the B_{ig} without column y, **then**
4 Place y into the new group: $C_y^{T+1} = l^T + 1$
5 Update $B_{ig}^F \; \forall i, 1 \le i \le k$.
6 **else** $C_y^{T+1} = C_y^T = g$
7 **if** *size of new feature-group* > 0 **then** $l^{T+1} = l^T + 1$
8 **else** $l^{T+1} = l^T$

Procedure Split-NodeGroup($\mathbf{A}$, $\mathbf{F}$, R^T, C^T, k^T, l^T)

Input: $n \times n$ connectivity matrix $\mathbf{A}$, $n \times f$ feature matrix $\mathbf{F}$, (R^T, C^T), (k^T, l^T)
Output: R^{T+1}, k^{T+1}

Split the row group g with the maximum entropy per-row using Equation (10.3)
for *each row* x *in row group* g **do**
  if *removal of* x *from* g *decreases the per-row entropy of* g *as in Equ.(10.4)* **then**
    Place x into the new group: $R_x^{T+1} = k^T + 1$
    Update $B_{gj}^F \; \forall j, 1 \leq j \leq l$ as well as B_{gj}^A and $B_{jg}^A \; \forall j, 1 \leq j \leq k$.
  else $R_x^{T+1} = R_x^T = g$
if *size of new node-group* > 0 **then** $k^{T+1} = k^T + 1$
else $k^{T+1} = k^T$

'Shuffle' procedure

When the number of node or attribute groups changes, `Shuffle` finds a lower-cost mapping by reassigning the nodes and attributes to the existing groups. It does so by first iterating over all the rows (nodes) (Lines 4-13) and then the columns (attributes) (Lines 14-20), assigning each row (column) to the node (attribute) group that minimizes its encoding cost (Lines 12 and 19, resp.). It repeats the same process until the total cost cannot be reduced any further (Lines 21-22).

Equations

$$g := \arg\max_{1 \leq i \leq k} \frac{1}{r_i} \left(\sum_{j=1}^{l} n^F(B_{ij}) H(P_{ij}^F(1)) + \sum_{j=1}^{k} n^A(B_{ij}) H(P_{ij}^A(1)) + \sum_{j=1}^{k} n^A(B_{ji}) H(P_{ji}^A(1)) \right) \tag{10.3}$$

$$\frac{1}{r_g - 1} \left(\sum_{j=1}^{l} n^F(B'_{gj}) H(P'^F_{gj}(1)) + \sum_{j=1}^{k} \left(n^A(B'_{gj}) H(P'^A_{gj}(1)) + n^A(B'_{jg}) H(P'^A_{jg}(1)) \right) \right)$$
$$< \frac{1}{r_g} \left(\sum_{j=1}^{l} n^F(B_{gj}) H(P_{gj}^F(1)) + \sum_{j=1}^{k} \left(n^A(B_{gj}) H(P_{gj}^A(1)) + n^A(B_{jg}) H(P_{jg}^A(1)) \right) \right) \tag{10.4}$$

where B'_{gj} denotes the B_{gj} without row x.

$$R_x^{t+1} := \arg\min_{1 \leq i \leq k} \Bigg\{ \sum_{j=1}^{l} \sum_{u=0}^{1} n_u^F(x^j) \log \frac{1}{P_{ij}^F(u)} \tag{10.5}$$
$$+ \sum_{j=1}^{k} \sum_{u=0}^{1} \left(n_u^A(x_r^j) \log \frac{1}{P_{ji}^A(u)} + n_u^A(x_c^j) \log \frac{1}{P_{ij}^A(u)} \right) + d_{xx} (\log P_{iR_x^t}^A(1) + \log P_{R_x^t i}^A(1) - \log P_{ii}^A(1))$$
$$+ (1 - d_{xx}) (\log P_{iR_x^t}^A(0) + \log P_{R_x^t i}^A(0) - \log P_{ii}^A(0)) \Bigg\}$$

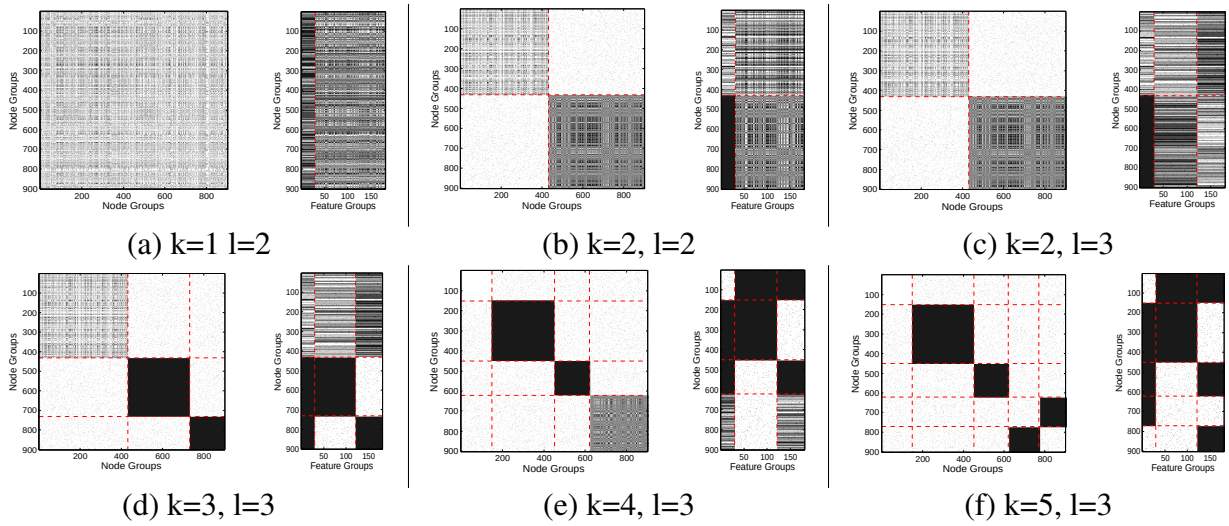
Procedure Shuffle($\mathbf{A}$, $\mathbf{F}$, R^T, C^T, k^T, l^T)

Input: $n \times n$ connectivity matrix $\mathbf{A}$, $n \times f$ feature matrix $\mathbf{F}$, (R^T, C^T), (k^T, l^T)
Output: (R^{T+1}, C^{T+1})

Let t denote the inner iteration index. Set $t = T$
Compute B_{ij}^t and P_{ij}^t with respect to (R^t, C^t, k^t, l^t)
while *not converged* **do**
    Shuffle rows: Fix column assignments C^t
    for *each row* x **do**
        Splice x in $\mathbf{F}$ into l^t parts (each corresponding to one of the column groups in $\mathbf{F}$) Denote them as $x^1, \ldots, x^{l^t}$.
        for *each of these parts* **do**
            Compute the number of 1s and 0s, that is, $n_u^F(x^j)$, $u = 0, 1$ and $j = 1, \ldots, l^t$
        Splice x in $\mathbf{A}$ and the corresponding column in $\mathbf{A}$ into k^t parts. Denote them as $x_r^1, \ldots, x_r^{k^t}$ and $x_c^1, \ldots, x_c^{k^t}$, respectively.
        for *each of these parts* **do**
            Compute the number of 1s and 0s, that is, $n_u^A(x_r^j)$ and $n_u^A(x_c^j)$, $u = 0, 1$ and $j = 1, \ldots, k^t$
    Assign each row (i.e., node) x into the node group R_x^{t+1} that yields the minimum encoding cost for x using Equation (10.5)
    Re-compute B_{ij}^{t+1} and P_{ij}^{t+1} with respect to (R^{t+1}, C^t, k^t, l^t)
    Shuffle columns: Fix row assignments R^{t+1} in $\mathbf{A}$ and $\mathbf{F}$, (also fixes the column assignment in $\mathbf{A}$, so we operate only on $\mathbf{F}$ here)
    for *each column* y **do**
        Splice y in $\mathbf{F}$ into k^t parts (each corresponding to one of the row groups in $\mathbf{F}$) Denote them as $y^1, \ldots, y^{k^t}$.
        for *each of these parts* **do**
            Compute the number of 1s and 0s, that is, $n_u^F(y^i)$, $u = 0, 1$ and $i = 1, \ldots, k^t$
    Assign each column (i.e., feature) y to the feature group C_y^{t+1} that yields the minimum encoding cost for y using Equation (10.6)
    Recompute B_{ij}^{t+1} and P_{ij}^{t+1} w.r.t. $(R^{t+1}, C^{t+1}, k^t, l^t)$
    if *there is no decrease in total cost* **then return** (R^t, C^t)
    else Set $t = t + 1$

where first two lines respectively denote the cost of shifting row x in $\mathbf{F}$, and the same row x and its corresponding column in $\mathbf{A}$ to a new group i. Last two lines account for the double-counting of cell d_{xx} in $\mathbf{A}$.

$$C_y^{t+1} := \arg\min_{1 \le j \le l} \left\{ \sum_{i=1}^{k} \sum_{u=0}^{1} n_u^F(x^i) \log \frac{1}{P_{ij}^F(u)} \right\} \tag{10.6}$$



(a) k=1 l=2 (b) k=2, l=2 (c) k=2, l=3

(d) k=3, l=3 (e) k=4, l=3 (f) k=5, l=3

Figure 10.3: *Step-by-step operation of* PICS *on the synthetic attributed graph in Figure 10.2.*

Notice that the PICS algorithm employs a top-down approach in finding the clusterings, where we start with a single node and a single attribute cluster and split into more clusters over iterations. In light of this, an interesting future direction is to study a bottom-up approach; starting with n node and f attribute clusters and merge into fewer clusters over iterations.

Synthetic Graph Example

In Figure 10.3, we show the step-by-step operation of PICS on the synthetic dataset in Fig. 10.2. In Fig. 10.3(a), we see the output $\mathbf{A}$ and $\mathbf{F}$ after `Split-FeatureGroup` and `Shuffle` (Steps 6 and 7) are executed on the original $\mathbf{A}$ and $\mathbf{F}$ shown in Fig. 10.2(a). Here, `Split-FeatureGroup` increases the number of feature groups to 2, and then `Shuffle` reorders the rows and columns in both matrices. Next, `Split-NodeGroup` increases the number of row groups to 2 (Step 8) and `Shuffle` reorders the rows and columns that yields a lower encoding cost (Step 9). This is also visually clear in Fig. 10.3(b). PICS repeats the same steps in Fig. 10.3(c) and (d). Notice that `Split-FeatureGroup` cannot increase the number of feature groups above 3, and thus `Shuffle` is called only after `Split-NodeGroup` in Fig. 10.3(e) and (f), after which `Split-NodeGroup` also stops finding new node groups for reduced cost and the algorithm converges.

Convergence

The stopping criterion for `Shuffle` is satisfied if and only if the total encoding cost cannot be reduced any further by the new ordering. Therefore, lines 7 and 9 in Algorithm 1 decreases the objective criterion $L(\mathbf{A}, \mathbf{F}; R, C)$. Since the objective criterion, i.e. the total encoding cost in bits, has the lower bound zero and the number of node and feature clusters have respective upper bounds n and f, the algorithm is guaranteed to converge.

Computational Complexity

The computationally most demanding component of PICS is the `Shuffle` procedure, which takes as input the $\mathbf{A}$ and $\mathbf{F}$ matrices and the number of clusters (k, l), and finds a new ordering of the rows and columns that gives a lower encoding cost. It achieves this goal by iterating between two steps: (1) shuffling the columns in $\mathbf{F}$, and (2) shuffling the rows in $\mathbf{A}$ and $\mathbf{F}$, simultaneously.

One iteration of step (1) above is $O(n_1(F) * l)$, where $n_1(F)$ denotes the number of non-zeros in $\mathbf{F}$, as we access each column and compute its number of non-zeros and consider l possible feature groups to place it into. Similarly, one iteration of step (2) is $O((n_1(F) + 2n_1(A)) * k)$. Therefore, the total complexity of `Shuffle` is $O([2n_1(A)k + n_1(F)(k + l)] * t)$, where t is the total number of inner iterations for `Shuffle`.

PICS calls `Shuffle` two times at each outer iteration T (Lines 7 and 9 in Algorithm 1). T is equal to $\max(k^*, l^*)$. Thus, the overall complexity of PICS is $O(\max(k^*, l^*) * [2n_1(A)k^* + n_1(F)(k^* + l^*)] * \hat{t})$, where $\hat{t}$ denotes the maximum number of inner iterations. Note that PICS scales linearly with respect to the total number of non-zeros in $\mathbf{A}$ and $\mathbf{F}$, the total number of `Shuffle` iterations $\hat{t}$ (typically $\hat{t} \ll n_1(A) + n_1(F)$), and quadratically with respect to the number of clusters (k^*, l^*), where k^* and l^* are usually small.

10.2 Experimental Results

10.2.1 Datasets

In our experiments we studied six real-world datasets from various domains which we describe next. A summary is given in Table 10.1. In each case we are able to use PICS to automatically discover interesting node and attribute structures, with further investigation providing explanations for their presence.

Dataset	n	f	$n_1(A) + n_1(F)$	**Connectivity and Attribute Description**
YOUTUBE	77381	30087	994542	User friendships and group memberships
TWITTER	9654	10000	81770	User 'mention's and hashtag usages
CALL	94	7	391	Phone calls and affiliations
DEVICE	94	7	5233	Bluetooth device scans and affiliations
POLBOOKS	92	2	840	Book co-purchases and politic inclinations
POLBLOGS	1490	2	20580	Blog citations and political inclinations

Table 10.1: Dataset summary. n: number of nodes, f: number of features, $n_1(A) + n_1(F)$: number of non-zeros (nnz), i.e. edges, in the connectivity matrix $\mathbf{A}$ plus the nnz in the feature matrix $\mathbf{F}$. See §10.2.1 for more details.

YOUTUBE data consists of users and YouTube-groups. The links represent user-user friendships and node attributes are the group memberships of each user. The data is compiled by [Mislove et al., 2007] in 2007.

TWITTER graph contains early-adopter users, with links representing the who-mentioned-whom interactions between them. The attributes are the hashtags (tokens starting with a #) that the users used in their messages (Tweets).

The CALL and DEVICE graphs are constructed using the Reality Mining data sets provided by the MIT Media Lab [Eagle et al., 2007]. The Reality Mining project was conducted in 2004-05 with 94 human subjects using mobile phones pre-installed with special software that recorded data. CALL is built using the call logs, and represents the phone-call interactions between the subjects. DEVICE is constructed using Bluetooth device scans; an edge from i to j indicates person i's device discovered person j's device within five meters proximity. Subjects included 67 students (*1st year grad*, *grad (not 1st year)*, *1st year undergrad*, and *undergrad (not 1st year)*), *faculty*, and *staff* working in the Media Lab, and 27 *business student*s at the university's Sloan School of Management.

POLBOOKS is a graph of books about U.S. politics published during 2004 presidential election [`http://www-personal.umich.edu/~mejn/netdata/`]; an edge from i to j indicates that book i was frequently co-purchased with book j by the same buyers. Similarly, POLBLOGS is a directed graph of hyperlinks between web-blogs on U.S. politics, compiled by [Adamic and Glance, 2005] in 2005. In both graphs, the nodes (books or blogs) are attributed as being liberal or conservative.

The YOUTUBE and POLBOOKS graphs are undirected and the rest are directed.

10.2.2 Clusters, bridges, and outliers

Next, as for our real datasets no ground truth (if any) for "true" clustering exists, we provide anecdotal and visual study.

YOUTUBE Dataset:

PICS finds node and feature clusters of various sizes in our largest dataset YOUTUBE, as shown in Figure 10.1, § 1. In Table 10.2, we show example YouTube-groups in major feature clusters; notice that these clusters can be human-labeled as 'porn', 'music', 'anime', and 'special interest'.

With respect to node clusters, PICS finds a very sparse group of 'familiar strangers' in cluster '1' with high feature coherence but scarce connections to the rest of the graph. These users belong to arbitrarily many and mostly same YouTube-groups labeled as 'A1', yet are not well-connected among themselves. Node clusters in the blue square labeled as '2' exhibit dense connectivity among each other as well as high feature homogeneity as seen in the 'anime' groups

labeled as ‘A2’. The node clusters in the blue square labeled as ‘4’ mostly belong to YouTube groups associated with ‘porn and music’ labeled as ‘A4’. Notice that the nodes in each of these clusters have quite similar connectivity to the graph. The node cluster ‘3’ contains nodes with connections across many clusters, behaving like ‘bridges’. They also mostly belong to the same YouTube-groups, labeled as ‘A3’. Finally, the small node clusters constitute the outliers, with arbitrarily many connections across clusters. All in all, by using PICS we are able to understand and summarize the YOUTUBE graph in a completely unsupervised fashion.

porn	**music**	**anime**	**interest**
hotmodels	poptastic	anime4ever	streetboarders
upskirt	raphiphop	narutoholics	modelcooks
lesbokiss	guitarsolos	crazyforanime	poetryandmusic
men4men	metallovers	animefreak1	mexicogrupero
gayestgay	classicalmusic	AnimeDaisuki	bodypainting
bootyshake	xtinaaguilera	SailorMoon	nflfans
sexysolo	heavymetal	Tsyukomi	chelseafcfans

Table 10.2: Examples of YouTube-groups in feature clusters found by PICS. See Figure 10.1.

TWITTER Dataset:

Our Twitter dataset consists of directed edges (i, j) between users if i directs a message to j at least three times. Attributes in our network are the hashtags used at least three times by a user. In Figure 10.4, PICS finds a group of ‘casual users’ in node cluster ‘1’ with few connections and few number of hastags used. Node cluster ‘2’ is the most prominent: this is a dense group of users (mostly tech bloggers) all from Italy who extensively mention each other but do not frequently message other users. They also use common and distinctive hashtags in Italian such as ‘#terremoto’ and ‘#partigi’. The two clusters in blue square (labeled as ‘4’) are the so-called ‘heavy-hitters’ who use overwhelmingly many different hashtags (labeled as ‘A4’). PICS also reveals a ‘core-periphery’ pattern within these users. The clusters in blue square labeled as ‘5’ form dense diagonal blocks with dense connections within. The smaller clusters also correspond to the so-called ‘bridges’, with many connections across clusters. Notice that the bridge-nodes are mostly mentioned *by* others but do not themselves mention others. Upon further inspection of users in this group, we discover Jeffrey Zeldman, a well known author, as well as Jack Dorsey and Ev Williams, the two founders of Twitter. We also observe that users in groups labeled as ‘3’ and ‘5’ have never used the hashtags in the rightmost feature cluster of Figure 10.4.

CALL Dataset:

PICS finds 3 major node clusters as shown in Figure 10.5. The first cluster corresponds to the group of *casual* subjects who make phone calls to only a few people. The next two clusters,

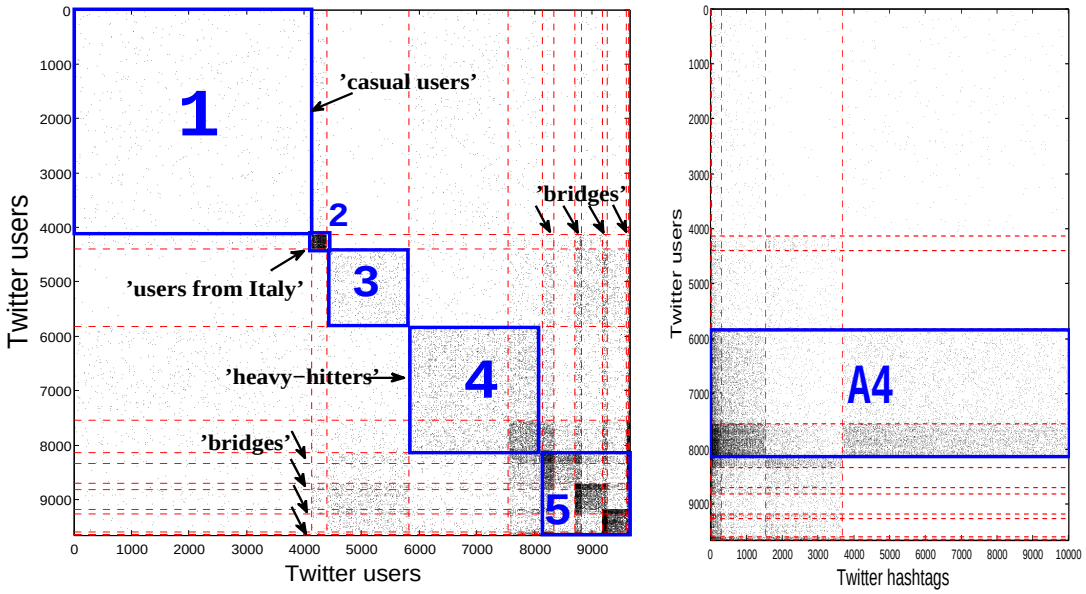


Figure 10.4: PICS on TWITTER finds a tight group of users from Italy and reveals groups of casual users, heavy-hitters, and bridges. Left: user-mentions-user adjacency matrix; right: users-to-hashtags feature matrix.

which are relatively densely connected, are a group of *business* and a group of *grad* students, respectively.

With respect to outliers, notice a cluster of size 1, an outlier subject who does not belong to any of the clusters. This subject, whose affiliation is not given in the dataset, does not make any outgoing calls but receives calls from almost everyone, which is presumably a call service center in the campus. Another outlier we can spot is a 1st year grad student who is in the same cluster with the business students in cluster 2; he/she neither calls nor gets called by other grad students but some business students.

With respect to bridges, we notice one grad student in cluster 3 who is in mutual contact with two business students. Notice that none of the other subjects has phone call interactions with the business students.

DEVICE Dataset:

In Figure 10.6, we observe 3 major dense clusters; clusters 1, 3 and 5. These three clusters involve mostly the *grad*, *business* and *undergrad* students, respectively. The reason these clusters form near-cliques, i.e. are almost fully connected, may be due to the Bluetooth scans occurring when these groups of students sit in the same classroom all within five meters.

In this dataset, the sparser clusters are also of interest. For instance, cluster 2 is a group of subjects whose devices seem to report far less scan results. This might be due to powered-off

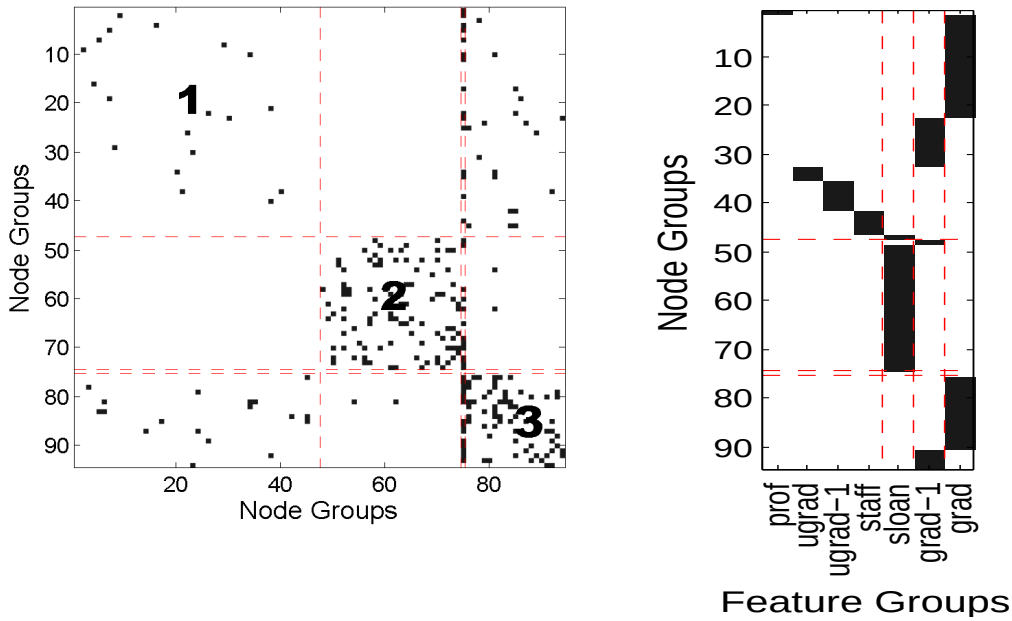


Figure 10.5: PICS *on* CALL. *Notice* 3 *major node groups of casual users, business and grad students, respectively as well as a group of size* 1*, receiving many calls, probably a call service center.*

devices or turned-off Bluetooth functionality.

In cluster 3 of business students, we also notice a column consisting of almost all zeros. This corresponds to a subject whose device can scan other devices, but somehow it does not get detected by others. This may be due to a malfunctioning device or missing data.

POLBOOKS Dataset:

The POLBOOKS dataset consists of liberal and conservative books which might be thought as two major clusters. In Figure 10.7, PICS gives more information about the cluster structure by finding 4 node clusters. The denser clusters (clusters 2 and 4) correspond to the "core" conservative and liberal books, respectively, which are often purchased together. Clusters 1 and 3 are then the corresponding "peripheral" books. Table 10.3 gives a list of several books in each "core". Notice that the "core" books do seem to lie at the two extremes of the political spectrum.

In cluster 1, we also observe 3 bridging books, namely *Bush at War*, *The Bushes: Portrait of a Dynasty*, and *Rise of the Vulcans: The History of Bush's War Cabinet*, which are co-purchased with both some liberal and conservative books. These books are human-labeled as conservative although they seem to have more of a historical perspective. Notice that the bridge nodes reside in the "periphery" rather than in the "core".

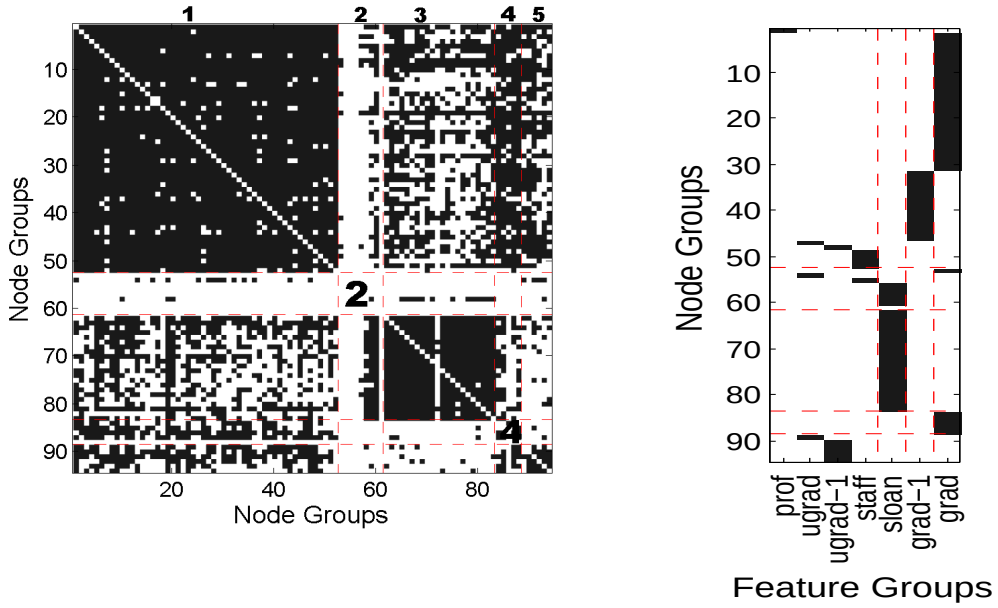


Figure 10.6: PICS on DEVICE finds 3 major dense node groups of grad, business and undergrad students, respectively, as well as anomalies, probably missing data.

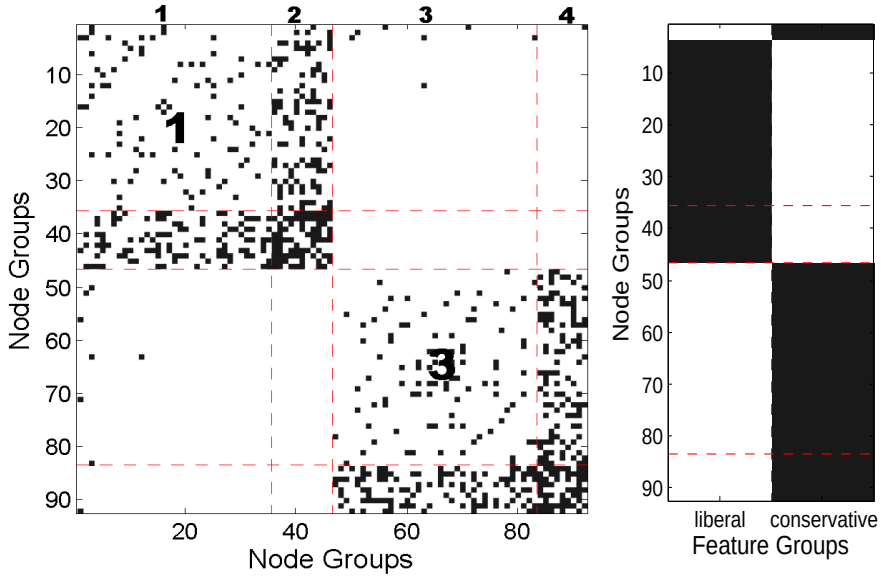


Figure 10.7: PICS on POLBOOKS finds 4 node groups corresponding to "core" and "peripheral" liberal and conservative books, as well as several bridge-books, with historical content and not extremely liberal or conservative.

POLBLOGS Dataset:

In Figure 10.8, we observe 7 major node clusters in POLBLOGS. The first and the largest cluster contains liberal and conservative blogs which do not have many citations. Clusters 2-4 consist

Liberal	Conservative
–Lies and the Lying Liars Who Tell Them: *A Fair and Balanced Look at the Right*	–Persecution: *How Liberals Are Waging War Against Christianity*
–Big Lies: *The Right-Wing Propaganda Machine and How It Distorts the Truth*	–Deliver Us from Evil: *Defeating Terrorism, Despotism, and Liberalism*
–The Lies of George W. Bush	–Tales from the Left Coast
–Dude, Where's My Country?	–A National Party No More
–Worse Than Watergate: *The Secret Presidency of George W. Bush*	–Bush Country: *How George W. Bush Became the First Great Leader of 21st Century*
–Thieves in High Places: *They've Stolen Our Country and It's Time to Take It Back*	–Losing Bin Laden: *How Bill Clinton's Failures Unleashed Global Terror*

Table 10.3: Examples of "core" liberal and conservative books.

of conservative and 5-7 consist of liberal blogs. Here, PICS seems to also reveal the "core" and "periphery" structure for the political blogs. In particular, cluster 3 is a core conservative group with a fanatic follower group (cluster 4), and a less fanatic follower group (cluster 2) of other conservative blogs, which often cite the blogs in cluster 3. Examples to "core" conservative blogs include *rightwingnews.com*, *georgewbush.com*, and *conservativeeyes.blogspot.com*. Similarly, cluster 6 is a core liberal group with cluster 7 being its more fanatic, and cluster 5 being its less fanatic follower group of other liberal blogs, with many citations to cluster 6. The blogs in the liberal "core" include *talkleft.com*, *liberaloasis.com*, and *democrats.org/blog*.

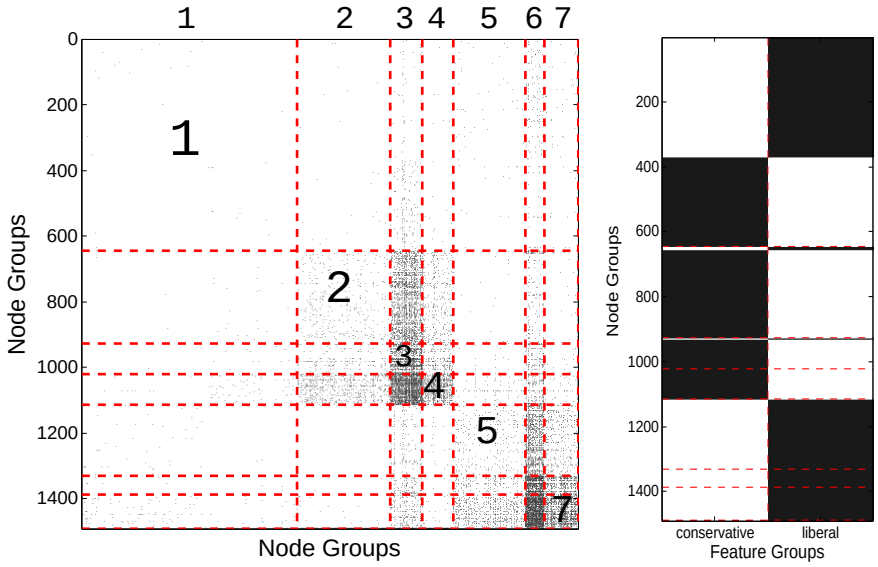
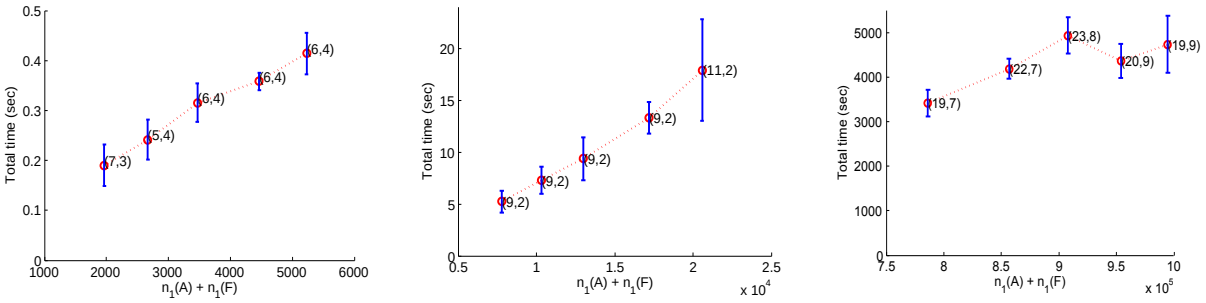


Figure 10.8: PICS on POLBLOGS. Notice the "core" conservative and liberal blogs (clusters 3 and 6), each with two sets of "peripheral" groups with many citations.



(a) DEVICE (b) POLBLOGS (c) YOUTUBE

Figure 10.9: Run time of PICS versus the total number of non-zeros (nnz) in $\mathbf{A}$ and $\mathbf{F}$ (averaged over 10 runs, bars depict $\pm$ one standard deviation). The numbers in parentheses denote the number of node and feature clusters (k^*, l^*) found. Notice that the run time grows linearly w.r.t. total nnz.

10.2.3 Scalability

In § 22 we theoretically showed that the computational complexity of PICS is linear in the total graph and attribute size. In this section, we also demonstrate the time complexity of PICS experimentally. Figure 13.5 shows the running time with respect to increasing total size for several datasets we studied (the total size is the total number of non-zeros (nnz) in the $\mathbf{A}$ and $\mathbf{F}$ matrices). Notice that the run time grows linearly with respect to the total nnz. (Recall that the run time also depends on the number of clusters found, hence the slight dip for YOUTUBE at total size 950K as fewer clusters (20 vs 23) are found.) Experiments were performed on a 4-CPU Intel 3.0GHz Xeon server with 16GB RAM. All code was written in Matlab.

10.3 Summary of contributions

In this chapter, we introduced PICS[2], a parameter-free method that finds cohesive clusters and hence helps us spot outliers in attributed graphs. The contributions of our work include:

- *Algorithm design:* We introduce a novel clustering model; PICS finds groups of nodes in an attributed graph with (1) *similar connectivity*, and (2) *attribute homogeneity*. By definition, clusters with similar connectivity include but are not limited to dense clusters. It also groups the node attributes into meaningful clusters. The nodes deviating from the discovered patterns correspond to bridge-nodes with connections across clusters or outlier-nodes that do not belong well to any cluster.
- *Parameter-free nature:* PICS is *fully automatic.* It works without any user-specified input, such as the number of clusters, choice of density or similarity functions and thresholds. To the best of our knowledge, no other work has been proposed to mine attributed graphs in a parameter-free fashion.
- *Scalability:* The run time of the proposed algorithms grows *linearly* with respect to the total graph and attribute size.
- *Effectiveness:* We show that PICS discovers quality clusters, bridges and outliers in diverse real-world datasets including YouTube and Twitter.

There are a number of interesting future directions for PICS, such as extending the algorithms to time-evolving attributed graphs by dynamically updating the mapping, and to more general cases such as soft-clustering, where the node, and similarly attribute, clusters may overlap.

[2]Source code of PICS: `www.cs.cmu.edu/~lakoglu/#code`

Chapter 11

Anomalies in Categoric-attributed Data

PROBLEM STATEMENT: *Given a database with categorical attributes, how can we find meaningful patterns that succinctly describe the data and spot outliers?*

In this chapter, we address the problem of anomaly detection in multi-dimensional databases using pattern-based compression. Compression-based techniques have been explored mostly in communications theory, for reducing transmission cost and increased throughput as well as in the field of databases, for reduced storage cost and increased query performance. Here, we improve over recent work that identified compression as a natural tool for spotting anomalies [Smets and Vreeken, 2011]. Simply put, we define the *norm* by the patterns that compress the data well. Then, any data point that can not be compressed well is said not to comply with the norm, and thus can be flagged as *abnormal*.

The heart of our method, which we call COMPREX, is to use a collection of dictionaries to encode a given database. We exploit correlations between the features in the database, grouping features with high information gain together, and build dictionaries (a.k.a. look-up tables or code tables [Vreeken et al., 2011]) for each group of strongly interacting features.

Informally, these dictionaries capture the data distribution in terms of patterns; the more often a pattern occurs, the shorter its encoded length. The goal is to find the optimal set of dictionaries, those yield the minimal lossless compression, and then spot the tuples with long encoded lengths; those described using relatively infrequent patterns.

Dictionary based compression for anomaly detection was first studied in [Smets and Vreeken, 2011], employing the KRIMP itemset-based compressor introduced by Siebes et al. [Siebes et al., 2006; Vreeken et al., 2011]. Besides high performance, it allows for characterization: one can easily inspect how tuples are encoded, and hence why one is deemed an anomaly. Where KRIMP builds a *single* code table, we use multiple code tables to describe the data. As such, our method can better exploit correlations between *groups* of features, as by doing away with uncorrelated features it can locally assign codes more efficiently—leading to both better compression and higher accuracy.

Furthermore, unlike KRIMP, we build code tables directly. That is, instead of filtering very large collections of pre-mined candidate patterns, we construct our code tables in a bottom-up fashion. We iteratively join those feature groups by which most compression can be gained, having only to consider patterns over the two groups. As such, COMPREX is much more efficient, with running time growing only linearly with number of features and database size.

Most importantly, by not requiring the user to provide a collection of candidate itemsets, nor a minimal support threshold, COMPREX is parameter free in both theory and practice. We employ the Minimum Description Length (MDL) principle to automatically decide the number of feature groups, which features to group, what patterns to include in the corresponding code tables, as well as to point out anomalies. Most existing anomaly detection methods, on the other hand, have several parameters, such as the choice of a similarity function, density or distance thresholds, number of nearest neighbors, etc.

In a nutshell, we improve over the state of the art by encoding data using multiple code tables, instead of one—allowing us to better grasp strongly interacting features. Moreover, we build our models directly from data—avoiding the expensive step of mining and filtering large collections of candidate patterns.

Experiments show that the resulting models obtain high performance in anomaly detection; improving greatly over the state of the art for categorical data. It is more generally applicable, though; even after discretization it matches the state of the art for numerical data, and correctly identifies anomalies both in translated large graphs and image data.

11.1 COMPREX: Method Description

We first describe how dictionary-based compression works, where we introduce the preliminaries and notation. Next, we present our method and the proposed algorithms.

11.1.1 Dictionary-based Compression

Notation and Preliminaries

In this work we consider categorical databases. A database D is a bag of n tuples over a set of m categorical features $\mathcal{F} = \{f_1, \ldots, f_m\}$. Each feature $f \in \mathcal{F}$ has a domain $dom(f)$ of possible values $\{v_1, v_2, \ldots\}$. The number of values $v \in dom(f)$ is the *arity* of f, i.e. $arity(f) = |dom(f)| \in \mathbb{N}$.

The domains are distinct between features. That is, $dom(f_i) \cap dom(f_j) = \emptyset, \forall i \neq j$. The domain of a feature set $F \subseteq \mathcal{F}$ is the Cartesian product of the domains of the individual features $f \in F$, i.e., $dom(F) = \prod_{f \in F} dom(f)$.

A database D is simply a collection of n tuples, where each tuple t is a vector of length m containing a value for each feature in $\mathcal{F}$. As such, D can also be regarded as a n-by-m matrix, where the possible values in a column i are determined by $dom(f_i)$.

An *item* is a feature-value pair $(f = v)$, with $f \subseteq \mathcal{F}$, and $v \in dom(f)$. A itemset is then a pair $(F = v)$, for a set of features $F \subseteq \mathcal{F}$, and $v \in dom(F)$ is a vector of length $|F|$. We typically refer to an itemset as a *pattern*.

A tuple t is said to contain a pattern $(F = v)$, denoted as $p(F = v) \subseteq t$ (or $p \subseteq t$ for short), if for all features $f \in F$, $t_f = v_f$ holds. The support of a pattern $(F = v)$ is the number of tuples in D that contain it: $supp(F = v) = |\{t \in D \mid (F = v) \subseteq t\}|$. Its frequency is then $freq(F = v) = supp(F = v)/|D|$.

Finally, the entropy of a feature set F is defined as

$$H(F) = -\sum_{v \in dom(F)} freq(F = v) \log freq(F = v) \ .$$

All logarithms are to base 2, and by convention, $0 \log 0 = 0$.

In this work we take a pattern based approach to encode a given database, which we explain in detail next.

Data encoding

As models, we will use code tables. A code table is a simple two column table. The first column contains *pattern*s, which are ordered descending (1) first by length and (2) second by support. The second column contains the code words $code(p)$ corresponding to each pattern p. An illustrative database with 6 tuples and an example code table for the database is illustrated in Table 11.1.

Data	*Code Table*			
$f_1 f_2 f_3$	$p(F = v)$	$code(p)$	$usage(p)$	$L(code(p))$
a b x	a b x	0	4	1 bit
a b x	a c	10	2	2 bits
a b x	x	110	1	3 bits
a b x	y	111	1	3 bits
a c x				
a c y				

Table 11.1: An illustrative database D and an example code table CT for a set of three features, F=$\{f_1, f_2, f_3\}$.

The code words in the second column of a code table CT are not important: their lengths are. The length of a code word for a pattern depends on the database we want to compress. Intuitively,

the more often a pattern occurs in the database, the shorter its code should be. The *usage* of a pattern $p \in CT$ is the number of tuples $t \in D$ which contain p in their *cover*, i.e. have the code of p in their encoding.

The encoding of a tuple t, given a CT, works as follows: the patterns in the first column are scanned in their predefined order to find the first pattern p for which $p \subseteq t$. p is then said to be *used* in the *cover* of t , and the corresponding code word for p in the second column becomes part of the encoding of t. If $t \setminus p \neq \emptyset$, the encoding continues with $t \setminus p$ until t is completely covered, which yields a unique set of patterns that form the *cover* of t.

Given the usages of the patterns in a CT, we can compute the lengths of the code words for the optimal prefix code [Grünwald, 2007]. The Shannon entropy gives the length for the optimal prefix code for p as

$$L(code(p) \mid CT) = -\log\big(\frac{\textit{usage}(p)}{\sum_{p' \in CT} \textit{usage}(p')}\big).$$

The number of bits to encode a tuple t is simply the sum of the code lengths of the patterns in its cover, that is,

$$L(t \mid CT) = \sum_{p \in \textit{cover}(t)} L(code(p) \mid CT).$$

The total length of the encoded database is then the sum of encoded data tuple lengths,

$$L(D \mid CT) = \sum_{t \in D} L(t \mid CT).$$

Model encoding

To find the MDL-optimal compressor, we also need to determine the encoded size of the model, the code table in our case. Clearly, the size of the second column in a given code table CT that contains the prefix code words $code(p)$ is trivial; it is simply the sum of their lengths. For the size of the first column, we need to consider all the singleton items $\mathcal{I}$ contained in the patterns, i.e., $\mathcal{I} = \bigcup_{f \in \mathcal{F}} dom(f)$.

For encoding the patterns in the left hand column we again use an optimal prefix code. We first compute the frequency of their appearance in the first column, and then by Shannon entropy calculate the optimal length of these codes. Specifically, the encoding of the first column in a code table requires $cH(P)$ bits, where c is the total count of singleton items in the patterns $p \in CT$, $H(.)$ denotes the Shannon entropy function, and P is a multinomial random variable with the probability $p_i = \frac{r_i}{c}$ in which r_i is the number of occurrences of singleton item $i \in \mathcal{I}$ in the first column (Note that for the actual items, one could add an ASCII table providing the matching from the prefix codes to the original names. Since all such tables are over $\mathcal{I}$, this only adds an additive constant to the total cost). All in all,

$$L(CT) = \sum_{p \in CT} L(code(p) \mid CT) + \sum_{i \in \mathcal{I}} -r_i \log(p_i).$$

11.1.2 Compression with Sets of Code Tables: the Theory

In the previous section, we showed how to encode a database D using a *single* code table CT. In fact, this is the approach introduced in [Siebes et al., 2006] to compress transaction databases, employing frequent itemset mining to generate the candidate patterns for the code table. Here, we do not regard transaction data, but regular relational data, where tuples are data points in a multi-dimensional categorical feature space. In this space, some groups of features may be highly correlated; and hence may be compressed well together.

As a result, we can *improve* by using multiple code tables, as we can then exploit correlations, and build a separate CT for each highly correlated group of features. As such, using *multiple*, that is a *set* of code tables, and mining these efficiently, are two of the main contributions of our work. Next, we formally introduce the concept of feature partitioning, and give our problem statement.

Definition 7 *A feature partitioning $\mathcal{P} = \{F_1, \ldots, F_k\}$ of a set of features $\mathcal{F}$ is a grouping of $\mathcal{F}$, for which (1) each partition contains one or more features: $\forall F_i \in \mathcal{P} : F_i \neq \emptyset$, (2) all partitions are pairwise disjoint: $\forall i \neq j : F_i \cap F_j = \emptyset$, and (3) every feature belongs to a partition: $\bigcup F_i = \mathcal{F}$.*

Formal Problem Statement 1 *Given a set of n data tuples in D over a set of m features in $\mathcal{F}$, find a partitioning $\mathcal{P} : \{F_1, F_2, \ldots, F_k\}$ of $\mathcal{F}$ and a set of associated code tables $\mathcal{CT} : \{CT_{F_1}, CT_{F_2}, \ldots, CT_{F_k}\}$, such that the total compression cost in bits given below is minimized.*

$$L(\mathcal{P}, \mathcal{CT}, D) = L(\mathcal{P}) + \sum_{F \in \mathcal{P}} L(\pi_F(D)|CT_F) + \sum_{F \in \mathcal{P}} L(CT_F) \ ,$$

in which $\pi_{F_i}(D)$ denotes the projection of D on feature subspace F_i.

Note that the number of features m and the number of tuples n are fixed over all models we consider for a D, and hence are a constant additive that we can safely ignore.

Partition encoding

The first term of $L(\mathcal{P}, \mathcal{CT}, D)$ denotes the length of encoding the partitioning, which consists of two parts; encoding (a) the number of partitions and (b) the features per partition.

(a) Encoding the number of partitions: First, we need to encode k, the number of partitions and code tables. For this, we use the MDL-optimal encoding of an integer [Grünwald, 2007]. The cost for encoding an integer value k is $L^0(k) = \log^\star(k) + \log(c)$, with $c = \sum 2^{-\log(n)} \approx 2.865064$,, and $\log^\star(k) = \log(k) + \log\log(k) + \cdots$ sums over all positive terms. Note that as $\log(c)$ is constant for all models, we ignore it.

(b) Encoding the features per partition: Then, for each feature we have to describe to which partition it belongs. This we do using $m \log(k)$ bits.

In summary, $L(\mathcal{P}) = \log^\star(k) + m \log(k)$.

Data encoding

The second term of $L(\mathcal{P}, \mathcal{CT}, D)$ denotes the cost of encoding the data with the given set of code tables. To do so, each tuple is partitioned according to $\mathcal{P}$, and encoded using the optimal codes in the corresponding code tables following the procedure described in §11.1.1.

Model encoding

The last term of $L(\mathcal{P}, \mathcal{CT}, D)$ denotes the model cost, that is the total length of encoding the code tables. Each code table is encoded following the procedure described in §11.1.1.

Note that the number of feature groups k is not a parameter of our method but rather is determined by MDL. In particular, MDL ensures that we will not have two separate code tables for a pair of highly correlated feature groups as it would yield lower data cost to encode them together. On the other hand, combining feature groups may yield larger code tables, that is higher model cost, which may not compensate for the savings from the data cost. In other words, we group features for which the total encoding cost $L(\mathcal{P}, \mathcal{CT}, D)$ is reduced. Basically, we employ MDL to guide us both in finding which features to group, and in choosing the number of groups to have.

11.1.3 Mining Sets of Code Tables: the Algorithm

Having defined the cost function as $L(\mathcal{P}, \mathcal{CT}, D)$, we need an algorithm to search for the best set of code tables $\mathcal{CT}$ for the optimal vertical partitioning $\mathcal{P}$ of the data such that the total encoded size $L(\mathcal{P}, \mathcal{CT}, D)$ is minimized.

The search space for finding the best code table for a given set of features, yet alone for finding the optimal partitioning of features, however, is quite large. Finding the optimal code table for a set of $|F_i|$ features involves finding all the possible patterns with different value combinations up to length $|F_i|$ and choosing a subset of those patterns that would yield the minimum total cost on the database induced on F_i. Furthermore, the number of possible partitioning of a set of m features is the well-known Bell number B_m.

While the search space is prohibitively large, it neither has a structure nor exhibits monotonicity properties which could help us in pruning. As a result, we resort to heuristics. Our approach builds the set of code tables in a greedy bottom-up, iterative fashion. We give the pseudo-code as Algorithm 2, and explain it in more detail in the following.

COMPREX starts with a partitioning $\mathcal{P}$ in which each feature belongs to its own group (1), and separate, elementary code tables CT_i for each feature f_i associated with the feature sets (2).

Definition 8 *An elementary code table CT encodes a database D induced on a single feature $f \in \mathcal{F}$. The patterns $p \in CT$ consist of all length-1 unique items $v_1, \ldots, v_{arity(f)}$ in $dom(f)$. Finally, $usage(p \in CT) = freq(f = v)$.*

Typically, some features of the data will be more strongly correlated than others, e.g., the age of a car and its fuel efficiency, or the weather temperature and flu outbreaks. In such cases, it will be worthwhile to represent features for which the correlation is 'high enough' together within one CT, as we can then exploit correlation to save bits.

More formally, we know from Information Theory that given two (sets of) random variables (in our case feature groups) F_i and F_j, the average number of bits we can save when compressing F_i and F_j together instead of separately, is known as their Information Gain. That is,

$$IG(F_i, F_j) = H(F_i) + H(F_j) - H(F_i, F_j) \geq 0,$$

In fact, the IG of two (sets of) variables is always non-negative (zero when the variables are independent from each other), which implies that the data cost would be the smallest if all the features were represented by a single CT. On the other hand, our objective function also includes the compression cost of the CTs.

Clearly, having one large CT over many (possibly uncorrelated) features will typically require more bits in model cost than it saves in data cost. Therefore, we can use IG to point out good merge candidates, subsequently employing MDL to decide whether the total cost is reduced, and hence, whether to approve the merge or not.

The first step then is to compute the IG matrix for all pairs of the current feature sets, which is a non-negative and symmetric matrix (3). Let $|F_i|$ denote the cardinality, i.e. the number of features in the feature set F_i. We sort the pairs of feature sets in decreasing order of IG-per-feature, i.e. normalized by their total cardinality, and start *outer* iterations to go over these pairs as the candidate CTs to be merged, say CT_i and CT_j (5). The starting cost $cost_{init}$ is set to the total cost with the initial set of CTs (6).

The construction of the new $CT_{i|j}$ then works as follows: we put all the existing patterns $p_{i,1}, \ldots, p_{i,n_i}$ and $p_{j,1}, \ldots, p_{j,n_j}$ from both CTs into the new CT (7). Following the convention, they are sorted first by length (from longer to shorter) and second by usage (from higher to lower) (8). Candidate partitioning $\hat{\mathcal{P}}$ is built by dropping feature sets F_i and F_j from $\mathcal{P}$ and including the concatenated set $F_{i|j}$ (9). Similarly, we build a temporary code table set $\hat{\mathcal{CT}}$ by dropping the candidate tables CT_i and CT_j and adding the new $\hat{CT}_{i|j}$ (10).

Next, we find all the unique rows of the database induced on $F_{i|j}$ (11). These patterns of length ($|F_i|$+$|F_j|$) are sorted in decreasing order of their occurrence in the database and constitute the candidates to be inserted into the new CT. Let $p_{i|j,1}, \ldots, p_{i|j,n_{i|j}}$ denote these patterns of the combined feature set $F_{i|j}$ in their sorted order of frequency. In our *inner* iterations (12), we insert these one-by-one (13), update (i.e. decrease) the usages of the existing overlapping patterns (14), remove those patterns whose usage drops to zero (15), recompute the code word lengths with the updated usages (16) and compute the total cost after each insertion. If the total cost is reduced, we store the candidate partitioning $\hat{\mathcal{P}}$ and the associated set of code tables $\hat{\mathcal{CT}}$ (18), otherwise we continue insertions (from 12) with the next candidate patterns for possible future cost reduction.

In the outer iterations, if total cost is reduced the IG between the new feature set $F_{i|j}$ and the rest are computed (20). Otherwise the merge is rejected and the candidates $\hat{\mathcal{P}}$ and $\hat{\mathcal{CT}}$ are discarded (22). Next the algorithm continues to search for future merges, and the search terminates when there are no more pairs of feature sets that can be merged for reduced cost.

Algorithm 2: COMPREX Algorithm

Input: Database D with n tuples and m (categorical) features
Output: A heuristic solution to Problem Statement 1: a feature partitioning $\mathcal{P} : \{F_1, \ldots, F_k\}$, associated set of code tables $\mathcal{CT} : \{CT_1, \ldots, CT_k\}$, and total encoded size $L(\mathcal{P}, \mathcal{CT}, D)$

1 $\mathcal{P} \leftarrow \{F_1, \ldots, F_m\}$, $F_i = \{f_i\}, 1 \leq i \leq m$
2 $\mathcal{CT} \leftarrow \{CT_1, \ldots, CT_m\}$, where each CT_i is elementary
3 Compute IG between $\forall (F_i, F_j) \in \mathcal{P}$, $i > j$
4 **while** *no convergence, i.e. more merges* **do**
5 **for** *each* $(F_i, F_j) \in \mathcal{P}$ *in decreasing normalized* IG **do**
6 Compute $cost_{init} \leftarrow L(\mathcal{P}, \mathcal{CT}, D)$
7 Put patterns $p \in CT_i$ and $p \in CT_j$ into a new $\hat{CT}_{i|j}$
8 Sort $p \in \hat{CT}_{i|j}$, (1) by length and (2) by usage
9 $\hat{\mathcal{P}} \leftarrow \mathcal{P} \backslash (F_i \cup F_j) \cup F_{i|j}$
10 $\hat{\mathcal{CT}} \leftarrow \mathcal{CT} \backslash (CT_i \cup CT_j) \cup \hat{CT}_{i|j}$
11 Find unique rows (candidate patterns) $p_{i|j}$ in $D_{F_{i|j}}$
12 **for** *each unique row* $p_{i|j,x}$ *in decreasing frequency* **do**
13 Insert $p_{i|j,x}$ to new $\hat{CT}_{i|j}$
14 Decrease usages of overlapping patterns $p \in \hat{CT}_{i|j}$ and $p \in cover(p_{i|j,x})$ by $freq(p_{i|j,x})$
15 Remove patterns $p \in \hat{CT}_{i|j}$ with $usage(p)$=0
16 Recompute $L(code(p \in \hat{CT}_{i|j}))$ with new $usage$s
17 **if** $L(\hat{\mathcal{P}}, \hat{\mathcal{CT}}, D) < L(\mathcal{P}, \mathcal{CT}, D)$ **then**
18 $\mathcal{P} \leftarrow \hat{\mathcal{P}}$ and $\mathcal{CT} \leftarrow \hat{\mathcal{CT}}$
19 **if** $L(\mathcal{P}, \mathcal{CT}, D) < cost_{init}$ **then**
20 Compute IG between $F_{i|j}$ and $\forall F_x \in \mathcal{P}, F_x \neq F_{i|j}$
21 **else**
22 Discard $\hat{\mathcal{P}}$ and $\hat{\mathcal{CT}}$

11.1.4 Computational Speedup and Complexity

In Algorithm 2, computationally most demanding steps are (1) finding all the unique rows in the database under a particular feature subspace when two feature sets are to be merged (11) and (2)

after each insertion of a new pattern to the code table, finding the existing overlapping patterns the usages of which to be decreased (14).

With a naive implementation, step (1) is performed on the fly scanning the entire database once and possibly using many linear scans and comparisons over the unique rows found so far in the process. Furthermore, step (2) requires a linear scan over the current patterns in the new code table $\hat{CT}_{i|j}$ for each new insertion. The total computational complexity of these linear searches depends on the database, however, with the outer and inner iteration levels (Lines 5 and 12, respectively), those may become computationally infeasible for very large databases.

We improve with a simple design choice. Instead of an integer vector of $usage$ per pattern in CT, we have a sparse matrix C for patterns versus data points, the binary entries c_{ji} indicating whether data tuple i contains pattern j in its $cover$. Note that the row sum of the C matrix gives the $usage$s of the patterns. In such a setting, step (1) (mining for candidate patterns) works as follows: Let F_i and F_j denote the feature sets to be merged. Let C_i denote the $n_i \times n$ patterns versus data tuples matrix for code table CT_i and similarly C_j denote the $n_j \times n$ matrix for CT_j, in which n_i and n_j respectively denote the number of patterns each table has. We obtain the usages for the new candidate patterns (merged unique rows) under the merged feature subspace $F_{i|j}$ by multiplying C_i and C_j^T into a $n_i \times n_j$ matrix U, which takes $\mathcal{O}(n_i n n_j)$.

Next, we sort the merged patterns in decreasing order by their usage $U_{x,y}$ and insert them to $\hat{CT}_{i|j}$ one-by-one. Note that we exploit the existing patterns in the code tables to be merged, rather than finding all the unique rows of the database. This way, we quickly identify good frequent candidates and consider only $n_i n_j$ of them. Since $n_i, n_j \ll n$, we practically reduce the number of inner iterations (12) to a constant.

From here, step (2) (insertions) works as follows: Let $U_{x,y}$ denote the highest usage associated with the merged pattern $p_i(x)|p_j(y)$. The insertion of $p_i(x)|p_j(y)$ into the code table simply means the addition of a new row to the $C_{i|j}$ matrix ($C_{i|j}$ is obtained by concatenating the rows of C_i and C_j and reordering its rows (1) by length and (2) by usage of the patterns it initially contains). The new row is then the dot product (i.e. logical AND) of row x in C_i and the row y in C_j (data tuples which contain *both* $p_i(x)$ and $p_j(y)$ in their cover). Moreover, we decrease the usages of the merged patterns $p_i(x)$ and $p_j(y)$ by subtracting the new row from both of their corresponding rows. All these updates are $\mathcal{O}(n)$ operations.

All in all, the inner loop (starting in 12) goes over $n_i n_j$(≈constant) number of candidate patterns and each insertion takes $\mathcal{O}(n)$. Therefore, the inner loop takes $\mathcal{O}(n)$.

Next, we consider the outer loop (starting in 5), which tries to merge pairs of code tables. In the worst case we get $\mathcal{O}(m^2)$ trials when all merges are discarded. As a result the worst case complexity of COMPREX becomes $\mathcal{O}(m^2 n)$. In practice, however, many features are correlated and we obtain a merge at almost every step, yielding approximately linear complexity in both data size and dimension.

11.1.5 Anomaly Detection

Compression based techniques are naturally suited for anomaly and rare instance detection. Next we describe how we exploit our dictionary based compression framework for this task.

In a given code table, the patterns with short code words, that is those that have high usage, represent the patterns in the data that can effectively compress the majority of the data points. In other words, they capture the trends summarizing the *norm* in the data. On the other hand, the patterns with longer code words are rarely used and thus encode the sparse regions in the data. Consequently, the data tuples can be scored by their encoding cost for anomalousness.

More formally, given a set of code tables $CT_1, \ldots, CT_k$ found by COMPREX, each data tuple $t \in D$ can be encoded by one or more code words from each CT_i, $i = \{1, \ldots, k\}$. The corresponding patterns constitute the *cover* of t as discussed in §11.1.1. The encoding cost of t is then considered as its *anomalousness* score; the higher the compression cost, the more likely it is "to arouse suspicion that it was generated by a different mechanism" [Hawkins, 1980].

$$\textit{score(t)} \quad = \quad L(t|CT) = \sum_{F \in \mathcal{P}} L(\pi_F(t)|CT_F) = \sum_{F \in \mathcal{P}} \sum_{p \in cover(\pi_F(t))} L(code(p)|CT_F)$$

Having computed the compression costs, one can sort them and report the top k data points with the highest scores as possible anomalies. An alternative way [Smets and Vreeken, 2011] is to determine a decision threshold θ and flag those points with a compression cost greater than θ as anomalies. One can use the Cantelli's Inequality [Grimmett and Stirzaker, 2001] which provides a well-founded way to determine a good value for the threshold θ for a given confidence level, that is, an upper bound for the false positive rate. For generality, in our experiments we show the accuracy at all decision thresholds.

11.2 Experimental Results

In our study, we experimented with many datasets from diverse domains, as shown in Tables 11.2 and 11.3. Other datasets we used include graph and satellite image datasets, which we will discuss later in the experiments.

We evaluated our method with respect to four criteria: (1) compression cost in bits, (2) running time, (3) detection accuracy, and (4) scalability. We also compared our results with two state of the art methods, KRIMP [Smets and Vreeken, 2011] and `LOF` [Breunig et al., 2000a]. KRIMP is also a compression based anomaly detection method, but uses a single code table to encode a dataset. It performs frequent itemset mining as a pre-processing step to generate candidate patterns for the code table. `LOF` is a density-based outlier detection method that computes the local-outlier-factor of data points with respect to their nearest neighbors. Neither of the methods are parameter-free, they respectively require the minimum support threshold and the number of neighbors to be considered.

11.2.1 Compression-cost and Running-time

One goal of our study is to develop a fast method that would model the *norm* of the data and hence give low compression cost in bits. In this section, we use both COMPREX and KRIMP for compressing our datasets and show the total cost in bits and the corresponding running times in Figure 11.1 (a) and (b). Note that the results for our largest datasets Enron, Connect-4, and Covertype are given with broken y-axes with values in millions for cost and thousands for time.

We note that COMPREX achieves very nice compression rates, outperforming KRIMP for all of the datasets, and providing up to 96% savings in bits (47% on average).

With respect to running time, we notice that for most of the (smaller) datasets, our running time is only slightly higher than that of KRIMP but still remains under 16 seconds. Even though for small datasets the run time of KRIMP is negligible, for datasets especially with large number of features, its running time increases significantly. The computationally most demanding part of KRIMP is the frequent itemset mining, making it less feasible for large and high dimensional categorical datasets. For example, for the Connect-4 dataset with 42 features and the Chess (king,rook vs. king,pawn) datasets with 36 features, KRIMP cannot finish within reasonable time due to very large candidate sets.

To alleviate this problem, KRIMP accepts a *minsup* parameter, which is the minimum number of occurrences of an itemset in the database to be considered as frequent. The higher the *minsup* is set, the fewer the extracted itemsets are. However, high *minsup* comes with a trade-off; the higher the *minsup*, the smaller the number of candidates, the smaller the search space and the worse the final code table approximates the optimal code table. In contrast, our method does not require any sorts of parameters and frequent itemset mining.

In our experiments, we find *closed* frequent itemsets with *minsup* set to 5000 and 500 for the Connect-4 and Chess (kr-kp) datasets, respectively. Even then, the running time of our method remains lower than that of KRIMP (see Fig. 11.1). On the other hand, the time required for frequent itemset mining also depends on the dataset characteristics. For example, we observe in Figure 11.1 that the running time of KRIMP on (larger) Mushroom is much smaller than that on the (smaller) Spect-heart dataset, with both having the same number of (22) features.

For our largest datasets (in terms of size and number of features) in Figure 11.1, notice that the running time of KRIMP is quite large (about 20 mins) for Enron and Connect-4, and (45 mins) for Covertype. Moreover, its compression cost is still higher than that our method provides. We conclude that our method becomes more advantageous especially for large datasets.

11.2.2 Detection Accuracy

Besides achieving high compression rate, we would also (if not more) want our method to be effective in spotting anomalies. In this section, we experiment with various types of data including transaction, graph and image databases.

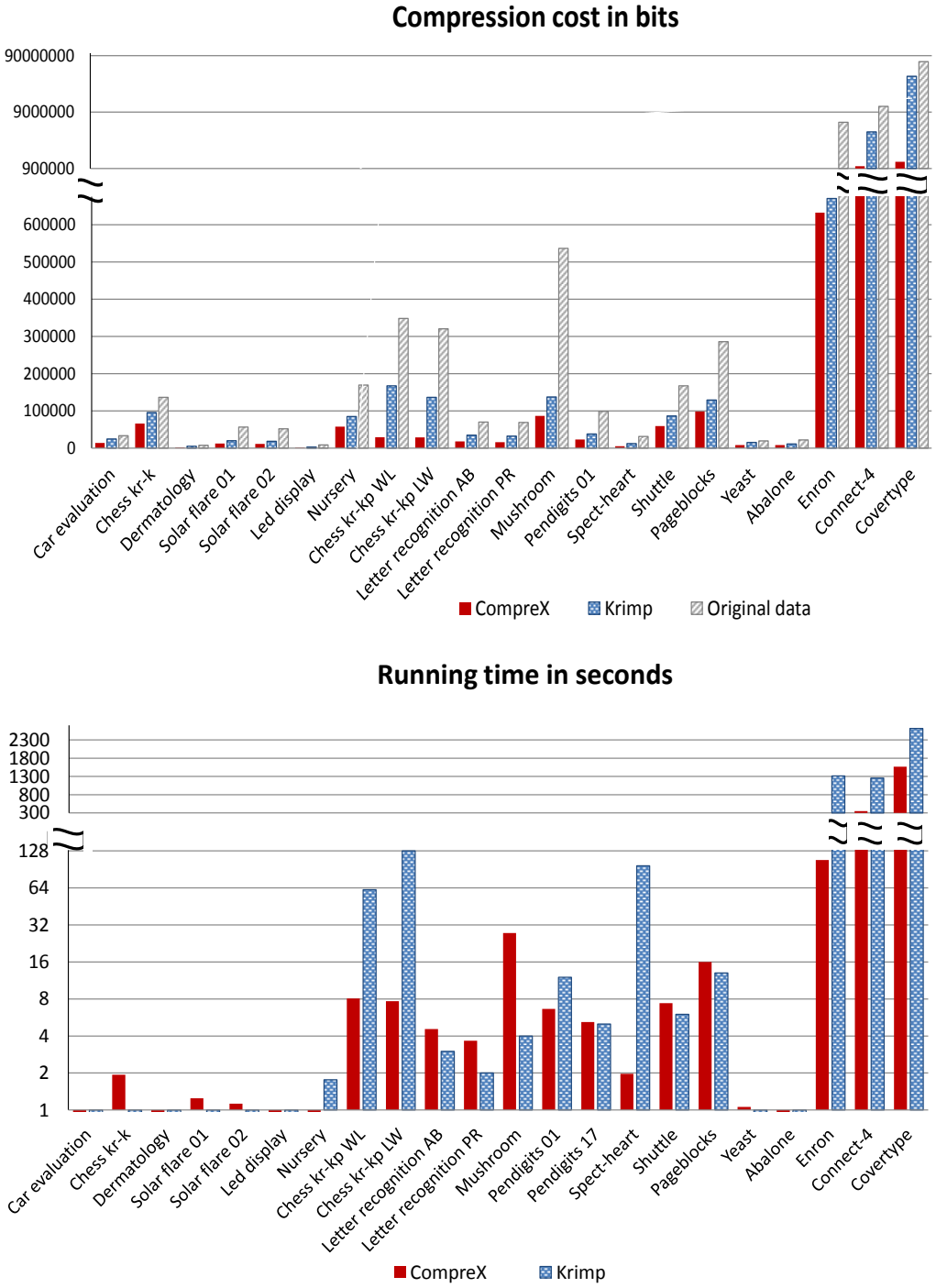


Figure 11.1: (top) Compression cost (in bits) when encoded by COMPREX vs. KRIMP. (bottom) Running time (in seconds) of COMPREX vs. KRIMP. For large datasets, extremely many frequent itemsets negatively affect the runtime for KRIMP.

COMPREX on categorical data

For measuring detection performance, we use two-class datasets. The number of data points from one class is significantly smaller than that from the other class. We call these classes as minority and the majority classes, respectively. The data points from the minority class are considered to be the "anomalies". Examples to the classes include poisonous vs. edible in Mushroom data, unaccountable vs. very good in Car data, and win vs. loss in Connect-4 data.

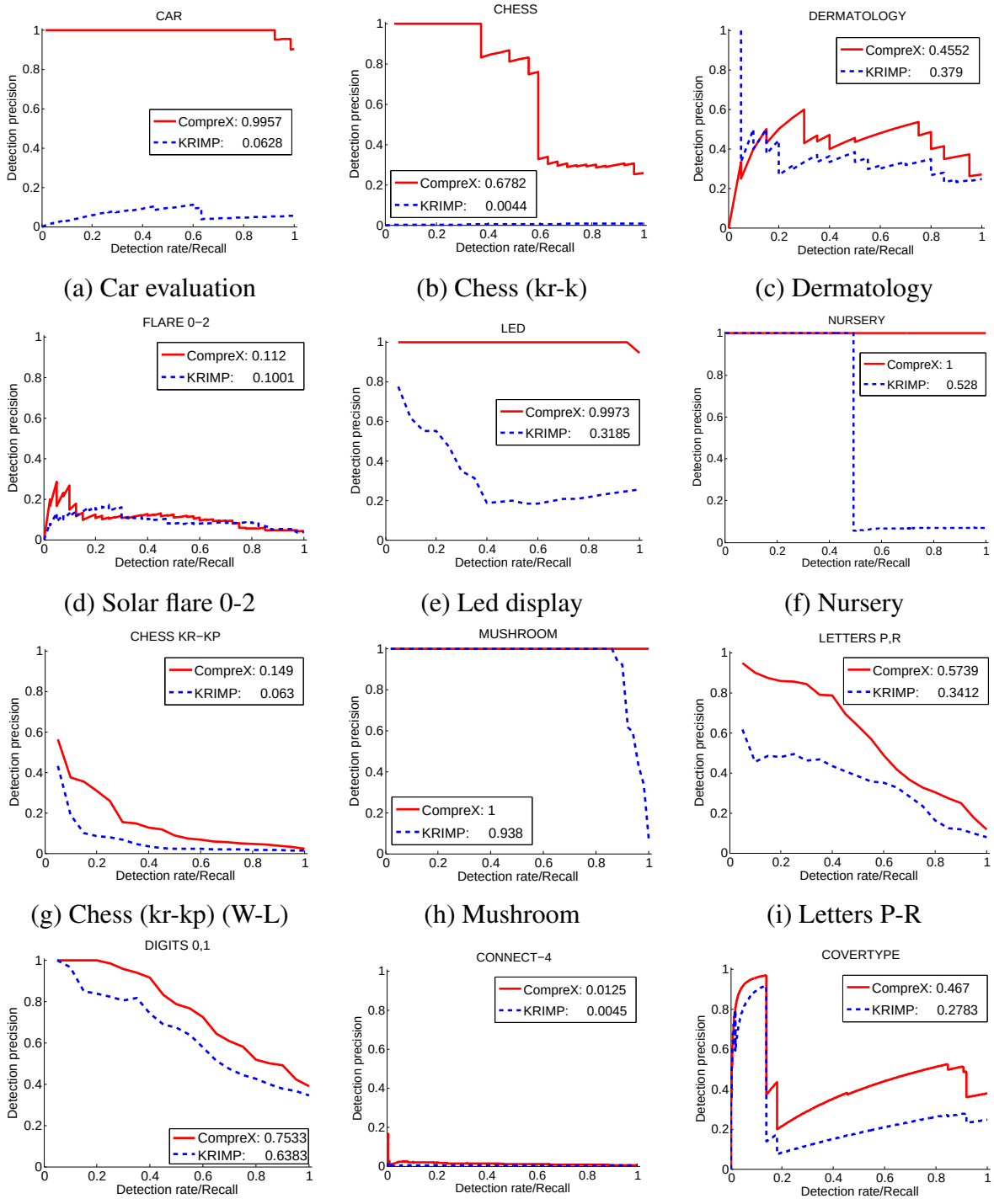
As a measure of accuracy, we use *average precision*; the average of the precision values obtained across recall levels. We plot the detection precision, that is the ratio of the number of true positives to the total number of predicted positives, against the recall (=detection rate), that is the fraction of total true anomalies that are detected. A point on the plot is obtained by setting a threshold compression cost—any record with a cost larger than that threshold is flagged as an anomaly. The corresponding precision and recall are then calculated. By varying the threshold, we obtain the curve for the entire range of recalls. The *average precision* then approximates the area under the precision-recall curve.

Figure 11.2 shows the precision-recall plots of COMPREX and KRIMP on several of our categorical datasets. Here, a higher curve denotes better performance, since it corresponds to a higher precision for a given recall. The average precision values are also given in Table 11.2, for all the categorical datasets. Notice that for most datasets COMPREX achieves higher accuracy than KRIMP. This is obvious especially for the Car, Chess, Led and Nursery datasets. We notice that the performance of the methods also depends on the detection task. For example, both methods perform well on the Mushroom dataset, for which the poisonous ones exhibit quite different features than the edible ones. However, the accuracies of both methods drop for the Connect-4 dataset for which the detection task, i.e. which player is going to win the game given the 8-ply positions, is much harder.

COMPREX on numerical data

While COMPREX is designed to work with categorical datasets, it can also be used to detect anomalies in datasets with numerical features. For that, we first convert the continuous numerical features to discretized nominal features. There exist various techniques to this end. In our study, we consider several: linear, logarithmic, SAX [Lin et al., 2003], and MDL-based [Kontkanen and Myllymki, 2007] binning. Linear binning involves dividing the value range of each feature into equal sized intervals. Logarithmic binning first sorts the feature values and assigns the lower b-fraction to the first bin, the next b-fraction of the rest to the second bin, and so on, until all the values are assigned to a bin. SAX has proved to be an effective symbolic representation, especially for time series data. MDL-based binning estimates variable-width histograms with optimal bin count automatically, for a given data precision.

We experiment with these various discretization methods under various parameter settings. In Figure 11.3, we show the accuracy of COMPREX versus KRIMP on the Shuttle dataset, using logarithmic binning with b=0.5, linear binning with 5 and 10 bins, SAX with 3 bins, and MDL-



(a) Car evaluation (b) Chess (kr-k) (c) Dermatology

(d) Solar flare 0-2 (e) Led display (f) Nursery

(g) Chess (kr-kp) (W-L) (h) Mushroom (i) Letters P-R

(j) Digits 0-1 (k) Connect-4 (l) Cover-type

Figure 11.2: Performance of COMPREX vs KRIMP on several two-class categorical transaction datasets. Plotted are, precision vs. recall for various thresholds to flag data points as an anomaly. Notice that COMPREX outperforms KRIMP for most detection tasks.

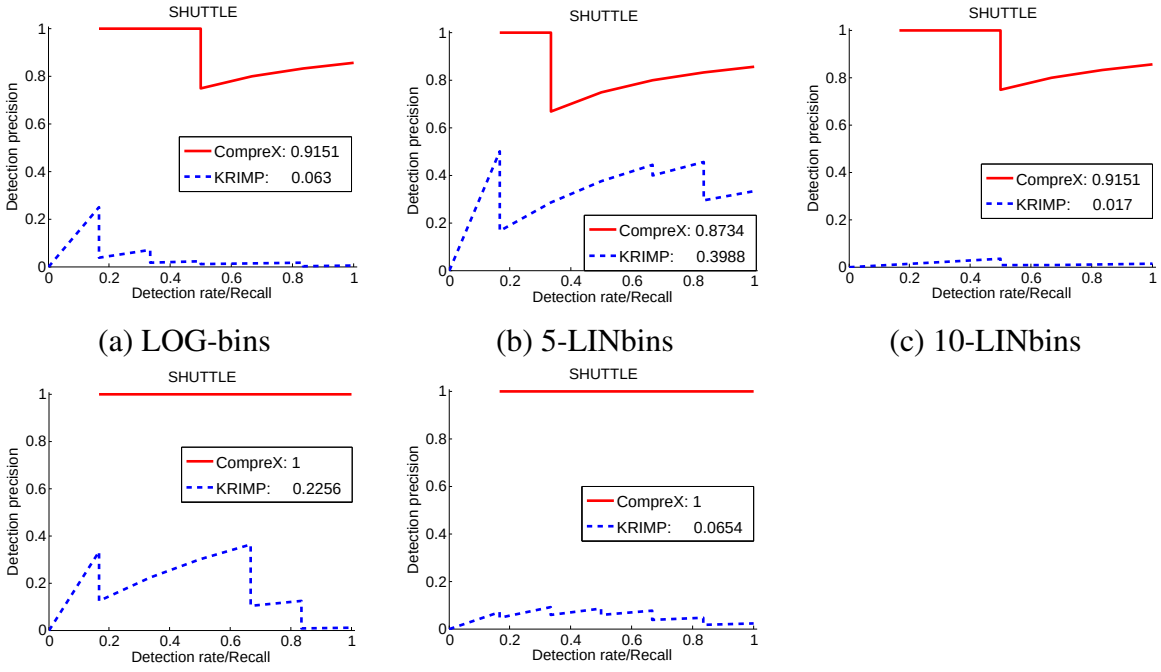
Dataset	$\|D\|$	$\|\mathcal{F}\|$	$\|\mathcal{P}\|$	*minsup*	CompreX	Krimp
Car evaluation	1275	6	6	1	**0.99**	0.06
Chess (k)	4580	6	4	1	**0.67**	0.01
Dermatology	132	12	12	1	**0.45**	0.37
Solar flare 0–1	1312	10	6	1	**0.19**	0.15
Solar flare 0–2	1211	10	6	1	**0.11**	0.10
Led display	370	7	7	1	**0.99**	0.31
Nursery	4648	8	6	1	**1.00**	0.52
Chess (kp) W-L	1689	36	18	500*	**0.14**	0.06
Chess (kp) L-W	1547	36	18	500*	**0.03**	**0.03**
Letter rec. A–B	809	16	11	1*	0.17	**0.18**
Letter rec. P–R	823	16	11	1*	**0.57**	0.34
Mushroom	4258	22	5	1*	**1.00**	0.93
Digit rec. 0–1	1163	16	12	10	**0.75**	0.63
Digit rec. 1–7	1163	16	9	20	**0.51**	0.34
Spect–heart	267	22	16	1*	**0.12**	**0.12**
Connect–4	44k	42	11	5k*	**0.02**	0.01
Covertype	286k	44	6	280k*	**0.46**	0.27
				MAP	**0.48**	0.26

Table 11.2: Average precision (normalized area under the precision-recall curve) for the categorical datasets, comparing COMPREX and KRIMP. Further, we give dataset size, number of features and partitions, and $minsup$ used for KRIMP (star denotes *closed* itemsets).

based binning with precision 0.01. Results are similar for many other settings and for the rest of the numerical datasets, which we omit for brevity. Notice that regardless of the discretization used, COMPREX performs consistently better than its competitor KRIMP.

To this end, we compare our method with LOF on some numerical datasets. LOF is a widely used outlier detection method based on local density estimation. While it is quite powerful when applied to numeric data, it cannot be directly applied to categorical datasets. In this comparison, both methods require the careful choice of a parameter; the number of nearest neighbors k for LOF, and the binning method and its corresponding parameter for COMPREX.

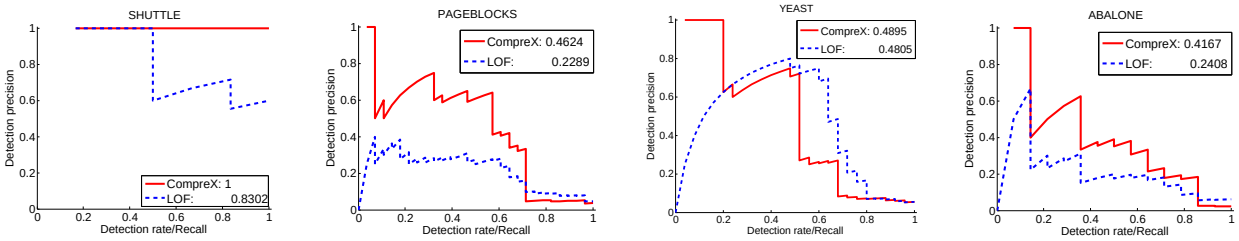
In Figure 11.4, we show the accuracy of LOF versus COMPREX with their best parameter choices on our numerical two-class datasets. Table 11.3 gives the corresponding average precision scores for all three methods. Notice that COMPREX achieves comparable or better performance than LOF, even though it operates on discretized data which loses some information due to this process, and thus is not as optimized as LOF for numeric data. This shows that COMPREX can also be applied to datasets with numerical or with a hybrid of both categorical and numerical features.



(a) LOG-bins (b) 5-LINbins (c) 10-LINbins

(d) 3-SAXbins (e) p0.01-MDLbins

Figure 11.3: Performance of COMPREX vs KRIMP on the two-class numerical transaction dataset Shuttle. Notice that COMPREX outperforms KRIMP consistently for various discretization methods used.



(a) Shuttle (b) Page blocks (c) Yeast (d) Abalone

Figure 11.4: Performance of LOF vs COMPREX with the best choice of parameters on the numerical transaction datasets. Notice that COMPREX achieves comparable or better performance than LOF even after discretization.

COMPREX on graph data

Given data points and their features in numerical or categorical space, our method can also be applied to other complex data, including graphs. To this end, we study the Enron graph, in which nodes represent individuals and the edges represent email interactions. In our setting the nodes correspond to the data points, and the features to the ego-net features we extract from each node. The ego-net of a node (ego) is defined as the subgraph of the node, its neighbors, and all the

Dataset	$\|D\|$	$\|\mathcal{F}\|$	$\|\mathcal{P}\|$	*minsup*	CompreX	Krimp	LOF
Shuttle	3416	9	5	1	**1.00**	0.22	0.83
Pageblocks	4941	10	6	1	**0.46**	0.37	0.23
Yeast	468	8	8	1	**0.49**	0.17	0.48
Abalone	703	7	3	1	**0.42**	0.15	0.24
				MAP	**0.59**	0.23	0.45

Table 11.3: Average precision for the numerical datasets, comparing COMPREX, KRIMP and LOF. Further, we give dataset size, number of features and partitions, and $minsup$ used for KRIMP.

links between them. We extract 14 numerical ego-net features, such as the number of edges, total weight, ego-net degree (number of edges connecting the ego-net to the rest of the graph), in- and out-degree, etc. Features are discretized into 10 linear bins.

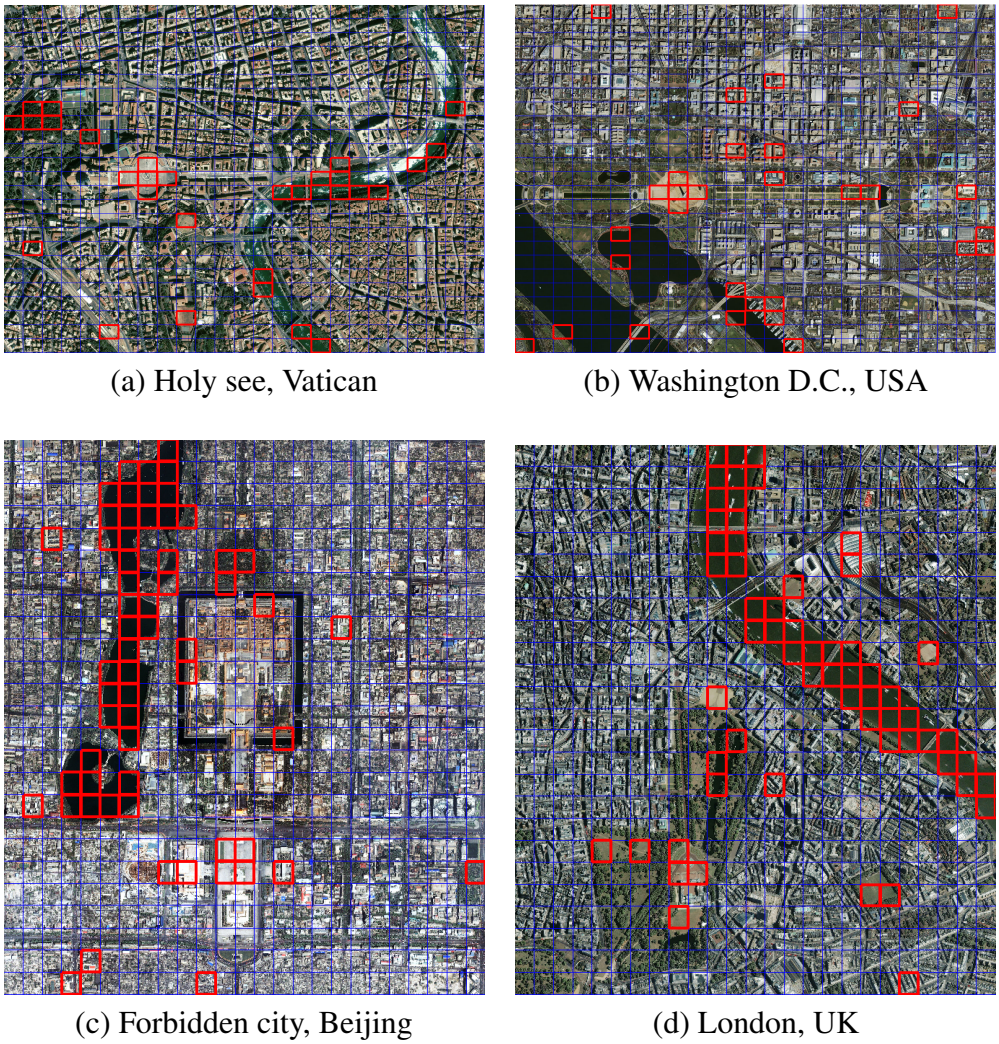
In Table 11.4, we show the top 5 email addresses with the highest compression cost found by COMPREX. The dataset does not contain any ground truth anomalies, therefore we provide anecdotal evidence for the discovered "anomalies". Our first observation is the significantly high compression cost of the listed points—103 to 107 bits given a global average of 6.72 bits (median is 4.27). This is due to the rare and high number of patterns used in their cover. Notice that each of them are covered with 7 patterns compared to a global average of 2.05 (median is 2). Moreover, the usages of the cover patterns is quite small—thus longer code words and high total compress-cost. Further inspection justifies our results: for example, 'sally.beck' (employee chief operating officer) contains the highest number of (31k) edges in its ego-net and the highest ego-net degree (of 85k), implying that she is highly connected to the rest of the graph as opposed to many other nodes in the graph.

name@enron.com	*cost (bits)*	$\|cover\|$	*avg usage*$\pm$*std of cover patterns*
sally.beck	107.28	7	3.7 ± 5.4
jeff.dasovich	107.11	7	3.4 ± 3.9
outlook.team	106.70	7	4.8 ± 6.3
david.forster	105.11	7	4.1 ± 5.2
kenneth.lay	103.24	7	5.8 ± 7.5
⋮	⋮	⋮	⋮
robert.badeer	1.53	2	52k $\pm$ 4.3k

Table 11.4: Top-5 anomalies for Enron, with one regular-joe example. Given are, email address, compression cost, size of cover, and average usages of the cover patterns; high usages correspond to short codes. Average cost is 6.72 bits, average number of covering patterns is 2.05.

COMPREX on image data

Next, for our image datasets for which class labels also do not exist, we provide an anecdotal and visual study. The image datasets are the satellite images of four major cities from around the world as shown in Figure 11.5. Each image is split into 25x25 rectangle tiles, for which we extracted 15 numerical features, and subsequently discretized into 10 linear bins. The first three features denote the mean RGB values for each tile and the rest denote the Gabor features.



(a) Holy see, Vatican

(b) Washington D.C., USA

(c) Forbidden city, Beijing

(d) London, UK

Figure 11.5: "Anomalous" tiles—with high compression cost— on the image datasets are highlighted with red borders (figures best viewed in color). Notice that COMPREX successfully spots qualitatively distinct regions that stand out in the images.

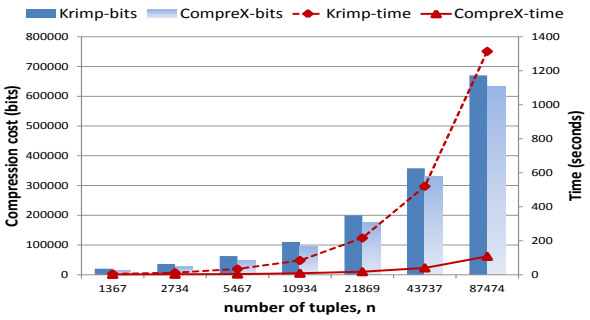
In Figure 11.5, top tiles with high compression cost are highlighted in red. We observe that COMPREX effectively spots interesting and rare regions. For example, the districts of Roman Catholic Church in Vatican and the Washington Memorial in Washington D.C. that distinctively stand out in the images are captured in top anomalies. In Forbidden city, COMPREX spots the

three lakes (Beihai, Zhonghai, Nanhai), the Jingshan Park on its right, as well as the Tiananmen Square on the south. Finally, in London CompreX marks the part of Thames river, the Buckingham Palace as well as several rare plain fields in the city.

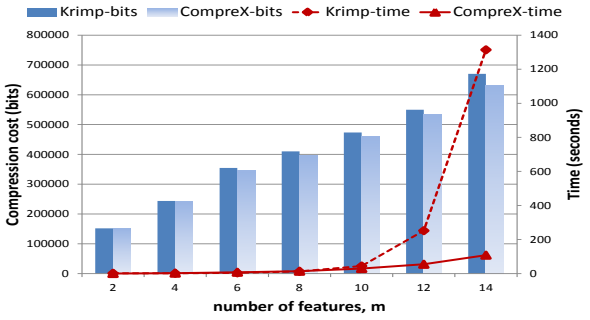
11.2.3 Scalability

Krimp falling short on large datasets arises the question of scalability, which is maybe even more important than the issue of speed. Therefore, in Figure 11.6, we also show the running time of both methods for growing dataset and feature sizes on Enron and Connect-4.

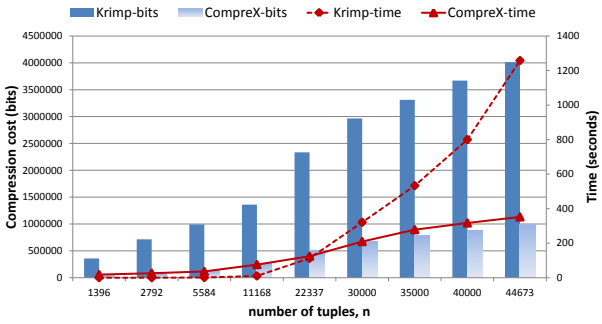
We observe that the running time of Krimp grows significantly with the increasing size in both cases. The difference is evident especially for growing feature size. This is due to frequent itemset mining scaling exponentially with respect to number of features. On the other hand, the increase in running time for our method follows a linear trend indicating that CompreX scales better with increasing database size and dimension.



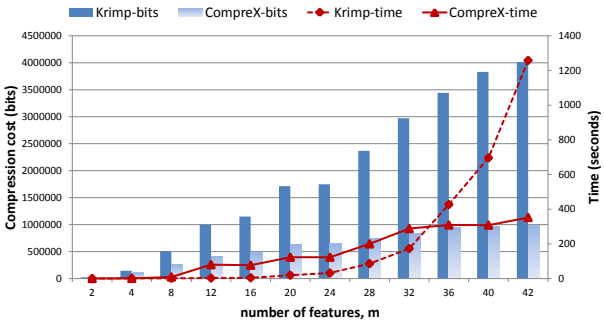
(a) Enron time vs. data size



(b)Enron time vs. feature size



(a) Connect-4 time vs. data size



(b) Connect-4 time vs. feature size

Figure 11.6: Scalability of CompreX and Krimp with increasing dataset and feature size. (top) Enron with large n, (bottom) Connect-4 with large m. Notice that CompreX achieves better scalability for large databases, following a linear trend in growth.

We note that the running time of CompreX could be further improved by operating on a smaller, random sample of the data. In a large enough random sample, the patterns are expected to occur as frequently as they occur in the original dataset and therefore the code tables built after random

sampling would be close to those built from the entire data. The theoretical and analytical study of random sampling for code-table compression, in particular the time versus accuracy trade-offs that the sampling poses, constitutes a promising research direction.

An advantage of our method is that it can work as an *any-time* algorithm. As it is a bottom-up approach which seeks for lower total cost over iterations via more merges, the compression procedure can be stopped at any point given the availability of time. The same goal can be achieved for KRIMP by setting the *minsup* parameter accordingly; the higher the minsup, the lower the running time.

To compare the methods, we show the compression costs achieved at various durations in Figure 11.7. Notice that for any given run-time duration, COMPREX achieves lower compression cost than its competitor. Moreover, even KRIMP is allowed to run for longer, it still cannot reach as low of a cost as COMPREX can in much less time.

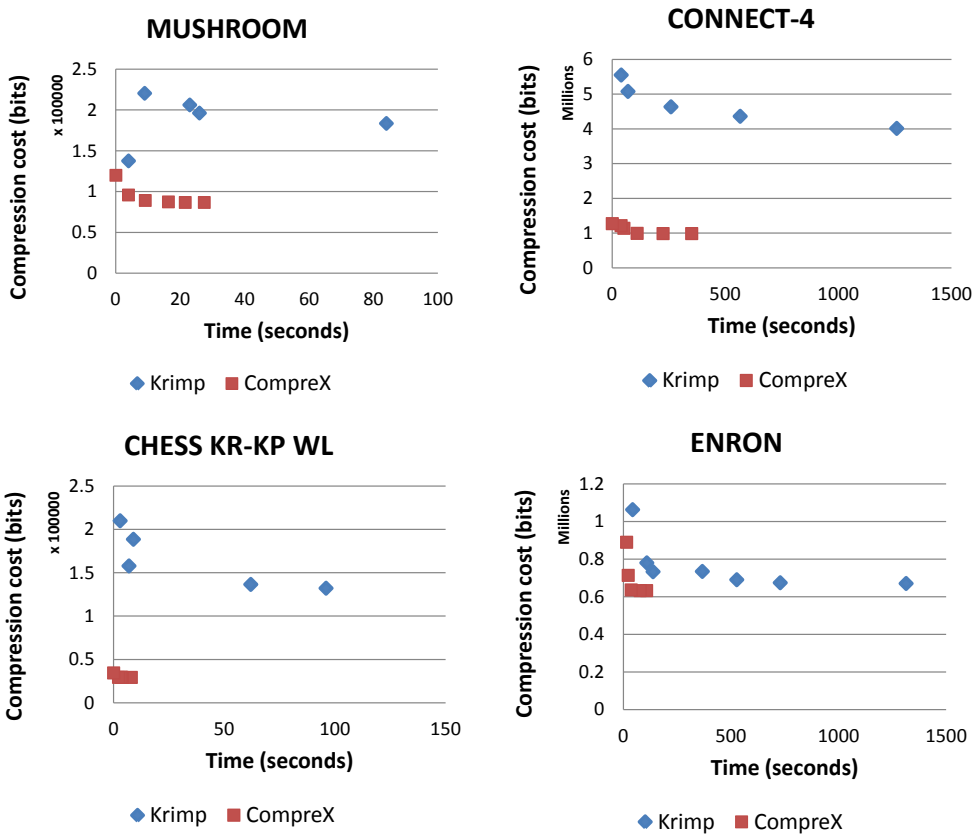


Figure 11.7: Scatter plots of compression cost versus running time of COMPREX and KRIMP. Notice that COMPREX achieves lower cost at any given time, which KRIMP cannot catch up with even when let to run for longer.

11.3 Summary of contributions

In this chapter, we introduced a novel, *parameter-free* method called COMPREX for the important task of anomaly detection in complex multi-dimensional databases. COMPREX builds a data compression model that uses multiple dictionaries for encoding, and reports the data points with high compression cost as anomalous. Our method proves *effective* for both tasks: It provides higher compression rates at lower running times than its state-of-the-art competitor KRIMP, especially for large datasets. In addition, it is capable of spotting rare instances effectively, with detection accuracy that is often higher than its closest competitor KRIMP on categorical datasets, and `LOF`, its state-of-the-art competitor on numerical datasets.

Experiments on diverse datasets show that COMPREX successfully *general*izes to a broad range of datasets including image, graph, and traditional relational databases with both categorical and numerical features. Moreover, it is *scalable*, with running time growing linearly with increasing database size and dimension.

Chapter 12

Events in Time-evolving Graphs

PROBLEM STATEMENT: *At what points in time many of the nodes in a given time-varying graph change their behavior significantly? Can we attribute the change to specific nodes, that is, can we characterize which nodes change in behavior the most?*

In this chapter, we develop an algorithm to spot change-points in a time-varying graph at which many nodes deviate from their normal 'behavior'. In a nut-shell our method works as follows. We first extract time sequence of several network (egonet) features for all nodes in the graph. Then, we derive a 'behavior' vector of all nodes and compare it to recent past 'behavior' vectors detected over several previous time windows. If the current behavior is found to be significantly different than recent past, we flag the current time window as anomalous and report as an event has occurred. Moreover, in order to attribute the change to a subset of nodes which contribute to the change score the most, we compare their 'behavior' values and mark those nodes for which this value differs much more than others.

To demonstrate the effectiveness of our method, we study the texting behavior of users in our SMS network. This data consists of six months' of activity and is therefore time-varying. Also, the edges are weighted, weights denoting the total number of SMSs sent/received between individual pairs. More specifically, the SMS network constructed from the SMS records includes over 2 million users with 50 million SMS interactions between them over a period from Dec. 1, 2007 to May 31, 2008.

12.1 Event Detection: Method Description

12.1.1 Feature Extraction

In order to find patterns that nodes of a graph follow, we characterize the nodes with several features so that each node becomes a multi-dimensional point. In particular, each node is sum-

marized by a set of features extracted from its egonet (egonet of a node includes the node itself, its neighbors, and all the interactions between these nodes). The 12 features considered in this work are as follows: 1) in-degree, 2) out-degree, 3) in-weight, 4) out-weight, 5) number of neighbors, 6) number of reciprocal neighbors, 7) number of triangles, 8) average in-weight, 9) average out-weight, 10) maximum in-weight, 11) maximum out-weight, and finally 12) maximum weight ratio on reciprocated edges in the egonet.

12.1.2 Change-Point Detection

The flow of our method to detect change-points in the behavior of nodes is illustrated in Figure 12.1. This method is similar to [Idé and Kashima, 2004], but differs in the construction of the 'dependency' matrices C (top right in the figure) as we describe in the following.

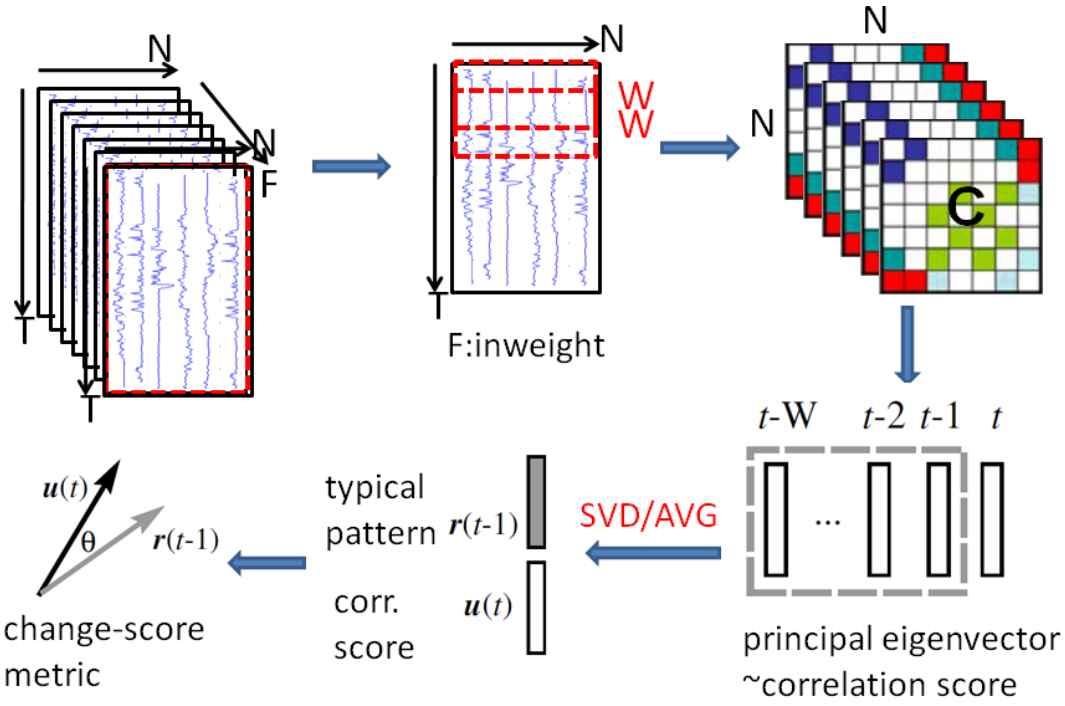


Figure 12.1: The Work-flow of Change-Point Detection

Here, the data we study looks like the $3-D$ tensor on the top left of Figure 12.1, where T denotes the number of time ticks ($T = 183$ days), N denotes the number of nodes in our graph ($N = 2$ million customers), and F denotes the number of features extracted for each node ($F = 12$ as described in the previous section). To start with, we take one 'slice' of this $3-D$ $T \times N \times F$ tensor for a particular feature F_i, say in-weight, which is a $T \times N$ matrix (top middle in the figure). Next, we define a window of size W over the time-series of values of all nodes for that particular feature F_i. Then for pair of nodes, we compute the correlation between their time-series vectors over the window of size W using Pearson's ρ as follows.

$$\rho_{X,Y} = \left| \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \right|$$

In the above equation, X and Y are the length-W vectors for each node pair (X, Y). So, for each window we construct a correlation matrix C, where $C_{x,y} = \rho(x, y)$ over window W. Next, we slide the window down one time tick (day) and compute the correlations over the next window of W time ticks. Similarly we keep repeating this process until we reach the end of our data. To be representative and given the periodic behavior of human nature, we chose the size W as 7 days (one week). As a result, we end up constructing 177 C matrices (top right in the figure).

By the Perron-Frobenius theorem [Pillai et al., 1912], the largest (principal) eigenvector of each of the C matrices is positive. The value for each node in the eigenvector can be thought as the 'activity' of that node; that is, the more correlated a node is to the majority of the nodes, the higher its 'activity' value will be. Here, we call each such size N eigenvector as the 'eigen-behavior' of all the nodes in the graph on the whole.

Note that in principle, we do not have to explicitly compute the C matrices in order to obtain its eigenvector(s). Since $C = W^T W$ after standardizing W's rows, the right singular vectors of W are the same as the eigenvectors of C. Therefore, we obtain the eigen-behaviors by operating directly on the W matrices.

Metric to Score Time Points for 'Event'

After finding all the eigenvectors for all the 177 C matrices, the change-point in the 'eigen-behavior' of the nodes is found as follows. For the eigenvector computed at time say t denoted by $u(t)$, we compute a 'typical eigen-behavior' denoted by $r(t-1)$ from the previous W' eigen-vectors (bottom right in the figure). We experimented with two different ways to compute the mentioned typical eigen-behavior. Firstly, we simply took the arithmetic average of all previous W' eigenvectors. Secondly, we constructed a new $N \times W'$ matrix, each column being an eigen-vector over that window, and then we computed the left singular vector of that new matrix using SVD decomposition. Similar to the principal eigenvector for square positive matrices, the left singular vector of a positive non-square matrix yields an average 'behavior' score for all nodes. One can think of the left singular vector as the weighted average of eigenvectors in W'.

Finally, after we obtain the 'typical eigen-behavior' for each C matrix (for each week) using either SVD or regular averaging, the 'eigen-behavior' $u(t)$ computed at time t is compared to the 'typical eigen-behavior' $r(t-1)$ by taking the dot-product of those two unit vectors. The change metric we used is $Z = (1 - u^T r)$. Here, if $u(t)$ is perpendicular to $r(t-1)$, then their dot-product gives a value of 0, or $Z = 1$, whereas if $u(t)$ is exactly the same as $r(t-1)$, then their dot-product gives a value of 1, or $Z = 0$. Therefore, Z takes values between 0 and 1, and a higher value of Z indicates a change-point (bottom left in the figure).

12.2 Experimental Results

We start our analyses by looking at the distribution of correlation values $C_{x,y}$ in the C matrices. Figure 12.2 shows the histogram as well as the CDF of $C_{x,y}$ values for two different days, Dec. 1 and Dec. 26, for feature F:in-weight.

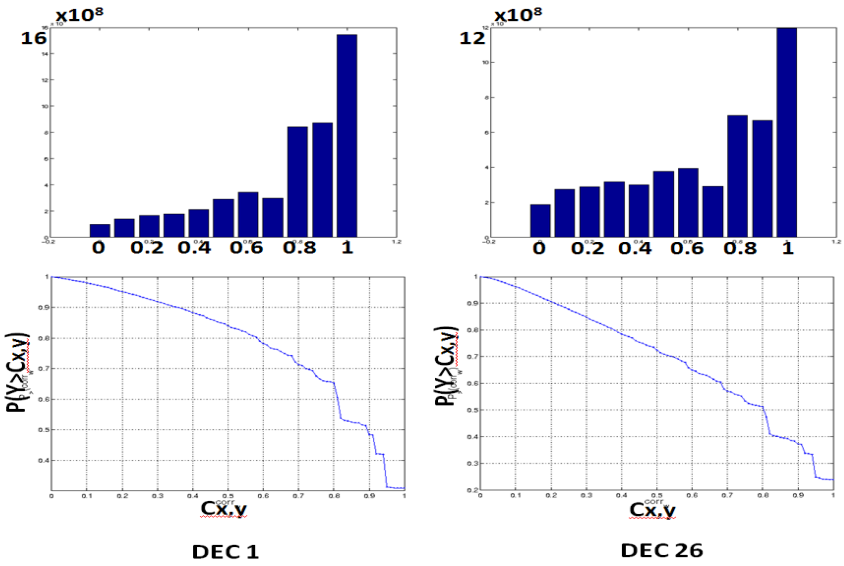


Figure 12.2: (top) Histogram and (bottom) CDF distribution of correlation scores $C_{x,y}$ for (left) Dec. 1 and (right) Dec. 26 using F: in-weight.

Here, one observation is that the distribution of correlations between time-series of nodes is skewed as might be expected. Surprisingly, though, it is skewed towards large values. That is, there are lots of pairs with correlation score close to or equal to 1. This happens because over the time window W of 7 days, most of the nodes have no activity –their W-length vectors are all 0's and thus the pair-wise correlations of such 0 vectors are computed to be 1. This suggests that the nodes have bursty activity where nodes have no activity for several weeks and have activity at bursts.

Another observation is that the total number of correlation scores 1 reduces in Dec. 26 compared to Dec. 1, suggesting for no activity weeks for fewer nodes, that is, more nodes become active during the week of Dec. 26. This is expected as this week is the New Year week. We note that the CDF distributions for these two days also look different. These observations strengthen our belief of studying correlations between behaviors of nodes would be important in detecting the change-points in our data.

Next, we compare the results when using SVD versus taking the regular average (AVG) for computing the 'typical eigen-behavior' $r(t-1)$ of earlier eigenvectors over a window of W' (see Section 12.1.2). Figure 12.3 shows the so-called Z scores computed (1) when $r(t-1)$ is computed with SVD (in blue bars) versus (2) when $r(t-1)$ is computed by simply taking

the average (in red lines) for four different values of W', (from left to right, top to bottom) $W' = 5, 7, 20, 50$. Notice that the red line almost exactly follows the blue bars. This means that SVD is giving equal weight (importance) to all W' eigenvectors in the past same as the AVG does. Therefore, since computing the average is less expensive, we will use the AVG to compute $r(t-1)$ in the rest of our experiments.

We also note that the Z scores follow somewhat a similar trend when different window sizes are considered. However, the larger the window gets, the more aggregated the results become (notice that the four spikes for $W' = 5, 7$ reduce to two spikes only for $W' = 20, 50$). In the rest of our experiments, we use $W' = 5$ over which the $r(t-1)$ vector is computed by using AVG.

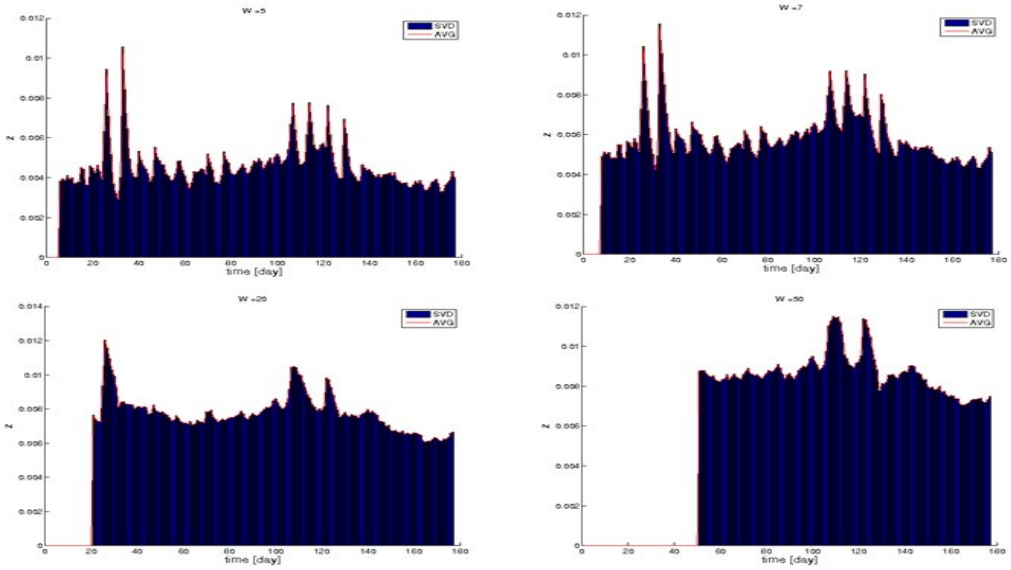


Figure 12.3: Z scores computed when the typical eigen-behavior vector $r(t-1)$ is computed by taking the SVD (blue bars) versus the regular AVG (red lines) for (from left to right top to bottom) $W' = 5, 7, 10, 20$. Notice AVG is very similar to SVD.

12.2.1 Detected Change-Points

After computing the deviation scores Z as was explained in Section 3.3, we use a simple heuristic to flag the high Z scores. Rather than using a threshold value, we simply compute the difference between two consecutive Z scores and rank the time points according to $|Z(t) - Z(t-1)|$. Figure 12.4 shows the top 10 time ticks for which the difference score is the highest. Here, feature F is taken to be the "inweight". Experiments with other features such as "number of reciprocal edges" and "outdegree" also flag similar time points which we will discuss later in this section.

In Figure 12.4, we observe that the top 2 time periods correspond to the weeks of Christmas and New Year (Dec 26, Jan 2). This shows that even though our data comes from India and mostly

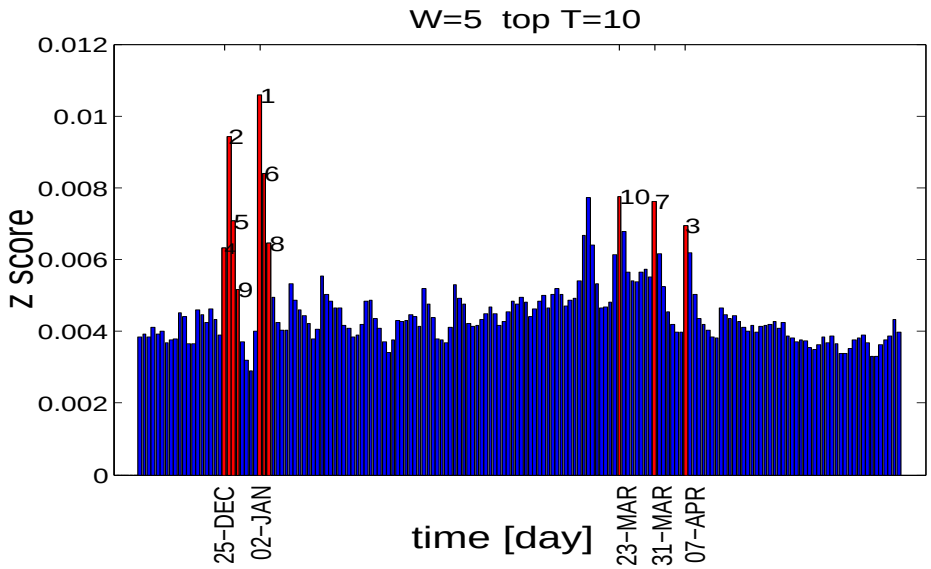


Figure 12.4: Top 10 time points with highest Z-scores flagged by our method (red bars) for feature F:inweight. Numbers on bars indicate rank of each day by Z-score.

people are not Christian, they would be "celebrating" the Christmas. The reason that Jan 2nd rather than Jan 1st is flagged is it is a change-point in which things went back to normal.

Another surprising finding is with the 3rd time tick which is Apr 7th. Similar to Jan 2nd, this is also a time-point where things turned back to normal. The actual interesting day here is indeed Apr 6th: `http://www.infoplease.com/ipa/A0777465.html` lists Apr 6th as the "Hindi New Year" (our data is in 2008). These results suggest that our method is effective in finding points in time for which the collective behavior of nodes deviate from recent past.

As a sanity check, we ran our method on other features such as *number of reciprocal edges* and *outdegree*. Figure 12.5 shows that our method flags almost the same time points including Jan 2nd and April 7th also with these features. Moreover, the difference/spike in the Z score is even clearer with these methods. This is intuitive in the sense that even though the "inweight" (number of SMSs received) is expected to increase on days such as Christmas and New Year, the number of reciprocated interactions are expected to increase even more (people tend to reply to celebration messages on such days).

We also compared the results of our method to the results we obtain by using the sheer volume at each time tick. In particular, we computed the total number of SMSs received (in-weight) per day and marked the top $k = 10$ time points for which the most number of SMSs were received in total. We repeated the same process for total number of reciprocal replies (numrecip) and total number of out-going contacts (out-degree).

We show the results in Figure 12.6 where the red bars depict the top 10 time points with highest volume. We observe that just the total volume for all three features was enough to detect change on for example Dec. 31 and Jan 1. However, we realize that the points reported for each feature

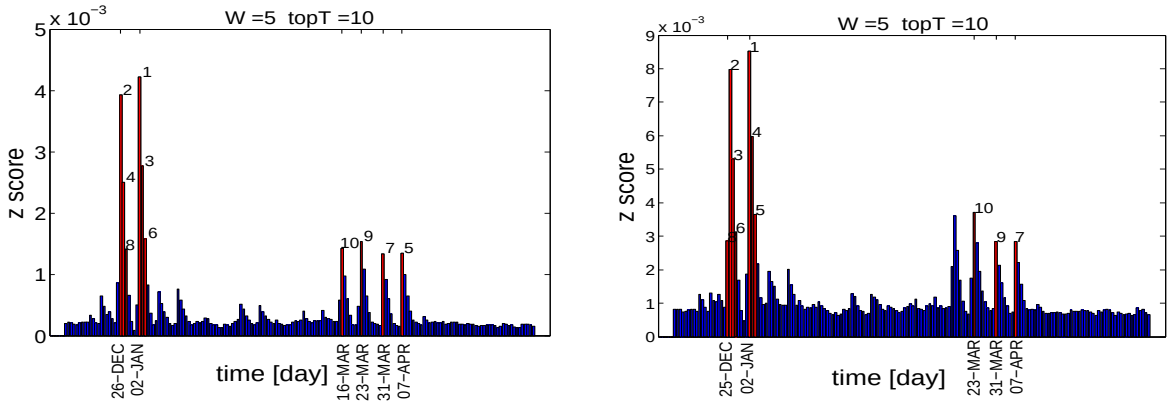
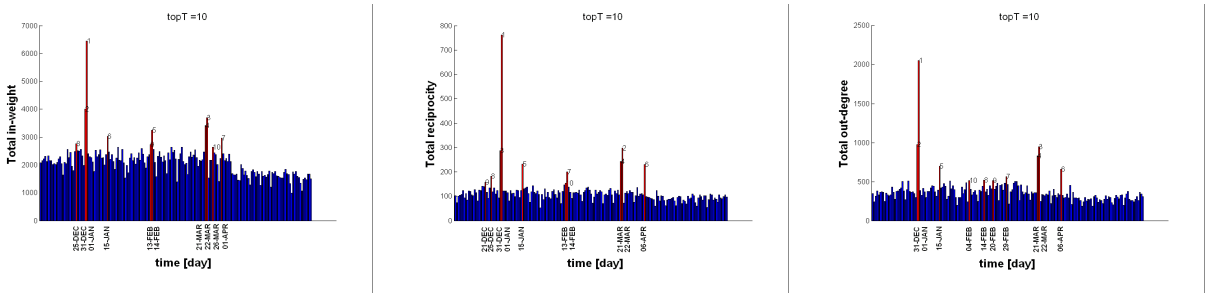


Figure 12.5: Top 10 time points with highest Z-scores flagged by our method (red bars) for (left) F:number of reciprocal edges and (right) F:outdegree. Notice the flagged time points are similar to those using F:inweight in Figure 12.4.

also partly differ from each other and are not as consistent as the earlier results. For instance April 6, even though was detected as a change-point using features 'numrecip' and 'out-degree', was not detected using 'in-weight'.

The main reason behind these observations is that our method considers every person in the network individually and flags change-points if the majority of them change their 'normal' behavior whereas the total volume considers the aggregated behavior. The aggregated data loses information in the individual level and thus is prone to flag change-points if only a few people change their behavior sufficiently a lot.



(a) F: in-weight (b) F: numrecip (c) F: out-degree

Figure 12.6: Top time points flagged by using total volume (red bars).

12.2.2 Attribution of Change

Here the question is for a given change-point detected in the previous section, can we go back and detect which node(s) contributed to the change the most?

Figure 12.7 shows the scatter plot of the values of the eigen-scores $u(t)$ versus the typical pattern $r(t-1)$ scores for all the nodes on December 26th. Here, we observe that most of the values lie on the diagonal, which shows that a majority of the nodes did not change much on their typical behavior. On the other hand, some points that are far off-diagonal (marked with red stars) contribute to the Z score the most.

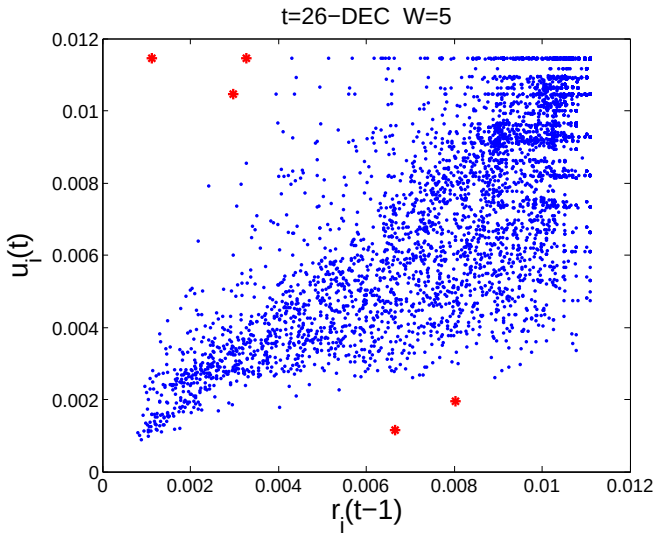


Figure 12.7: Scatter plot $u(t)$ versus $r(t-1)$ of nodes on December 26th. Each blue dot depicts a node. Nodes far away from the diagonal change in "behavior" the most (top 5 marked with red stars).

Similarly, Figure 12.8 shows the amount of change ratio $\frac{|u_i(t)-r_i(t-1)|}{r_i(t-1)}(\%)$ for 10K nodes. Again, the same top 5 nodes as in Figure 12.7 are marked in red.

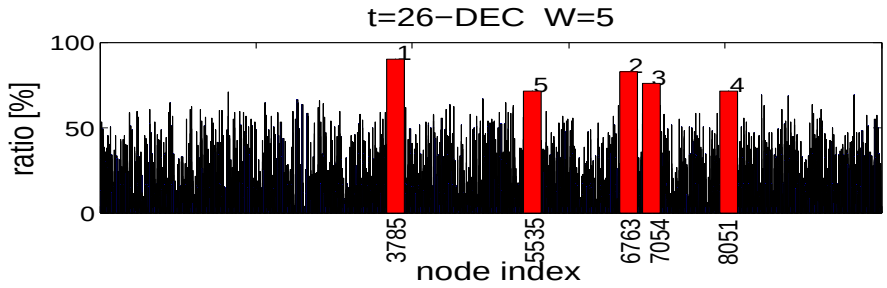


Figure 12.8: Change ratios (%) of top 10K nodes in $u(t)$ and $r(t-1)$. Each bar depicts a node (top 5 with highest change ratio is shown in red).

Since the data does not contain ground truth of anomalies, in Figure 12.9 we plot the time series (inweights versus days) of these top 5 nodes marked in Figures 12.7 and 12.8 (each row for each node). Here we observe that, three of the nodes (rows 1, 4 and 5) have no activity on the week of Dec 26th. This is flagged because they are observed to have some activity over the previous

weeks. On the other hand the other two nodes (rows 2 and 3) have the opposite behavior: they start receiving SMSs during Christmas. We also observe that these two sets of nodes lie in the different halves of the diagonal in Figure 12.7, also indicating an opposite change in their behaviors.

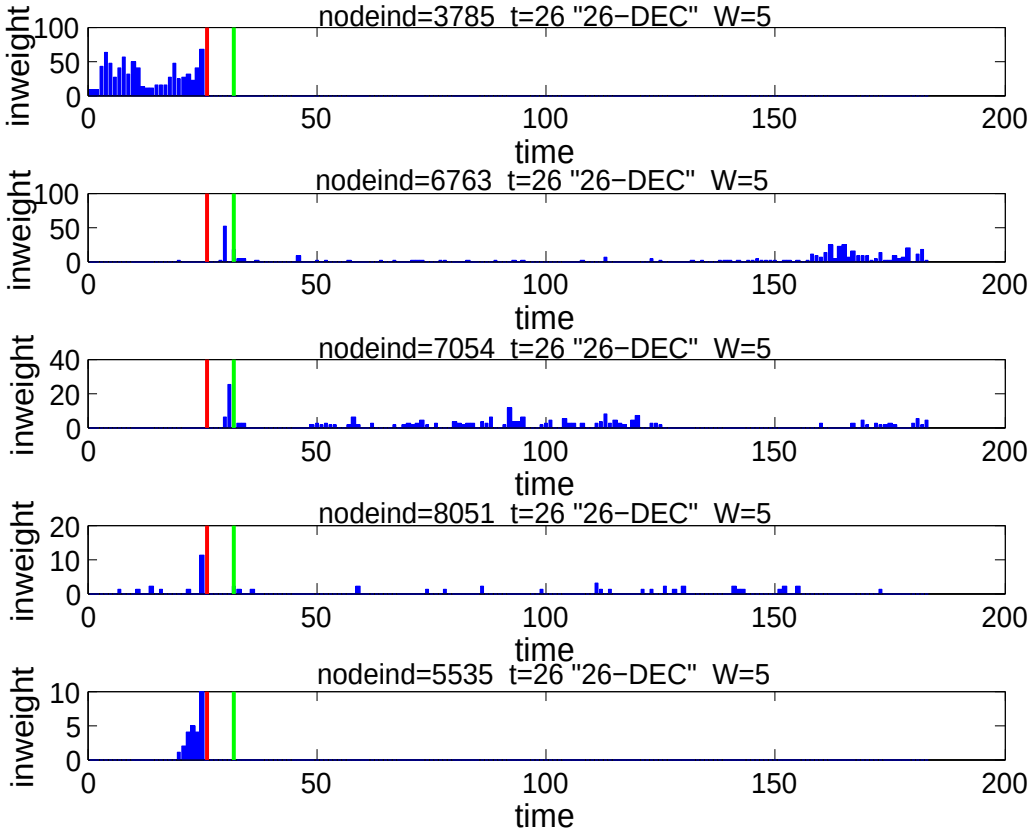


Figure 12.9: Time series of inweight values of top 5 nodes with highest deviation in "eigenbehavior" marked in Figures 12.7 and 12.8. Beginning and end of week December 26th is marked with red and green vertical bars on the time line, respectively.

12.3 Summary of contributions

In this chapter, we introduced an algorithm based on 'eigen-behavior' analysis to (1) spot 'change-points' in time at which the majority of the nodes in a given network deviate from their normal behavior; and (2) do 'attribution' to point out the specific nodes that are most related to the cause of a detected change-point.

We validated the effectiveness of our method on a network dataset of millions of mobile phone users and their SMS interactions over half a year. Although there exists no ground truth information in our SMS data analyzed, the experimental results suggest that our method is able to detect interesting time points.

Our contributions are summarized as the following:

- We used an "eigenbehavior"-based method on the time-series of users and considered the amount of change in their "eigenbehaviors" to flag change-points in time.
- Our method can also be reverse-engineered to spot the top users who contribute to the changes the most.
- Realistic anomaly detection is difficult with unlabeled data, but our results have demonstrated that we were able to detect events that coincide with major holidays and festivals in our data, as well as users whose change of behavior is evident during these events.

Chapter 13

Sensemaking for Graph Anomalies

PROBLEM STATEMENT: *How can we use the network structure to summarize a set of (anomalous) nodes, by partitioning them such that for each part we have a simple tour connecting the grouped nodes, while nodes in different parts are not easily reachable? In other words, how can we summarize the set of (anomalous) nodes for easy sensemaking?*

We motivate the above questions under the anomaly summarization setting. That is, given k nodes marked by an anomaly detection algorithm, we could group them to reveal associations and connectors, instead of simply listing them. This provides a better understanding (sensemaking) of how these nodes are correlated within the graph; whether they are 'close' to each other, and what pathways or other connectors play crucial role in their associations.

While in this chapter we mention the sensemaking for anomalies scenario, these questions find applications in numerous other settings, examples include the following.

- Given a gene interaction network, an experiment may reveal for particular conditions a number of genes to be up (or down) regulated [Liekens et al., 2011], how can we partition them with respect to their closeness in the graph? This would give us a good summary of groups of close-by correlated genes as well as possible pathways the genes may be involved in.
- Given k nodes marked by an anomaly detection algorithm, how can we explain the anomalies? Instead of simply listing them, we could group them and reveal their associations within groups.
- Given an event affecting a set of nodes in a graph (e.g. people affected by a certain disease, people buying a particular product), how can we group the nodes such that the network structure can be associated with the spread of the event within groups but not quite across groups?
- Given a social graph like Facebook and user attributes of interest (e.g. white girls and black boys in the same school), how can we explain their relations using the graph structure? The partitions, if any, and connection paths among the chosen students may be of interest to segregation studies [Shrum et al., 1988] in social sciences among others.

- Given the Web graph and a set of top ranked pages (nodes) returned by some keyword search, how can we group these nodes matching the keywords into groups and reveal connection paths among them, rather than simply listing them?

Intuitively, we want to partition the given set of marked nodes such that the nodes within a part are 'close-by' while nodes across parts are far apart. In addition, for the 'close-by' nodes in each part, we want to find a 'succinct' subgraph connecting them. Moreover, we rather not visit nodes of very high degree, such as hubs in social networks, because those nodes connect to virtually everything in the graph, they do not provide much information by association. Therefore, we think of such high degree nodes as separators, and try to avoid including them in our 'succinct' subgraphs.

For example, consider Figure 13.1. In (a), a list of 20 authors from DBLP are marked. In this plot, there is no information (other than author names) that explains any correlation among the authors. In (b), the marked nodes are projected to and highlighted in the co-authorship graph. In contrast, here it is hard to observe any patterns as there is information overload. We show our result in (c), which explains the marked nodes with two well-separated groups (islands) as well as revealing simple connections and connectors (bridges) among all the marked nodes.

We formalize this problem in terms of the Minimum Description Length principle [Grünwald, 2007]: a tour (collection of paths) is simple when we need few bits to direct the user from one node to the other. Hence we typically do not want to visit nodes of high degree, as it is more expensive to identify which edge to follow leading from it. Similarly, we require more bits if we have to visit many unmarked nodes in order to arrive to the next marked node. As such, the best tour, i.e. the summary, is the one for which we need the least number of bits to identify all marked nodes.

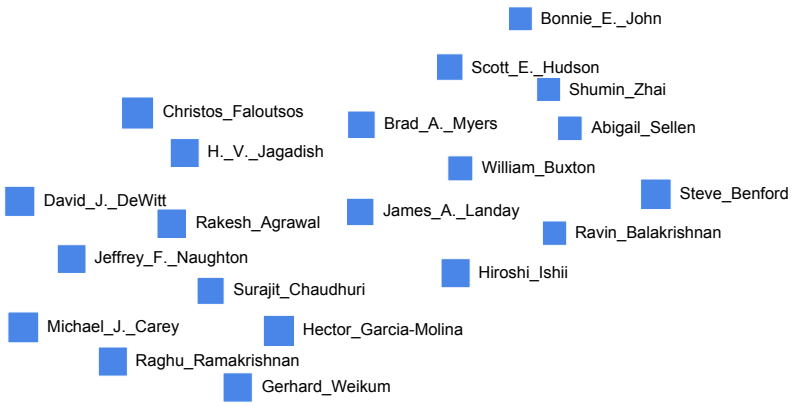
We show this problem is NP-hard, and has connections to well-known problems in network theory. We discuss a number of approximate methods for finding a partitioning and its respective simple paths, and introduce DOT2DOT, an efficient algorithm for summarizing marked nodes. Experimentation shows DOT2DOT correctly groups nodes for which simple paths can be constructed, while separating distant nodes.

13.1 DOT2DOT: Method Description

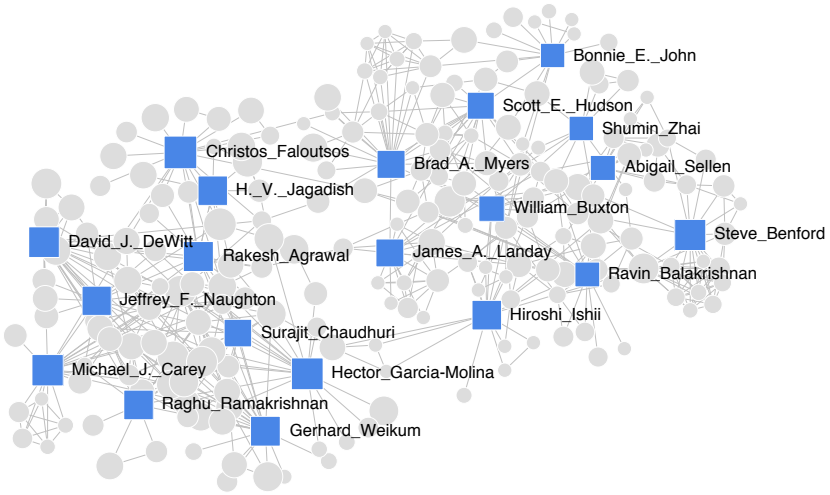
13.1.1 Problem Definition

Given a graph $G = (V, E)$ and a set of marked nodes $M \subseteq V$, we consider the following two related problems.

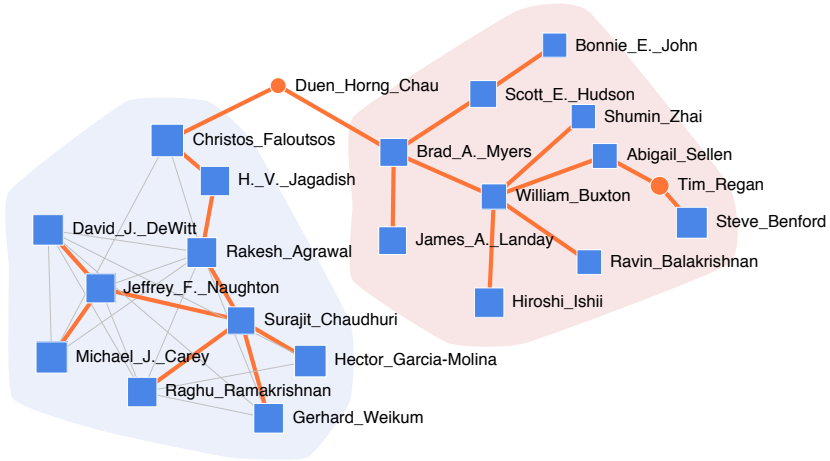
- **Problem 1. Optimal partitioning** Find a 'coherent' partitioning P of M. Find the optimal number of partitions $|P|$ automatically.
- **Problem 2. Optimal connection subgraphs** Find the 'minimum cost' set of subgraphs that connect the nodes in each part $p_i \in P$ efficiently.



(a) What to say about this "list" of authors?



(b) Any patterns? "Too many" connections.



(c) The "right" connections → Better sensemaking

Figure 13.1: 20 chosen authors from DBLP. Edges denote co-authorship relations.

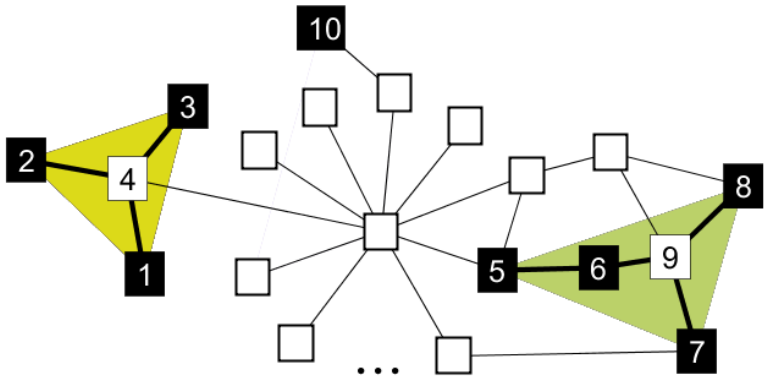


Figure 13.2: Toy graph with 8 marked (black) nodes. Our DOT2DOT algorithm automatically 'describes' them in 3 groups, discovering 'missing connectors' (nodes '4' and '9').

The two problems we consider above involve three sub-problems:

- **(sub-problem 1)** how to define the 'coherence' of a set of nodes,
- **(sub-problem 2)** how to define the 'cost' for a connection subgraph, and
- **(sub-problem 3)** how to find the connection subgraph(s) quickly in large graphs.

As an example, let us consider Figure 13.2. In it, we depict a simple graph in which 8 nodes have been marked. It is clear to see that given the graph structure, the marked nodes naturally form three groups: $p_1 = \{1, 2, 3\}$, $p_2 = \{10\}$, and $p_3 = \{5, 6, 7, 8\}$, which are all well separated by the big star-node in the middle. While only nodes 5 and 6 have a direct connection, for each part there exists a connection subgraph, here highlighted with bold edges, such that we can construct a simple dot-to-dot 'road map' of which branches to follow in order to visit all marked nodes in a part—without having to visit too many unmarked connector nodes, like node 4 for p_1, and node 9 for p_3.

Since we are interested in summarizing, that is, in describing the given marked nodes as succinctly as possible, we borrow ideas from Information Theory and employ the Minimum Description Length (MDL) principle (see Section 8.2 for background on MDL).

13.1.2 Formulating Tours: Theory and Objective

The key idea behind our method builds on an encoding scheme, involving one sender and one receiver. We assume both the sender and the receiver already know graph $G = (V, E)$ and only the sender knows the set of marked nodes M. The goal of the sender, then, is to come up with an encoding scheme to *transmit to the receiver the information of which nodes are marked, using as few bits as possible*.

One straightforward way to encode the set of marked nodes is to encode their node-id's separately, using $\log |V|$ bits each. On the other hand, it might be more efficient to exploit the neighborhood information for 'close-by' nodes. In the simplest case, for example, if nodes u and

v are both marked and they are direct neighbors, i.e. $e(u, v) \in E$, then the sender can encode u's id and then use only $\log d(u)$ bits to encode v, where we assume a canonical order on nodes (say by increasing node-id) and $d(u)$ denotes the degree of u in G. As such, the sender follows a path from one marked node to the other to encode 'close-by' nodes. Depending on the graph structure, from time to time it might be more efficient to restart encoding from a new node by directly providing its id, in case the cost of the path to that node exceeds $\log |V|$ bits. In fact, that is exactly what determines the partitioning P of the nodes.

Simply put, one can imagine the way of encoding discussed above as hopping from node to node for encoding close-by nodes and from time to time flying to a completely new node for encoding farther nodes until all marked nodes are encoded. This resembles a *tour* (union of paths) on the graph that travels from a marked node to another which succinctly describes the marked set (hence the name DOT2DOT).

In effect, we are after the shortest description for a group of marked nodes $M \subseteq V$ in a graph $G = (V, E)$. More generally, the idea is that per part p_i of P, we find the easiest/simplest subgraph T in G that spans *at least* all marked nodes in p_i. Simplicity of T is determined by the number of nodes we visit in this tour, how many unmarked nodes we visit, and in particular how easily per visited node we can identify which edge we have to follow next; nodes with (very) high degree hence make the path more complex, or, less likely. Also notice that the simplest subgraph would in fact be a ***tree*** since it would require less bits to refer to a node we have already visited in our encoding.

In this section, we describe the cost function for a given partitioning P and the given connection trees for each part p_i. We use this cost function as our objective function that we aim to minimize for model selection.

More formally, we first transmit the number of partitions $|P|$, for we need at most $\log |V|$ bits as there will be at most $|V|$ marked nodes in total, which in the worst case are all put in separate parts.

$$L(|P|) = \log |V| \tag{13.1}$$

Then, per part $p_i \in P$, we have a tree T spanning at least the marked nodes of p_i. To identify the root node of T in G we have to spend $\log |V|$ bits.

Then, recursively per node $t \in T$, we transmit how many branches t has, denoted by $|t|$. As t corresponds to a node v_t in G, and $d(v_t)$ gives us the out-degree of v_t, we can transmit $|t| \leq d(v_t)$ in $\log d(v_t)$ bits. On the other hand, since a simple tree would presumably have small branch-out factor, we choose to encode $|t|$ using universal integer encoding [Grünwald, 2007]. This encoding specifies that in order to encode a non-zero positive integer n we require $L_{\mathbb{N}}(n) = \log^\star n + \log(c)$ bits with $c = \sum 2^{-\log n} \approx 2.865064$, and $\log^\star(n) = \log n + \log\log n + \cdots$ sums over all positive terms. So, as $|t|$ can be zero (for leaves), we transmit its value in $L_{\mathbb{N}}(|t| + 1)$ bits.

Next, we identify which out-edges of v_t have to be visited. This we can encode most succinctly by assuming a canonical order of all possible subsets of selected edges of that size, and transmitting the index of the actual subset. This takes $\log \binom{d(v_t)}{|t|}$ bits.

Leaf-nodes are easily identified as their number of branches is given as 0. If such is the case, we traverse back up the tree until we find a node with an unvisited branch. Once all branches have been visited, we stop transmitting the structure of the tree.

Now that we know which nodes $p_i \subset V$ are in our tour, we need to know which of these are marked. Let $|T|$ denote the number of nodes in T, and $||T||$ the number of marked nodes in T. As the recipient now knows the tree, and hence $|T|$, and $||T|| \leq |T|$ we need $\log |T|$ bits to transmit $||T||$. Next, we need to identify which $||T||$ nodes of T are marked, which we again do by a binomial: $\log \binom{|T|}{||T||}$.

As such, for one part $p_i \in P$, we have

$$L(p_i) = \log |V| + L(t) + \log |T| + \log \binom{|T|}{||T||} \tag{13.2}$$

in which per node in the tree of p_i

$$L(t) = L_{\mathbb{N}}(|t| + 1) + \log \binom{d(v_t)}{|t|} + \sum_{j}^{|t|} L(b(t, j))$$

where $b(t, j)$ identifies the node t' in T we reach from node t by descending branch j (notice that by its recursive nature, $L(t)$ encodes the branching cost for all tree-nodes).

Putting this together, we get

$$L(P, M \mid G) = L(|P|) + \sum_{i} L(p_i) \quad . \tag{13.3}$$

Note that our initial assumption that both the sender and the receiver know G does not affect model selection: as G is constant for all possible sets of marked nodes on G, explicitly transmitting G would only add a constant cost for all possible models under consideration, and hence not influence measuring the quality of a model.

Given the above formalization, we now only have to find the optimal partitioning P of M, and the optimal tours per part $p_i \in P$ such that $L(P, M \mid G)$ is minimized.

13.1.3 NP-hardness

The total encoding cost $L(P, M \mid G)$ can score a given set of solutions and point to the best among them, however it does not provide direct means to find the optimal solution.

In this section, we show that this problem is NP-hard, with a reduction to the well-known Steiner tree problem: given an undirected unweighted graph $G = (V, E)$ and a subset of nodes $X \subseteq V$, the objective is to find the minimum cost tree that spans all the nodes in X. The cost of a tree is

defined as the total number of nodes, i.e. total number of edges, in the tree. Note that the tree may include nodes in X, as well as other nodes which are typically referred to as Steiner nodes. Steiner trees are well-known in graph theory, and find application in multicast models for finding good server nodes in computer networks for avoiding network congestion and reducing latency in multi-user streaming data settings.

Theorem 3 *Minimizing $L(P, M \mid G)$ is NP-hard.*

Proof Let $G = (V, E)$ be a graph and let $M \subseteq V$ be the marked nodes. Let $k = |V|$ the number of nodes. We will construct a graph $G' = (V', E')$ such that solving the Steiner tree problem for G will reduce to finding a partition with optimal cost in G'.

In order to do that, let d be the maximal degree of a node in G. We are now ready to define the graph G'. The nodes V' consists of the nodes V and for each edge $e \in E$ we will add s auxiliary nodes. We will define s later. In addition, we will add a large amount, say r, of isolated nodes, we will specify r later. For each edge $e(v, w) \in E$ we will add a path of $s + 2$ nodes connecting v and w with s nodes that were created specifically for this edge. We will refer to these paths as *auxiliary paths*.

Note that auxiliary nodes always have a degree of 2 and they will never be leaves of trees corresponding to the optimal partition. That is, either the whole auxiliary path is in the partition or none of the nodes in that path is included.

The cost of having a non-root auxiliary node in a partition is $c = L_{\mathbb{N}}2 + \log \binom{2}{1}$. If auxiliary node is a root, then the cost is $c_r = L_{\mathbb{N}}3 + \log \binom{2}{2}$.

The encoding of one part is equal to

$$\log |V'| + ncs + corr + \log |T| + \log \binom{|T|}{\|T\|} + \sum_{t \in T, v_t \in V} L_{\mathbb{N}}|t| + 1 + \log \binom{d(v_t)}{|t|},$$

where n is the number of auxiliary paths in T. The correction term $corr$ takes into account the possibility that auxiliary node may be a root term. If this is the case, then $corr = c_r - c$, otherwise $corr = 0$. We can bound the last 4 terms by

$$b = \log k + k \log k + k(L_{\mathbb{N}}d + 1 + d \log d).$$

We can now bound the cost by

$$\log |V'| + ncs + c_r - c + b.$$

Let us define $s = (c_r + c + b)/c$. It is easy to see that the smaller the n, the better is the code for the tree.

Our last step is to make sure that we will have only part in our partition. In order to do that we need to make sure that $|V'|$ is so large that is not beneficial to have more than 1 parts. In order to do this, we set $r = 2^{kcs+c_r+c+b}$ which will guarantee that we will have only one partition.

The optimal partition in G' will now have only one part and will have minimal number of auxiliary paths. Since each path corresponds to an edge in G, we can transform this to a Steiner tree in G with minimal number of edges. This proves the theorem. ∎

13.2 Finding Good Paths: Methods

Since minimizing $L(P, M \mid G)$ is NP-hard, we are interested in fast approximations. To find fast solutions, we will exploit heuristics for the directed Steiner (d-Steiner) tree problem [Charikar et al., 1998]: given a directed weighted graph $G = (V, E)$ and a subset of nodes $X \subseteq V$, the objective is to find the minimum cost *arborescence* that spans all the nodes in X, in other words, a rooted minimum cost tree that has a directed path from the root to every node in X. The cost of a tree is defined as the sum of the costs of the edges in the tree.

The d-Steiner problem is a well-studied combinatorial optimization problem, for which algorithms have been proposed that provide approximation guarantees [Charikar et al., 1998; Helvig et al., 2001; Melkonian, 2007; Zelikovsky, 1997]. However, while providing fairly good bounds, these algorithms all require great amounts of computing power, making them intractable for application on large graphs. In this section we therefore propose fast heuristics for large graphs.

Before we proceed with proposed solutions, we first define how we transform the input graph G to a directed weighted graph G', which we use in obtaining a solution.

Definition 9 *Given an undirected unweighted graph $G = (V, E)$ and a set of marked nodes $M \subseteq V$, the transformed graph $G' = (V, E')$ is a directed weighted graph in which each edge $e(u, v) \in E$ is replaced by two directed edges $e'(u, v) \in E'$ and $e'(v, u) \in E'$, for which the weights $w'(u, v) = \log d(u)$ and $w'(v, u) = \log d(v)$.*

13.2.1 Proposed Algorithms

Given the transformed directed weighted graph G', we want to find *the set of trees with minimum total cost* on the marked nodes. Without loss of generality we assume that the marked nodes will be the leaf nodes in the resulting trees (if a marked node is a non-leaf node, we can always add its copy and connect it to its copy with a zero-cost edge), therefore from hereafter we refer to the marked nodes as the *terminals*.

Finding Bounded-length Paths

Most of our proposed methods use short paths between the terminals as a starting point. Therefore, we first present an algorithm, called `FindBoundedPaths`, to find *multiple* short paths of up to a certain length (hence, indirectly, up to a certain cost), and in order of increasing length between the terminal nodes. The threshold on the length of the paths is not a parameter; as it takes $\log |V|$ bits to start a new partition, we set it to $\log |V|$.

Procedure 1: `FindBoundedPaths` (G, T)

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: multiple (asc. length) short paths SP from terminals

```
1  lengths ← log(degree(V))
2  paths ← ∅, pathcosts ← ∅, SP ← ∅, curlen ← 0
3  foreach t ∈ T do
4      paths.add({t}), pathcosts.add(lengths(t))
5      curlen ← curlen + min(pathcosts)
6      while curlen < log(|V|) do
7          pathcosts ← pathcosts − min(costs)
8          foreach v s.t. pathcosts(v) = 0 do
9              path ← paths(v)
10             foreach n ∈ N_v do
11                 if n ∉ path then
12                     SP.add(from: t, to: n, len: curlen, path)
13                     paths.add({path, n})
14                     pathcosts.add(lengths(n))
15             paths.remove(v), pathcosts.remove(v)
16         curlen ← curlen + min(pathcosts)
17 return SP
```

`FindBoundedPaths` employs a BFS-like expansion starting from each terminal until the threshold path length is reached. The paths from the terminal to the nodes encountered over this expansion as well as their total lengths are stored. A major advantage of our transformed graph formulation for the BFS-like expansion is that the cost of all out-edges for a node v are the same and equal to $\log d(v)$. As a result we only need to keep the length per node, rather than per edge, in our BFS-list.

The pseudo-code is given in Procedure 1. First, *lengths* is a vector of size $|V|$ that holds $\log d(v)$ for each node $v \in V$ (line 1). Second, *paths* is a dynamic list that stores all current list of paths during the BFS expansion. Third, *pathcosts* stores their respective lengths. Last, SP is a structure list that stores the paths from a terminal to other nodes encountered in the BFS expansion (2). The paths are indexed by their end nodes and stored in increasing order of length.

In each iteration, we expand the node(s) with the minimum length (7). For each such node v (8), we expand to its neighbors N_v (10). We first store the path information to these neighbors (12), and then add the new paths from the root to these neighbors to the *paths* list (13), and their respective lengths to the *pathcosts* list (14) for further expansion. Note that a node can appear multiple times in our BFS lists since each expansion is associated with only a single unique path (that is, the path from the root in the BFS tree), and a node may belong to multiple paths. We continue iterating, i.e. expanding nodes with minimum length (16) until the threshold length is reached (6).

At the end of `FindBoundedPaths`, we have all the paths from every terminal to all the nodes in G' that are within a length $\log|V|$ path, ordered in increasing length. Notice that the expansion can be performed completely in parallel for each terminal for speed.

Next, we give fast approximate solutions for finding a low-cost set of trees on the terminals.

DOT2DOT-ConnectedComponents

A basic heuristic to partition the terminals is to simply consider the connected components they induce on the graph. That is, the terminals that are directly connected are put in the same and otherwise separate parts. By definition, all the edges are reciprocated in the transformed graph, and therefore the subgraph induced on each part including two or more nodes is not a directed acyclic tree (it may as well have cycles of length greater than 2). To find the minimum cost rooted directed tree(s) (a.k.a. arborescence), we use the Chu-Liu algorithm [Chu and Liu, 1965].

DOT2DOT-MinArborescence

Our second method (Algorithm 1) uses the *transitive closure* graph of the terminals to find a minimum arborescence. The transitive closure graph $G_t = (T, E_t)$ consists of the terminals and directed edges $e(t_i, t_j) \in E_t$ between terminals having weight equal to the shortest path length $w(t_i \rightsquigarrow t_j)$ from t_i to t_j, $1 \leq i, j \leq |T|$. If the shortest paths between all pairs of terminals are of length less than $\log|V|$, then G_t is simply a directed clique graph. As we find up to length $\log|V|$ paths from every terminal in `FindBoundedPaths`, a terminal does not have an edge in G_t to those terminals that are more than this threshold apart from it.

Algorithm 1: DOT2DOT-MINARBORESCENCE

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: Partitions P: a Steiner tree on each $p \in P$

1 $SP \leftarrow$ `FindBoundedPaths` (G, T)
2 Construct distance-graph (transitive closure) $G_t = (V_t, E_t, d_t)$, where $V_t = T$, and for every $(v_i, v_j) \in E_t$, $d_t(v_i, v_j) = \min -len(SP(v_i, v_j))$.
3 Add universal node u which connects to $\forall v_i \in V_t$ with $d_t(u, v_i) = \log(|V|)$.
4 Find minimum arborescence A of G_t.
5 Delete universal node u and its out-edges from A.
6 **return** `ExpandPartition` (A, SP, T)

Having constructed the transitive closure graph (2), we add a so-called universal node u with directed edges $e(u, t_i)$ to every terminal t_i with weight $\log|V|$ (3). We find the minimum weight arborescence A in this new graph (4). Since the universal node u does not have any incoming edges, it constitutes the root of A. In fact, the number of out-edges of u in A gives us the number of parts $|P|$, and its sub-trees constitute the partitioning P (5). Next, we replace the edges in each of u's sub-trees with their corresponding paths in the transformed graph G' (6). The expanded

sub-trees may contain both marked and unmarked nodes. Also notice that the expanded sub-trees might no longer be trees but contain cycles. Therefore, we rerun the arborescence algorithm on the expanded sub-trees and remove any unmarked leaf nodes in the resulting arborescences, which yields the final forest of Steiner trees. The min-arborescence algorithm takes $O(|V|^2)$ for dense graphs. As we run it on the closure of marked nodes only, we get $O(|M|^2)$.

Procedure 2: `ExpandPartition` (R, SP, T)

Input: tree(s) R, short paths SP, terminals T
Output: Partitions P: a Steiner tree on each $p \in P$

1 Construct subgraph(s) R' by replacing each edge in R by its corresponding shortest path.
2 Find minimum arborescence A of R'.
3 Construct Steiner tree(s) S from A by deleting edges in A, if necessary, s.t. all leaves in S are terminal (marked) nodes.
4 **return** connected components of S as P.

Algorithm 2: DOT2DOT-1-LEVELTREE

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: Partitions P: a Steiner tree on each $p \in P$

1 $SP \leftarrow$ `FindBoundedPaths` (G, T)
2 mincost $\leftarrow \infty$
3 **foreach** $t \in T$ **do**
4 $N_t \leftarrow \{v \in T | SP(t, v)$ exists$\}$
5 Construct 1-level tree $L_1 = (V_p, E_p, d_p)$, where $V_p = t \cup N_t$, and for every $(t, v \in N_t) \in E_p$, $d_p(t, v) = \min(SP(t, v))$.
6 $P \leftarrow$ `ExpandPartition` (L_1, SP, T)
7 **if** $|V_p| < |T|$ **then**
8 $P \leftarrow P \cup$ DOT2DOT-1-LEVELTREE $(G, T \backslash V_p, SP)$
9 **else**
10 cost $\leftarrow$ cost of partition P by Eq. 13.3
11 **if** cost $<$ mincost **then**
12 mincost $\leftarrow$ cost, $minP \leftarrow P$
13 **return** $minP$

DOT2DOT-1-Level Tree

Our third method (Algorithm 2) builds a (set of) level-1 tree(s) from the transitive closure graph on the terminals. Simply put, we try each terminal as the root (3) and connect it to the other terminals with shortest paths on the transformed graph G' (4-5). If the selected root does not have shorter than length $\log |V|$ paths to all the terminals, a (set of) level-1 tree(s) is built on the remaining terminals (7-8). We return the least-cost (w.r.t. Eq. 13.3) tree(s) as the forest of

Steiner trees (10-12). Note that this heuristic algorithm provides a $|M|$-approximation to the optimal solution [Charikar et al., 1998].

Algorithm 3: DOT2DOT-k-LEVELTREE

Input: A graph $G = (V, E)$, terminals $T \subseteq V$
Output: Partitions P: a Steiner tree on each $p \in P$

```
SP ← FindBoundedPaths (G, T)
Construct candidate-graph G_c = (V_c, E_c): union of all top-3 shortest paths between all pairs of terminals.
mincost ← ∞
foreach t ∈ T do
    L_k ← ∅
    L_1 ← min-cost 1-level tree(s), one rooted at t
    foreach L_1 ∈ L_1 do
        S ← leaves(L_1), r ← root(L_1)
        if |S| < 2 then continue
        while S ≠ ∅ do
            L_k^p ← Partialk Tree(G_c, SP, r, S, k)
            L_k ← L_k ∪ L_k^p
            S ← S \ leaves(L_k^p)
    P ← ExpandPartition (L_k, SP, T)
    cost ← cost of partition P by Eq. 13.3
    if cost < mincost then
        mincost ← cost, minP ← P
return minP
```

DOT2DOT-k-Level Tree

Our final method generalizes the 1-level tree heuristic to k-level trees. The goal is to start with a (set of) 1-level tree(s) and successively refine each for lower cost. The main idea is the following: for a given tree with root r, find one or more intermediate nodes $v \in V$, such that the total cost from r to each v plus the costs of sub-trees rooted at v's (each with a mutual set of terminals as leaves) reduces the initial cost.

More specifically, we construct a k-level tree by a union of sub-trees, each consisting of the root r, exactly one intermediate node v, and all the descendants of this intermediate node. We refer to such sub-trees of level k as *partial k-tree*s, and we find them in a greedy manner described as follows. First, we find a *partial k-tree* L_k^p (if any) that reduces the cost for a subset of terminals that it spans. Next, we remove the terminals spanned by L_k^p from consideration and iterate this process until all terminals are spanned. Algorithm 3 formally describes the k-level tree heuristic.

Procedure 3: `Partial`k`Tree` (G_c, SP, r, S, k)

Input: A graph $G_c = (V_c, E_c)$, short paths SP, root r, a set S of nodes, level k
Output: partial k-level tree L_k^p with intermediate node $v \in \{V, \emptyset\}$ and $leaves(L_k^p) \subseteq S$

1 **foreach** $v \in V_c$ **do**
2 **if** $k = 2$ **then**
3 Sort $S = \{N_1, \ldots, N_{|S|}\}$ such that $SP(r, N_i) - SP(v, N_i) \geq SP(r, N_{i+1}) - SP(v, N_{i+1})$
4 Find $j \in \{1, \ldots, |S|\}$ that minimizes $cost(v) \leftarrow SP(r, v) + \sum_{i=1}^{j} SP(v, N_i)$
5 $S(v) \leftarrow \{N_1, \ldots, N_j\}$
6 **else**
7 $S' \leftarrow S$, $L_k^p \leftarrow (r, v)$, $cost(L_k^p) \leftarrow \infty$
8 **while** $S' \neq \emptyset$ **do**
9 $L_{k-1}^p \leftarrow$ `Partial`k`Tree`$(G_c, SP, v, S', k\text{-}1)$
10 **if** $cost(L_k^p) \leq cost(L_{k-1}^p)$ **then** exit while
11 $L_k^p \leftarrow L_k^p \cup L_{k-1}^p$
12 $S' \leftarrow S' \backslash leaves(L_{k-1}^p)$
13 $cost(v) \leftarrow cost(L_k^p)$, $S(v) \leftarrow leaves(L_k^p)$
14 Find v having minimum $cost(v)$
15 **return** partial k-level tree L_k^p with intermediate node v rooted at r and leaves $S(v)$

To complete the k-level heuristic, we need to describe its subroutine `Partial`k`Tree` which, given a root r and a set of terminals S, finds a low-cost partial k-level tree rooted at r and spanning (a subset of) the terminals. The `Partial`k`Tree` heuristic, as given in Procedure 3, is recursive: in order to find a partial k-level tree, we need to first find certain partial $(k\text{-}1)$-level trees that span all the given terminals (8-12). The base case is reached for level $k = 2$, which works as the following. For each candidate v for the intermediate node, we sort the terminals S according to the potential savings of inserting node v between the root r and each terminal in S (3). The potential saving for a terminal t_i stands for the difference between the shortest path lengths $w(r \rightsquigarrow t_i)$ and $w(v \rightsquigarrow t_i)$. These savings can take positive and negative values and are sorted in decreasing order. Finally, we include consecutive terminals from this sorted list while their inclusion decreases cost of the *partial* 2*-tree* (4-5).

13.2.2 Discussion

Before we conclude this section, we discuss some aspects of the k-level tree heuristic.

Firstly, note that in order to find good intermediate nodes v, Procedure `Partial`k`Tree` considers all the nodes in the given input graph (1). This raises two issues: 1) we would need the shortest paths from the root r to every v as well as from v to its descendants, and 2) the iteration would become computationally infeasible even for moderate size graphs. As a result, we provide `Partial`k`Tree` with the so-called *candidate graph* $G_c = (V_c, E_c)$. The candidate graph

consists of the union of the nodes and edges in all top-t shortest paths between every pair of terminals. Since G_c is much smaller than G' and as we have the shortest paths to the nodes in it computed, both issues are addressed. In the experiments we use $t = 3$, which can be adjusted depending on available computational resources. Considering a larger candidate graph may help finding lower cost trees while a smaller candidate graph provides lower running time as well as better visualization.

Secondly, notice that the cost of a *partial* 2*-tree* is computed by the sum of the shortest paths from root r to intermediate node v, and from v to a set of terminals (4). Using `FindBoundedPaths` recall that we find short paths from *marked* nodes to the nodes within $\log |V|$ distance. Since v might be an unmarked node, we might need the shortest path length from an *unmarked* node to a marked one. We find the shortest path from unmarked nodes in the candidate graph to marked nodes using the following Lemma.

Lemma 6 *The shortest path from i to j and the shortest path from j to i for all $i, j \in V'$ in the transformed graph G' contain exactly the same nodes with edges in reverse directions. Due to symmetry, the total weights of the reciprocal shortest paths obey the following equation.*

$$w(i \rightsquigarrow j) = w(j \rightsquigarrow i) - \log d(j) + \log d(i)$$

Proof Let $\{i \rightarrow v_1 \rightarrow v_2 \rightarrow \dots v_l \rightarrow j\}$ denote the shortest path from i to j. Notice that any path from i to j contains an out-edge of i to one of its neighbors, with weight $\log d(i)$. Thus, $w(i, v_1) = \log d(i)$. Let $R = \sum_{k=1}^{l-1} w(v_k, v_{k+1}) + w(v_l, j)$, such that $\log d(i) + R$ minimizes total length of the path. Since all out-edges of a node $v \in V'$ are equal, we can also write $w(v_1, i) + \sum_{k=l}^{2} w(v_k, v_{k-1}) = R$. So the reverse path from j to i has length $\log d(j) + R$. As R gives the shortest path length from i to j, it also gives the shortest from j to i. Hence the lemma holds. ∎

As a result, given that we know the shortest paths from terminals to other nodes $v \in V'$ within $\log |V|$, we can compute the shortest path from any such v to any terminal in constant time using Lemma 6. On the other hand, notice that for $k \geq 3$ in `Partial`k`Tree`, a descendant of an unmarked intermediate node might also be unmarked, in which case we would need the shortest paths between two unmarked nodes and run `FindBoundedPaths` starting from every new set of intermediate nodes. In our experiments, for speed we restrict ourselves to $k = 2$.

13.3 Experimental Results

In this section, we evaluate our proposed method; first we give intuitive results on synthetic examples, second we quantitatively compare the performance of the four proposed heuristic methods DOT2DOT-* (COMPONENTS, MINARBORESCENCE, 1-, and 2-LEVELTREE), and finally we provide case studies to show qualitative performance on real-world graphs.

13.3.1 Synthetic examples

We start by testing our method through three examples on a synthetic 100×100 grid graph, as well as one example on the well-known Zackary's karate graph [Zachary, 1977].

First, as shown in Figure 13.3(a), we select 8 marked (blue square) nodes relatively close to each other on the grid graph, and run all four of our approximation methods. We highlight (by orange bold edges) the minimum description cost tree found (orange nodes: connectors)—notice that it provides succinct connections among the marked nodes. Next we place 4 of the marked nodes farther apart in the grid graph, in which case our method successfully partitions the marked nodes and provides 2 connection trees for each as shown in Figure 13.3(b).

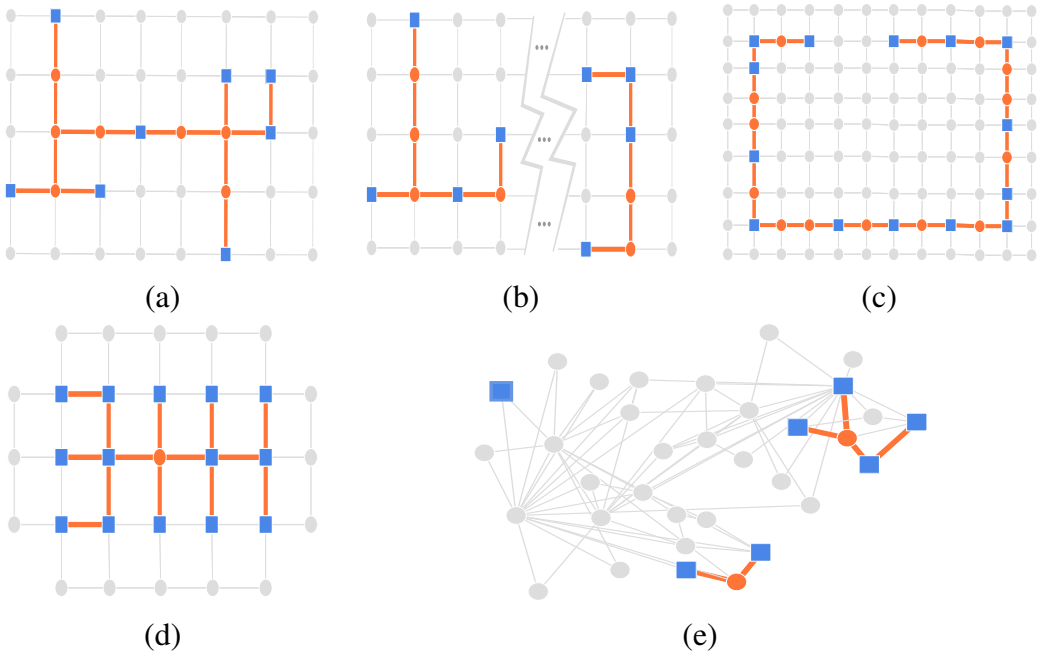


Figure 13.3: Synthetic examples: (a) 8 marked (square) nodes placed close to each other on a grid, (b) same 8 nodes in (a) spaced out on the grid, forming 2 parts, (c) connecting the dots, (d) recovering missing connector, (e) 7 marked nodes on Karate graph, forming 3 parts.

Second, we place the marked nodes intermittently on the grid as shown in Figure 13.3(c). Intuitively, human would connect these dots to form a rectangle—and so does our method. Figure 13.3(d) shows a set of marked nodes forming an almost full rectangle on the grid, except one node in the middle left unmarked. Notice that our method successfully recovers this 'left-behind' node as a significant connector.

Finally, we mark 7 nodes on the Karate graph as depicted in Figure 13.3(e). Our method partitions the nodes into 3 parts that are well separated through high degree hub nodes.

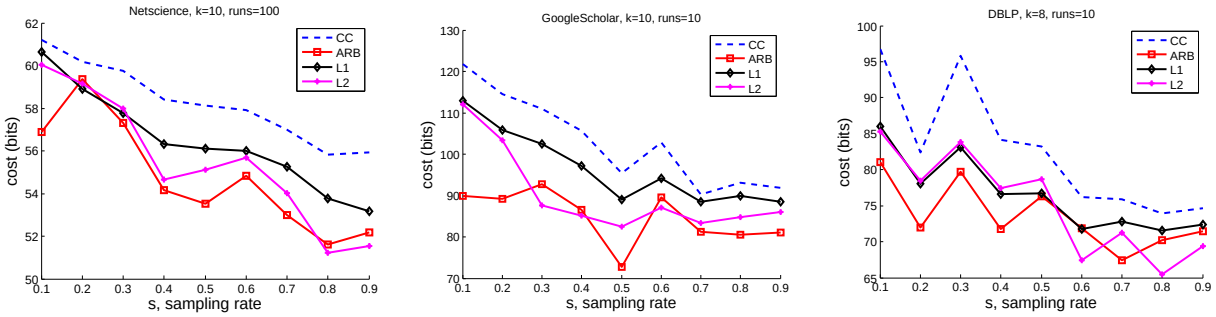
13.3.2 Comparing the heuristics

In this section, we aim to understand the average performance of the four approximation methods proposed in Section 13.2. To do so, we run simulations on three real graphs and compare their average description cost. In each simulated run we test the heuristics on a different set of marked nodes. The set of marked nodes are selected via random walk sampling, which we describe next. The dataset information is given in Table 13.1.

Name	$\|V\|$	$\|E\|$	**Description**
NETSCIENCE	379	914	Author collaborations
GSCHOLAR	83K	148K	Academic article citations
DBLP	329K	1094K	Author collaborations

Table 13.1: Dataset summary used for DOT2DOT.

To select a set of k marked nodes, we follow a random walk sampling scheme. First, we fix a sampling rate s. Then, we choose and mark a node at random in the given graph. Next we randomly visit $k' < k$ of its neighbors, and mark each neighbor with probability s. We continue this process until we have k marked nodes in total.



(a) NETSCIENCE (b) GSCHOLAR (c) DBLP

Figure 13.4: Comparison of our heuristics: total cost (bits) versus various sampling rates s to choose the marked nodes.

In Figure 13.4, we show average description cost (in bits) versus sampling rate s=$\{0.1, \ldots, 0.9\}$ of our simulations (k' is set to 3, k'=$1, 2$ gives similar results). We notice comparable performance results on all three graphs; the simplest heuristic COMPONENTS provides the costliest, while MINARBORESCENCE gives the most succinct description among the four methods on average. In addition, 2-LEVELTREE provides competitive results to MINARBORESCENCE, and outperforms 1-LEVELTREE as would be expected.

In addition, notice the downward trend of the cost for increasing sampling rate. This is expected, as for higher s the marked nodes are chosen among closer nodes. This also explains the relatively larger gap in performance between 2-LEVELTREE and MINARBORESCENCE for small s, as for farther apart nodes a higher than 2 level tree might be required.

Finally, we give the running time of our methods in Figure 13.5. We notice that due to the iterative nature of looking for good intermediate nodes, k-LEVELTREE heuristics (L1 and L2 respectively) take longer than the others. At the same time, L2 completes in about 50 seconds on our largest graph DBLP and can be further sped up by providing it with a smaller candidate graph. Since FINDBOUNDEDPATHS takes the most considerable time in our framework, we propose to run all heuristics and report the best result with the minimum cost.

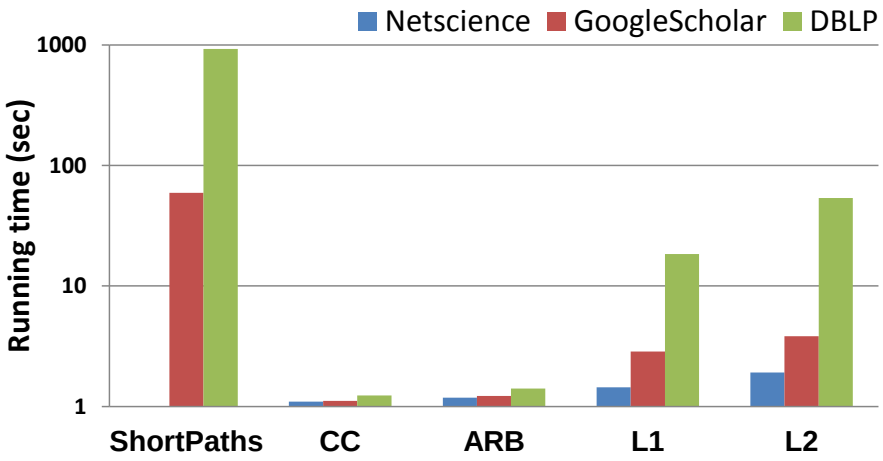


Figure 13.5: Run time of proposed methods on three real graphs.

13.3.3 Case studies on real graphs

Our method can provide useful insight about the connections and connectors among a group of nodes in a given graph. As such, we develop DOT2DOT as an interactive tool that aids in visualization and sensemaking. In this section, we provide qualitative analysis on two large real-world graphs.

Authors in DBLP

We first employ DOT2DOT among authors from various fields in computer science in the DBLP dataset. In particular, we select 2 major conferences in certain fields and mark the top 10 authors from each, who have the most number of papers appearing at that particular conference.

In Figure 13.6(a) we show the connection tree our method finds among respective authors from VLDB and CHI which are in the fields of databases and human computer interaction, respectively. Notice that the authors from these two fields are considerably well separated in the tree. We also observe a connector node among the communities: Duen Horng Chau, a PhD student at Carnegie Mellon (also a co-author) whose research focuses on bridging data mining, human computer interaction, and visualization.

We obtain similar results for the authors from RECOMB (computational biology) and KDD (data mining and machine learning) in Figure 13.6(b). Notice the simple connections among the authors within the same field and otherwise well separated communities. The connector is David Heckerman, the director of the eScience team at Microsoft Research whose work focuses on learning from, and analysis of biological and medical data.

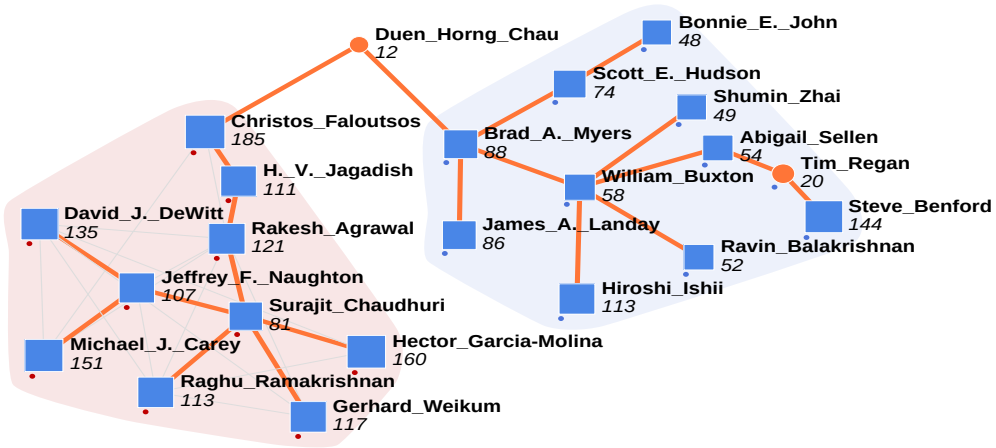
In the next example from DBLP we look at authors from NIPS (machine learning) and PODS (database systems), 5 authors from each. DOT2DOT connects the authors from each field through a few connectors as shown in Figure 13.6(c). Notice there are two parts in our partitioning; which suggests that authors from these two communities are sufficiently apart in the graph.

Articles in GoogleScholar

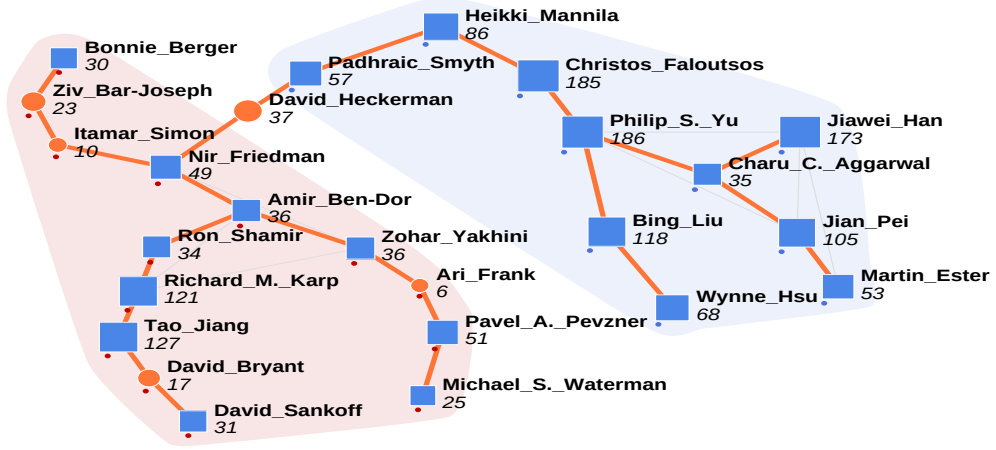
We next apply DOT2DOT to summarize academic articles on certain topics from GSCHOLAR. By their titles, we mark nodes in the citation graph containing specified keywords.

In Figure 13.7(a) we show 8 marked articles containing both 'large graphs' and 'visual' keywords in their title. Our visualization highlights the resulting partitioning on the candidate graph we generate by the union of top-3 shortest paths among the marked nodes. We find that DOT2DOT partitions the marked nodes into 4 parts, 3 of which are singletons. The connection tree for the largest part provides a concise summary for the 5 nodes in this part, with only two connectors. Also notice that the singleton nodes are at least three hops apart from this tree.

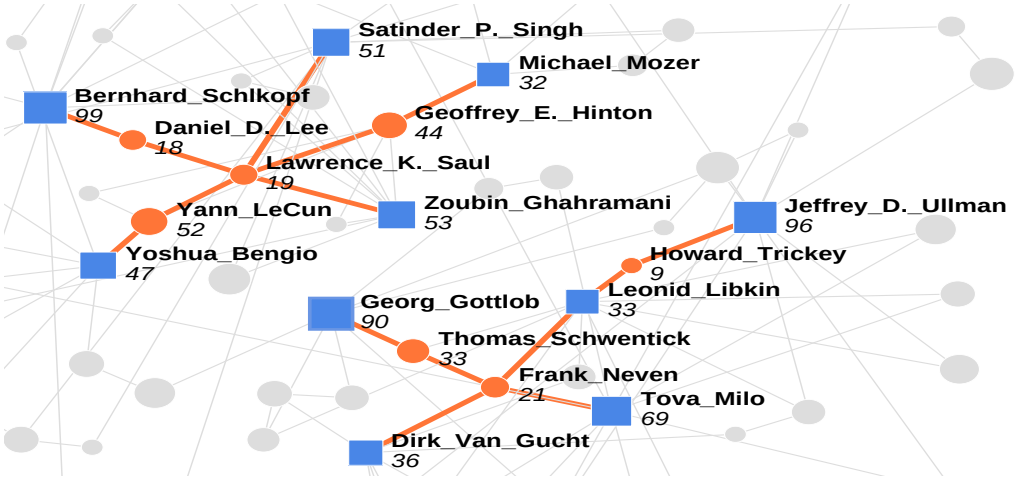
In the second example, we mark a random set of the articles with the keywords 'association rule' together with 'visual' and/or 'text' in their title. Here, we expect the articles to naturally split into two groups, one group on visualizing association rules and the other on association rule mining in text data. Our results, as shown in Figure 13.7(b), agree with our intuition: visualization articles are linked up through a meaningful connector and 'text' articles form separate multiple parts. It is interesting to observe that they can not be glued together by a single connection tree as in the former scenario.



(a) DBLP: VLDB vs. CHI

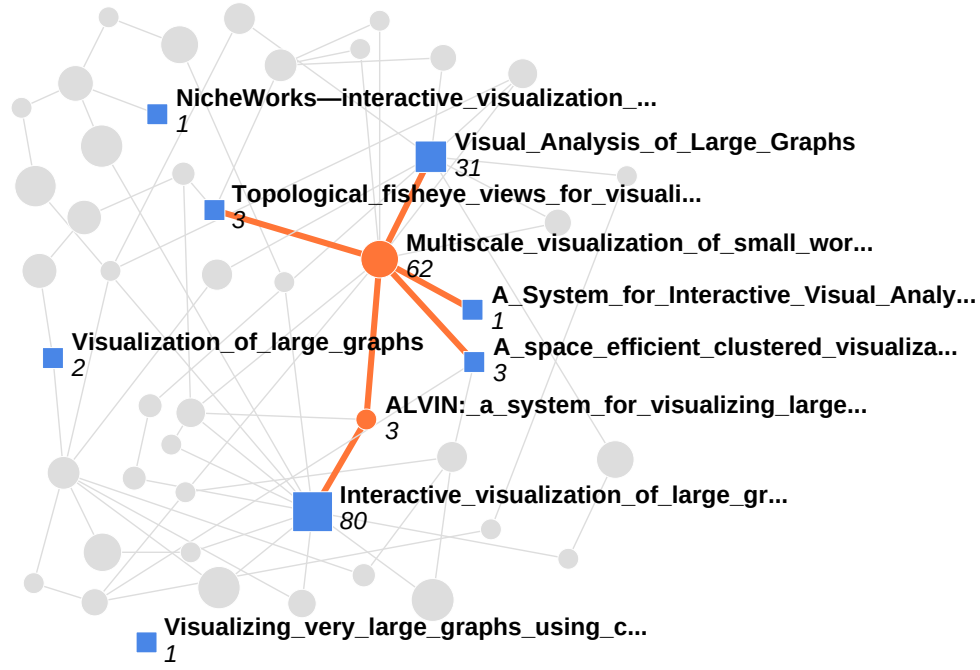


(b) DBLP: RECOMB vs. KDD

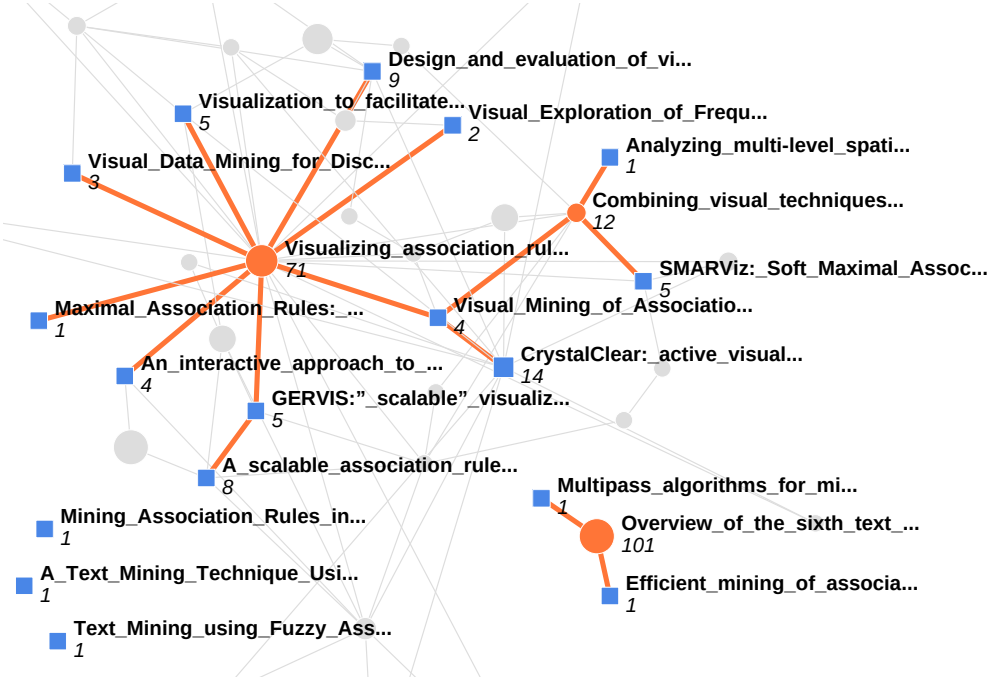


(c) DBLP: NIPS vs. PODS

Figure 13.6: Connection trees among authors from DBLP with most number of papers at specified conferences. (a) connector Duen Horng Chau: bridging data mining and human-computer interaction, (b) connector David Heckerman: mining biomedical data, (c) two trees sufficiently apart.



(a) GSCHOLAR: 'large graphs', 'visual'



(b) GSCHOLAR: 'association rule', 'visual', 'text'

Figure 13.7: Connection trees among articles from GSCHOLAR with the specified keywords in their title. (a) one tree summarizing 5 marked nodes while singletons reside farther apart, (b) one tree summarizing 'visualization' articles while 'text' articles are scattered.

13.4 Summary of contributions

In this chapter, we introduced DOT2DOT[1], a novel method to 'describe' a set of marked nodes in a graph; by grouping 'close-by' nodes together as well as providing 'concise' connections among the nodes in each group.

We motivated this problem and our solutions under the setting where the marked nodes are thought as anomalous nodes reported by an anomaly detection algorithm, like ODDBALL. On the other hand, we note that our problem framework is general and can find applications in various other settings such as gene pathway discovery, product adoption summarization and understanding of adoption correlation with graph structure, and visualization.

Our main contributions are listed as the following.

- *Problem formulation:* We formalize the problem of 'describing a set of chosen nodes in a graph' as an encoding problem. Our formulation exploits the Minimum Description Length principle: the best description is the one for which we need the least number of bits to encode the marked nodes. As such, our method does not require any user-specified parameters such as the number of groups.
- *Fast algorithms:* We show that our problem formulation has connections to the directed Steiner tree problem [Charikar et al., 1998], and that finding the minimum cost (in bits) solution is NP-hard. We propose fast solutions for large graphs.
- *Experiments on real graphs:* Experiments on real academic collaboration and citation as well as synthetic graphs demonstrate the effectiveness of DOT2DOT in discovering good connectors and connections that agree with human intuition.

[1] Source code of DOT2DOT: `www.cs.cmu.edu/~lakoglu/#code`

Part IV

Conclusion and Future directions

Chapter 14

Concluding Remarks

To review, we first summarize our major contributions and then give the impact of this thesis. We conclude with possible interesting future research directions.

14.1 Research Summary and Contributions

In this thesis, we focused on three interrelated aspects of the study of networks: (1) we analyzed the structure and dynamics of real-world networks to understand the regularities they exhibit, (2) using the understanding of how networks form and evolve we created several generative models, and finally (3) we developed new techniques to spot anomalies in network data in various settings. For a list of publications this work appeared in, see Appendix A.

14.1.1 New Patterns in Network Topology and Human Communications

In Part I, we analyzed numerous large (millions of nodes and edges) real-world networks from many diverse domains including political campaign donations, computer network traffic, blog citations, social and Web networks, as well as human communication networks. While previous work focused on static unweighted networks, we shifted our focus to *dynamic* and *weighted* networks. We discovered surprising new patterns, which we briefly review here. We refer to §3.3 and §4.4 for more details.

- **"Rebel probability" and oscillating/constant-size connected components:** We are *among the first* to focus on next-largest connected components (NLCCs) and their dynamics. In particular, we showed that (1) NLCCs look like chain graphs; (2) real graphs exhibit a *"gelling" point* at which the small components merge and a giant connected component (GCC) emerges; (3) after the gelling point, secondary and tertiary components cannot grow beyond a certain size beyond which they get absorbed to the GCC; (4) *graph fractal dimension* of components is stable over time; and (5) *rebel probability* —that a newcomer node will *not* join the GCC— drops exponentially with its degree and increases linearly

with the fraction-size of NLCCs. The "rebel" probability (*microscopic* dynamics) explains the component sizes oscillating around constant-size (*macroscopic* dynamics).

- **"Fortification effect" and other non-linear weighted-network patterns:** We are the *first* to focus on weights on graph edges and their dynamics. In particular, we showed that (1) total weight of a node and its degree follow power laws, and weight between two nodes follows a gravitational-force-like relation with the total weights of those nodes; (2) total weight of a graph grows superlinearly with number of its edges over time, which we call *fortification*; (3) weight additions over time are bursty; and (4) principal eigenvalues and number of edges of a graph follow a power law relation over time.
- **Power-laws in human communications:** In order to understanding human communication patterns, we analyzed *billions* of human communication (phone call, SMS, and instant message) records of millions of users over several months. (1) We found that the number of maximal cliques a node participates in follows a power-law relation with its degree; which translates to "popularity" growing superlinearly with the number of contacts. (2) We proposed the 3PL function, which models the reciprocity of user pairs very well, and better than bivariate Pareto and Yule. We observed that reciprocity is higher (a) for mutual pairs with larger local network overlap, that is, for people with more common friends; and (b) for mutual pairs with larger degree-similarity, that is, for people with similar number of contacts. (3) We also proposed the TLAC distribution, which fits a vast majority of individual phone call durations the best, much better than lognormal and exponential.

14.1.2 Realistic Generative Models of Networks

In Part II, we focused on the problem of how to build a generative model that could produce realistic-looking synthetic networks. We presented two generators for general graphs that mimic a *long* list of topological properties of real-world networks. We also developed an agent-based generator for human communication graphs, that mimic human behavior well. We review these models briefly next. We refer to §6.3 and §7.2 for more details.

- **Realistic generative models for graphs:** Our first model Recursive Tensor Model (RTM) is a simple, recursive generator based on Kronecker tensor multiplication. We rigorously proved that RTM produces several desired characteristics, such as bursty traffic, i.e. bursty weight additions. On the other hand, it creates multinomials rather than power laws, for example in degree distributions. Later, we proposed Random Typing Graphs (RTG) based on a simple 'random typing' procedure which is shown to mimic natural behavior very well. RTG meets all the desirable properties, being realistic (matching *all eleven* patterns), simple, parsimonious, flexible, and fast.
- **Utility-driven model for human communications:** We also designed an intuitive graph generator PaC, for modeling human communication behavior. In this model, each node (a) uses only local information, and (b) uses no randomness, but instead tries to maximize a well-defined utility function. This agent-based model allows us to understand the local mechanisms forming communication networks and answer what-if scenarios. Based on the

utility function of PaC, we can explore what the impact of certain changes (eg. increase in price-per-minute, flat call rate) would be on the structure and evolution of the network.

14.1.3 Tools for Anomaly and Event Detection in Networks

In Part III, we developed five novel methods for anomaly detection, each addressing the problem in a different setting. In particular, the methods address the problem for (1) plain unlabeled graphs, (2) binary- and (3) categorical-attributed graphs, (4) time-varying graphs, and finally (5) sensemaking and visualization of anomalies. We briefly highlight our contributions below and respectively refer to §9.3, §10.3, §11.3, §12.3, and §13.4.

- **Automatic anomaly detection in plain graphs:** We proposed a novel method, called ODDBALL, to detect outlier nodes in *graph* data, rather than in a collection of data points. The previous state-of-the-art had several limitations, such as the assumption of the existence of node labels and ground truth information. We developed our method so as to avoid these assumptions and to (1) work for unlabeled and weighted graphs, and (2) operate in a completely unsupervised fashion.
- **Parameter-free mining and clustering in attributed graphs:** We introduced a novel clustering model, called PICS, to find groups of nodes in an attributed graph with (1) *similar connectivity*, and (2) *attribute homogeneity*. The nodes deviating from the discovered patterns correspond to bridge-nodes with connections across clusters or outlier-nodes that do not belong well to any cluster. Two key advantages of our method are (1) it requires *no parameters* such as the number of clusters and similarity functions to be tuned, and (2) its running time scales linearly with total graph and attribute size.

  We also proposed a new approach, called COMPREX, for identifying anomalies in complex attributed graphs, that exploits pattern-based compression. The key features of this approach are (1) it builds the models directly from data and requires no parameters to be tuned, (2) it generalizes to a broad range of complex data, including graph, image and relational databases with various attributes, and (3) it proves effective on a broad range of datasets showing large improvements in both compression rate, as well as anomaly detection accuracy, outperforming its state-of-the-art competitors.
- **Automatic event detection and "attribution" in time-evolving graphs:** We developed an algorithm to quantify "change" for a graph growing/changing over time that flags those time points for which the change is significant as "events". Two key features of the algorithm are (1) it enables "anomaly attribution", that is, it points out those nodes that go through the highest degree of change, and (2) it works in an online fashion, by efficiently updating scores over time.
- **Sensemaking and visualization for anomalies:** We introduced a novel solution, called DOT2DOT, to visualize and summarize a given set of (anomalous) nodes in a graph. It groups 'close-by' nodes together and provides 'concise' connections among the nodes in each group, instead of simply listing them. This provides insight into what coarser groups

the nodes form, and thus facilitates summarization. It also shows good connection pathways between the given nodes as well as other key 'connector' nodes that play a crucial role in bridging the connections in between. We built an interactive visualization and exploration tool that enables end users specify their nodes of interest, visualize the discovered connection subgraphs, and explore further the neighborhood subgraphs around the given set of nodes on demand.

14.2 Community Impact

Our work in this thesis is making broader impact to academia and industry:

Our RTG generator raised a lot of interest with respect to its ties to earlier history of mathematics on random typing, and furthermore won the **Best Knowledge Discovery Paper award** in the Conference on Principles and Practice of Knowledge Discovery (PKDD) 2009. The software has been made publicly available and been used in the research community.

Our ODDBALL won the **Best Paper award** in the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD) 2010. It has also been integrated into Carnegie Mellon's Pegasus Tera-Scale Graph Mining system. Its stand-alone source code is made available and has been downloaded from all around the world. It has been a main component of the anomaly detection system developed for the ADAMS (Anomaly Detection at Multi Scale) collaborative research initiative supported by DARPA, for insider threat detection.

Our COMPREX and its proposed extension to the time-evolving setting led to **two patents** (filed with high-impact score) at IBM Research Labs. Compression has been primarily used in communications theory and databases, for reduced transmission cost and storage cost and increased throughput and query performance; our work is one of the few to explore compression for various data mining tasks. Moreover, our event detection system contributed to the Network Science Collaborative Technology Alliance (NS-CTA) supported by Army Research Labs.

14.3 Vision and Future Research Directions

Graphs are powerful and unified tools, effectively capturing long-range relations between the objects they represent. Not only is the theory of graphs abound with fascinating research problems, but also the practical usage of graphs is quite prevalent, as many real-world problems can be cast as graph problems. Therefore, I view graph mining as an exciting field to work in, with a promising future at the confluence of theory and practice.

All real data inherently contain patterns; intuitively even complex systems are often largely governed by a few underlying simpler mechanisms, which introduce considerable redundancy in the form of patterns. Therefore, it is intriguing and natural to attempt to find patterns in real data. Being closely related to pattern discovery, anomaly detection has *multi-billion dollar* applications

in finance (credit card, accounting fraud detection), health care (insurance claim fraud, disease outbreak, rare disease detection), security (network intrusion, insider threat detection), biology (neuroscience and genetics), and many more. Therefore, understanding how a complex system operates and detecting explanatory and emergent patterns (and anomalies therein) will continue to be a fascinating area in the future.

As real-world graphs like social networks and the Web continue growing and the technology to collect and store data continue advancing, huge amounts of data will become available. As a result *scalability* will become an even greater challenge. As older algorithms become obsolete due to their memory assumptions, mining patterns efficiently and spotting anomalies as early as possible in huge collections of data will require novel data mining approaches.

While I believe that pattern mining, anomaly detection, and scalability are promising research areas, finding solutions to domain-specific questions in other fields is also of interest. As the semantics captured by graphs differ, so do the problems they pose. Consequently, I believe that graph mining has a lot of *potential for cross-domain collaborations* in a variety of fields.

In general terms, it is interesting and crucial to *develop scalable methods that provide means to discover patterns, spot anomalies/events, and to summarize and interpret large complex graphs*. Below I elaborate on specific future research directions.

- **Patterns and anomalies in complex graphs** A vast majority of existing work on graphs is designed for plain undirected graphs. Real-world graphs, however, can appear in much richer forms (as directed weighted graphs, correlated with other graphs, with node and edge attributes, changing over time). For example Facebook data contains user-user interactions, user-wall post-user relations, user-product recommendations, user interests and demographics, all correlated and evolving over time. It remains an interesting open problem to study the inter-relations among *multiple correlated* graphs that represent more complex data, and to find patterns and anomalies in such complex graphs.
- **Scalability** As real-world data keep growing, their graph representations require peta-bytes of storage. This makes scalability a real bottleneck in algorithm design. While in the past the main goal of most graph mining algorithms was to be effective, the recent trend has been to build algorithms that are both effective and efficient on possibly disk-resident graphs that do not fit in the main memory. Future research could address this issue (1) by developing effective heuristics and ideally theoretically sound approximate algorithms which run in sub/linear time for large graphs, and (2) by designing algorithms so as to be able to exploit map-reduce type abstractions for parallelism.
- **Graph summarization and understanding** One main component of understanding data is to visualize it, however with the scale of real graphs visualization has become a real challenge. A good visualization requires a succinct summary of the graph. Future work could address this problem at two levels: (1) macro-level analysis involves summarizing complex graphs *at large*. For complex multi-facet graphs, the goal is to build novel clustering and compression models to offer the users a high-level understanding, and (2) micro-level analysis involves summarizing certain parts of *interest* in the graph (e.g., subsets of nodes, small neighborhoods, etc.).

Appendix A

List of publications

Part I: Patterns of Networks and
Part II: Generative Models of Networks

- L. Akoglu. M. McGlohon, and C. Faloutsos. RTM: Laws and a Recursive Generator for Weighted Time-Evolving Graphs. *IEEE International Conference on Data Mining (ICDM)*, 2008
- M. McGlohon, L. Akoglu, and C. Faloutsos. Weighted Graphs and Disconnected Components: Patterns and a Generator. *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, 2008
- N. Du, C. Faloutsos, B. Wang, L. Akoglu. Large human communication networks: patterns and a utility-driven generator. *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, 2009
- L. Akoglu, C. Faloutsos. RTG: A Recursive Realistic Graph Generator using Random Typing. *Data Mining and Knowledge Discovery Journal (DAMI)*, 2009
- U Kang, M. McGlohon, L. Akoglu, and C. Faloutsos. Patterns on the Connected Components of Terabyte-Scale Graphs. *IEEE International Conference on Data Mining (ICDM)*, 2010
- P. O. S. Vaz de Melo, L. Akoglu, C. Faloutsos, A. F. Loureiro. Surprising Patterns for the Call Duration Distribution of Mobile Phone Users. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2010
- L. Akoglu, P. O. S. Vaz de Melo, C. Faloutsos. Quantifying Reciprocity in Large Weighted Communication Networks. *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2012

Part III: Anomaly Detection

- L. Akoglu, M. McGlohon, C. Faloutsos. OddBall: Spotting Anomalies in Weighted Graphs. *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2010
- L. Akoglu, C. Faloutsos. Event Detection in Time Series of Mobile Communication

Graphs. *27th Army Science Conference*, 2010
- L. Akoglu, H. Tong, B. Meeder, C. Faloutsos. PICS: Parameter-free Identification of Cohesive Subgroups in Large Attributed Graphs. *SIAM International Conference on Data Mining (SDM)*, 2012
- L. Akoglu, H. Tong, J. Vreeken, C. Faloutsos. CompreX: Fast and Reliable Anomaly Detection in Categoric Data. *ACM International Conference on Information and Knowledge Management (CIKM)*, 2012
- L. Akoglu, J. Vreeken, H. Tong, D. H. Chau, C. Faloutsos. Islands and Bridges: Making Sense of Marked Nodes in Large Graphs. *Carnegie Mellon University Technical Report CMU-CS-12-124*, 2012

Bibliography

Lada Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *LinkKDD*, pages 36–43, 2005. 140

Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *SIGMOD*, pages 37–46, 2001. 108

Leman Akoglu and Christos Faloutsos. RTG: A recursive realistic graph generator using random typing. *Data Mining and Knowledge Discovery Journal (DAMI)*, 19(2):194–209, 2009. URL `http://www.cs.cmu.edu/~lakoglu/pubs/RTG_GraphGenerator.pdf`. 3, 75

Leman Akoglu and Christos Faloutsos. Event detection in time series of mobile communication graphs. In *27th Army Science Conference*, 2010. URL `http://www.cs.cmu.edu/~lakoglu/pubs/EVENTDETECTION_AkogluFaloutsos.pdf`. 3

Leman Akoglu, Mary Mcglohon, and Christos Faloutsos. RTM: Laws and a recursive generator for weighted time-evolving graphs. In *IEEE International Conference on Data Mining (ICDM)*, pages 701–706, 2008. URL `http://www.cs.cmu.edu/~lakoglu/pubs/RTM_LRGWTG_10.pdf`. 3, 9, 74, 75

Leman Akoglu, Mary McGlohon, and Christos Faloutsos. OddBall: Spotting anomalies in weighted graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 410–421, 2010. URL `http://www.cs.cmu.edu/~lakoglu/pubs/OddBall_cameraready.pdf`. 3

Leman Akoglu, Pedro O. S. Vaz de Melo, and Christos Faloutsos. Quantifying reciprocity in large weighted communication networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2012a. URL `http://www.cs.cmu.edu/~lakoglu/pubs/12-pakdd-reciprocity.pdf`. 3, 9

Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. PICS: Parameter-free identification of cohesive subgroups in large attributed graphs. In *SIAM International Conference on Data Mining (SDM)*, 2012b. URL `http://www.cs.cmu.edu/~lakoglu/pubs/12-sdm-pics.pdf`. 3

Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. CompreX: Fast and reliable anomaly detection in categoric data. In *ACM International Conference on Information and Knowledge Management (CIKM)*, 2012c. URL `http://www.cs.cmu.edu/~lakoglu/pubs/12-cikm-comprex.pdf`. 3

Leman Akoglu, Jilles Vreeken, Hanghang Tong, Duen Horng Chau, and Christos Faloutsos.

Islands and bridges: Making sense of marked nodes in large graphs. In *Carnegie Mellon University Technical Report CMU-CS-12-124*, 2012d. 3

Susanne Albers, Stefan Eilts, Eyal Even-Dar, Yishay Mansour, and Liam Roditty. On Nash equilibria for a network creation game. In *SODA*, pages 89–98. ACM Press, 2006. 76

Réka Albert, Hawoong Jeong, and Albert-László Barabási. Diameter of the World Wide Web. *Nature*, (401):130–131, 1999. 15, 74, 115

Nouf M. Kh. Alsudairy, Vijay V. Raghavan, Alaaeldin M. Hafez, and Hassan I Mathkour. Connection subgraphs: A survey. 11(17):3221–3232, 2011. 111

Reid Andersen and Kevin J. Lang. Communities from seed sets. In *WWW*, pages 223–232, 2006. 111

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *In FOCS*, pages 475–486, 2006. 16, 110, 111

Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A linear method for deviation detection in large databases. In *KDD*, 1996. 108

Albert-László Barabási. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume Books, April 2003. 15

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. 7, 17, 75, 76, 96, 102

V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Chichester, New York, 1994. 107

M. Basseville and I. Nikiforov. *Detection of Abrupt changes: Theory and Application*. Prentice-Hall, Inc., 1993. 109

Stephen Bay, Krishna Kumaraswamy, Markus G. Anderle, Rohit Kumar, and David M. Steier. Large scale detection of irregularities in accounting data. In *ICDM*, 2006. 104

Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, pages 16–24, 2008. 125

Robert Bell, Yehuda Koren, and Chris Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD*, pages 95–104, 2007. 8

Steve Bennett. Log-logistic regression models for survival data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 32(2):165–171, 1983. 60, 62

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *ICDT*, pages 217–235, 1999. 130

Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002. 111

Zhiqiang Bi, Christos Faloutsos, and Flip Korn. The dgx distribution for mining massive, skewed data. In *KDD*, pages 17–26, 2001. 12, 46

Béla Bollobás. *The Evolution of Random Graphs -the Giant Component*. Cambridge University Press, 2001. 25

Richard J. Bolton and David. Statistical fraud detection: A review. *Statistical Science*, 17: 235–255, 2002. 109

Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsa paras. Link analysis ranking: algorithms, theory, and experiments. *ACM Trans. Inter. Tech.*, 5(1):231–297, 2005. 8

M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying density-based local outliers. In *SIGMOD*, 2000a. 157

Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, pages 93–104, 2000b. 107, 118

Andrei Broder, Ravi Kumar, Farzin Maghoul1, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web: experiments and models. In *WWW*, 2000. 8, 21, 46, 104

B. E. Brodsky and B. S. Darkhovsky. *Nonparametric Methods in Change-Point Problems*. Kluwer Academic Publisher, 1993. 109

Alexandre Bronstein, Joydip Das, Marsha Duro, Rich Friedrich, Gary Kleyner, Martin Mueller, Sharad Singhal, and Ira Cohen. Using bayesian networks for detecting anomalies in Internet services. In *INM*, 2001. 108

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998. 109

C. Rodriguez-Sickert Cesar A. Hidalgo. The dynamics of a mobile phone network. *Physica A: Statistical Mechanics and its Applications*, 387(12):3017–3024, 2008. 9, 46

Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004. 108, 110

Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006. 76

Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88, 2004a. 110

Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. *SIAM Int. Conf. on Data Mining*, April 2004b. 15, 58, 74, 77

Soumen Chakrabarti, Byron E. Dom, S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, David Gibson, and Jon Kleinberg. Mining the web's link structure. *Computer*, 32(8):60–67, 1999. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.781636. 8

Pak K. Chan, Martine D. F. Schlag, and Jason Y. Zien. Spectral k-way ratio-cut partitioning and clustering. In *DAC*, pages 749–754, 1993. 110

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009. 108

Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200, 1998. 186, 190, 199

Duen Horng Chau, Shashank Pandit, and Christos Faloutsos. Detecting fraudulent personalities

in networks of online auctioneers. *PKDD*, pages 103–114, 2006. 104, 109

Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I. Hong, Brian Gallagher, and Tina Eliassi-Rad. Graphite: A visual query system for large graphs. In *ICDM Workshops*, pages 963–966, 2008. 111

Duen Horng Chau, Aniket Kittur, Jason I. Hong, and Christos Faloutsos. Apolo: interactive large graph sensemaking by combining machine learning and visualization. In *KDD*, pages 739–742, 2011. 111

Amitabh Chaudhary, Alexander S. Szalay, and Andrew W. Moore. Very fast outlier detection in large multidimensional data sets. In *DMKD*, 2002. 108

Ping Chen, Rebecca B. Buchheit, Jr, and Sue Mcneil. Web-vacuum: Web-based environment for automated assessment of civil infrastructure data. *Journal of Computing in Civil Engineering*, 19(2):137–147, 2005. 104

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14: 1396–1400, 1965. 188

Fan Chung, Linyuan Lu, and Van Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7(1):21–33, 2003. 15

Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009. 13, 46, 61

Brian Conrad and Michael Mitzenmacher. Power laws for monkeys typing randomly: the case of unequal probabilities. *IEEE Transactions on Information Theory*, 50(7):1403–1414, 2004. 84, 88

Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. In *IDA*, pages 105–114, 2001. 16

M. Crovella and A. Bestavros. Self-similarity in world wide web traffic, evidence and possible causes. *Sigmetrics*, pages 160–169, 1996. 90

Kaustav Das and Jeff G. Schneider. Detecting anomalous records in categorical datasets. In *KDD*, 2007. 108

Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley-Interscience, 2003. 104

Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, and Morteza Zadimoghaddam. The price of anarchy in cooperative network creation games. *CoRR*, abs/0902.1400, 2009. 76

I.S. Dhillon, S. Mallela, and D.S. Modha. Information-theoretic co-clustering. In *KDD*, 2003. 110

Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *ICDM*, 2001. 110

Nan Du, Christos Faloutsos, Bai Wang, and Leman Akoglu. Large human communication networks: patterns and a utility-driven generator. In *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, pages 269–278, 2009. URL `http://www.cs.cmu.edu/~lakoglu/pubs/kdd09-utility-model.pdf`. 3, 9, 75

N. Eagle, A. Pentland, and D. Lazer. Inferring social network structure using mobile phone data. *PNAS*, 2007. 140

William Eberle and Lawrence B. Holder. Detecting anomalies in cargo shipments using graph properties. In *ISI*, pages 728–730, 2006. 104, 108

William Eberle and Lawrence B. Holder. Discovering structural anomalies in graph-based data. In *ICDM Workshops*, pages 393–398, 2007. 108

Paul Erdos and Alfred Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960. 75, 79

Eyal Even-Dar, Michael J. Kearns, and Siddharth Suri. A network formation game for bipartite exchange economies. In *SODA*, pages 697–706, 2007. 76

A. Fabrikant, A. Luthra, E. N. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *PODC*, 2003. 76

Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *KDD*, 2004. 111

Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999. 7, 8, 12, 15, 46, 74

Illes J. Farkas, Imre Derényi, Albert-László Barabási, and Tamas Vicsek. Spectra of 'real-world' graphs: Beyond the semi-circle law. *Physical Review E*, 64, 2001. 15

Peter R. Fisk. The graduation of income distributions. *Econometrica*, 29(2):171–185, 1961. 60

Gary Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *KDD*, pages 150–160. 2000. 110, 111

Gary Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3), March 2002. 16

Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *KDD*, pages 401–410, 2005. 109

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*, pages 813–822, 2010. 108

D. Garlaschelli and M. I. Loffredo. Patterns of Link Reciprocity in Directed Networks. *Phys. Rev. Lett.*, 93:268701, 2004. 47

F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. pages 278–289, 2004. 109

Amol Ghoting, Matthew Eric Otey, and Srinivasan Parthasarathy. Loaded: Link-based outlier and anomaly detection in evolving data sets. In *ICDM*, 2004. 108

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. of Nat. Acad. of Sci.*, 99(12):7821–7826, 2002. 16, 58, 110, 111

Swapna S. Gokhale and Kishor S. Trivedi. Log-logistic software reliability growth model. In *HASE*, pages 34–41, 1998. 60

Maria E. Gomez and Vicente Santonja. Self-similarity in i/o workload: Analysis and modeling. In *WWC*, 1998. 90

M Granovetter. The strength of weak ties. *American Journal of Sociology*, 78:1360–1380, 1973. 8, 46

Steven D. Gribble, Gurmeet Singh Manku, Drew Roselli, Eric A. Brewer, Timothy J. Gibson, and Ethan L. Miller. Self-similarity in file systems. In *SIGMETRICS '98*, 1998. 90

G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001. 157

Peter D. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007. 106, 107, 109, 110, 132, 151, 152, 180, 183

Ziyu Guan, Jian Wu, Qing Zhang, Ambuj Singh, and Xifeng Yan. Assessing and ranking structural correlations in graphs. In *SIGMOD*, pages 937–948, 2011. 112

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. *Inf. Syst.*, 26(1):35–58, 2001. 108

Stephan Günnemann, Ines Färber, Brigitte Boden, and Thomas Seidl. Subspace clustering meets dense subgraph mining: A synthesis of two paradigms. In *ICDM*, 2010. 110

Junqiang Guo, Fasheng Liu, and Zhiqiang Zhu. Estimate the call duration distribution parameters in GSM system based on K-L divergence method. In *WiCom*, pages 2988–2991, 2007. 17, 59

Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *KDD*, pages 33–42, 1999. 109

Frederik Gustafsson. *Adaptive Filtering and Change Detection*. Wiley, 2000. 109

Daniel Hanisch, Alexander Zien, Ralf Zimmer, and Thomas Lengauer. Co-clustering of biological networks and gene expression data. In *ISMB*, pages 145–154, 2002. 110

D. M. Hawkins. *Identification of outliers*. Chapman and Hall, 1980. 107, 157

Jingrui He, Hanghang Tong, Spiros Papadimitriou, Tina Eliassi-Rad, Christos Faloutsos, and Jaime Carbonell. Pack: Scalable parameter-free clustering on k-partite graphs. In *SDM Workshop on Link Analysis, Counterterrorism and Security*, 2009. 110

Christopher S. Helvig, Gabriel Robins, and Alexander Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001. 186

Cesar A. Hidalgo and C. Rodriguez-Sickert. The dynamics of a mobile phone network. *Physica A: Statistical Mechanics and its Applications*, 387(12):3017 – 3024, 2008. 16

Shawndra Hill and Akash Nagle. Social network signatures: A framework for re-identification in networked data and experimental results. In *CASON*, pages 88–97, 2009. 16

Shawndra Hill, Foster J. Provost, and Chris Volinsky. Learning and inference in massive social networks. In Paolo Frasconi, Kristian Kersting, and Koji Tsuda, editors, *MLG*, 2007. 58

Tsuyoshi Idé and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *KDD*, pages 440–449, 2004. 109, 170

Wei Jin, Rohini K. Srihari, and Xin Wu. Mining concept associations for knowledge discovery through concept chain queries. In *PAKDD*, pages 555–562, 2007. 111

David S. Johnson, Shankar Krishnan, Jatin Chhugani, Subodh Kumar, and Suresh Venkatasub-

ramanian. Compressing large boolean matrices using reordering techniques. In *VLDB*, pages 13–23, 2004. 134

R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1998. 107

José Fernando Rodrigues Jr., Hanghang Tong, Agma J. M. Traina, Christos Faloutsos, and Jure Leskovec. Gmine: A system for scalable, interactive graph visualization and mining. In *VLDB*, pages 1195–1198, 2006. 111

U. Kang, Mary McGlohon, Leman Akoglu, and Christos Faloutsos. Patterns on the connected components of terabyte-scale graphs. In *IEEE International Conference on Data Mining (ICDM)*, pages 875–880, 2010a. URL `http://www.cs.cmu.edu/~lakoglu/pubs/patternsindisconnectedcomps.pdf`. 3, 9

U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations. In *SDM*, pages 548–558, 2010b. 15

George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proc. of Supercomputing*, pages 1–13, 1998. 110, 111

L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990. 108

Yoshinobu Kawahara, Takehisa Yairi, and Kazuo Machida. Change-point detection in time-series data based on subspace identification. In *ICDM*, pages 559–564, 2007. 109

Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *VLDB*, pages 180–191, 2004. 109

Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–17, 1999. 15, 74

Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998. 107

Petri Kontkanen and Petri Myllymki. MDL histogram density estimation. In *AISTAT*, 2007. 160

Kleanthis-Nikolaus Kontonasios and Tijl De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *SDM*, 2010. 109

Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and extracting proximity in networks. In *KDD*, 2006. 111

S. Kotz, N. Balakrishnan, and Norman L. Johnson. Continuous multivariate distributions, volume 1, models and applications, 2nd edition. 2000. 51

Scheinerman E.R. Kraetzl M., Nickel C. Random dot product graphs: a model for social networks. In *Preliminary Manuscript*, 2005. 76

Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Core algorithms in the CLEVER system. *ACM Trans. Inter. Tech.*, 6(2):131–152, 2006. 8

Nikolaos Laoutaris, Laura J. Poplawski, Rajmohan Rajaraman, Ravi Sundaram, and Shang-Hua

Teng. Bounded budget connection (bbc) games or how to make friends and influence people, on a budget. In *PODC*, 2008. 76

J. F. Lawless and Jerald F. Lawless. *Statistical Models and Methods for Lifetime Data (Wiley Series in Probability & Mathematical Statistics)*. John Wiley & Sons, January 1982. ISBN 0471085448. 60, 61

Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *KDD*, pages 157–166, 2005. 108

Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003. 8

Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication. In *PKDD*, Porto, Portugal, 2005a. 75, 76, 77

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of ACM SIGKDD*, pages 177–187, Chicago, Illinois, USA, 2005b. ACM Press. 7, 16, 17, 26, 74, 76, 79, 96

Jure Leskovec, Susan Dumais, and Eric Horvitz. Web projections: learning from contextual subgraphs of the web. In *WWW*, pages 471–480, 2007a. 111

Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie Glance, and Matthew Hurst. Cascading behavior in large blog graphs: Patterns and a model. In *Society of Applied and Industrial Mathematics: Data Mining*, 2007b. 8, 17, 46

Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *KDD*, pages 462–470, 2008. 104

M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 1993. 106

Anthony Liekens, Jeroen De Knijf, Walter Daelemans, Bart Goethals, Peter De Rijk, and Jurgen Del Favero. BioGraph: unsupervised biomedical knowledge discovery via automated hypothesis generation. *Genome Biology*, 12(6), 2011. 179

J. Lin, E. J. Keogh, S. Lonardi, and B. Y. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pages 2–11, 2003. 160

Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and Philip S. Yu. Mining behavior graphs for "backtrace" of noncrashing bugs. In *SDM*, 2005. 108

Bo Long, Zhongfei Zhang, Xiaoyun Wu, and Philip S. Yu. Spectral clustering for multi-type relational data. In *ICML*, volume 148, pages 585–592, 2006. 110

Talat Mahmood. Survival of newly founded businesses: A log-logistic model approach. *JournalSmall Business Economics*, 14(3):223–237, 2000. 60

M. Mampaey and J. Vreeken. Summarising data by clustering items. In *ECML PKDD*, 2010. 110

B. Mandelbrot. An informational theory of the statistical structure of language. *Communication Theory*, 1953. 84

F. J Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951. 63

Mary Mcglohon, Leman Akoglu, and Christos Faloutsos. Weighted graphs and disconnected components: Patterns and a generator. In *ACM Special Interest Group on Knowledge Discovery and Data Mining (SIG-KDD)*, pages 524–532, 2008. URL `http://www.cs.cmu.edu/~lakoglu/pubs/kdd08-weighted-disconnected.pdf`. 3, 9, 74, 76, 96, 116

Vardges Melkonian. New primal-dual algorithms for steiner tree problems. *Computers and Operations Research*, 34:2147–2167, 2007. 186

C.D. Sinclair M.I. Ahmad and A. Werritty. Log-logistic flood frequency analysis. *Journal of Hydrology*, 98:205–224, 1988. 60

P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. 20 (10):1348–1362, 2008. 110

M. Mihail and C. Papadimitriou. The eigenvalue power law, 2002. 15

S. Milgram. The small-world problem. *Psychology Today*, 2:60–67, 1967. 7

G. A. Miller. Some effects of intermittent silence. *American Journal of Psychology*, 70:311–314, 1957. 84, 86

Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007. 140

Michael Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1(3):305–333, 2003. 100, 102

Alan L. Montgomery and Christos Faloutsos. Identifying web browsing trends and patterns. *IEEE Computer*, 34(7):94–95, July 2001. 13

H. D. K. Moonesinghe and Pang-Ning Tan. Outrank: a graph-based outlier detection framework using random walk. *International Journal on Artificial Intelligence Tools*, 17(1), 2008. 108

Flavia Moser, Recep Colak, Arash Rafiey, and Martin Ester. Mining cohesive patterns from graphs with feature vectors. In *SDM*, pages 593–604, 2009. 110

Amit A. Nanavati, Siva Gurumurthy, Gautam Das, Dipanjan Chakraborty, Koustuv Dasgupta, Sougata Mukherjea, and Anupam Joshi. On the structural properties of massive telecom call graphs: findings and implications. In *CIKM*, pages 435–444, New York, NY, USA, 2006. 17

John Nash. Non-Cooperative Games. *The Annals of Mathematics*, 54(2):286–295, 1951. 76

M. E. J. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46 (5):323–351, 2005. 12, 13, 15, 32, 44, 74

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004. 92

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001. 110

Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994. 108

Viet-An Nguyen, Ee-Peng Lim, Hwee-Hoon Tan, Jing Jiang, and Aixin Sun. Do you trust to get trust? a study of trust reciprocity behaviors and reciprocal trust prediction. In *SDM*, pages 72–83, 2010. 46

Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003. 108

Ronald Nussbaum, Abdol-Hossein Esfahanian, and Pang-Ning Tan. Clustering social networks using distance-preserving subgraphs. In *ASONAM*, 2010. 8, 46

Jukka-Pekka Onnela, Jari Saramáki, Jórkki Hyvónen, Gábor Szabó, M Argollo de Menezes, Kimmo Kaski, Albert-László Barabási, and János Kertész. Analysis of a large-scale weighted network of one-to-one human communication. *New Journal of Physics*, 9(6):179, 2007a. 16, 58

Jukka-Pekka Onnela, Jari Saramáki, Jórkki Hyvónen, Gábor Szabó, David Lazer, Kimmo Kaski, János Kertész, and Albert-László Barabási. Structure and tie strengths in mobile communication networks. *PNAS*, 104(18):7332–7336, 2007b. 16

C. R. Palmer, P. B. Gibbons, and C. Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *SIGKDD*, Edmonton, AB, Canada, 2002. 11

Shashank Pandit, Duen H. Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*, pages 201–210, 2007. 8, 104

Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. LOCI: Fast outlier detection using the local correlation integral. In *ICDE*, 2003. 107

David M. Pennock, Gary W. Flake, Steve Lawrence, Eric J. Glover, and Lee C. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, 2002. 76

S Unnikrishna Pillai, Torsten Suel, and Seunghun Cha. Perron-Frobenius theorem. *IEEE Signal Processing Magazine*, 22(1):713–698, 1912. 171

B. Aditya Prakash, Hanghang Tong, Nicholas Valler, Michalis Faloutsos, and Christos Faloutsos. Virus propagation on time-varying networks: Theory and immunization algorithms. In *ECML/PKDD*, pages 99–114, 2010. 28

Cartic Ramakrishnan, William H. Milnor, Matthew Perry, and Amit P. Sheth. Discovering informative connection subgraphs in multi-relational graphs. 7(2):56–63, 2005. 111

William J. Reed and Murray Jorgensen. The double pareto-lognormal distributiona new parametric model for size distributions, 2004. 100

M. Richardson and P. Domingos. Mining the network value of customers. In *KDD*, pages 57–66, 2001. 13

Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002. 8

Jason Riedy, David A. Bader, Karl Jiang, Pushkar Pande, , and Richa Sharma. Detecting communities from given seeds in social networks. *Technical Report GT-CSE-11-01*, 2011. 111

J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983. 133

Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *KDD*, pages 737–746, 2009. 8, 46

Manfred Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W.H. Freeman and Company, New York, 1991. 13

Michael F. Schwartz and David C. M. Wood. Discovering shared interests among people using graph analysis of global electronic mail traffic. *Communications of the ACM*, 36:78–89, 1992. 16

Karlton Sequeira and Mohammed Javeed Zaki. Admit: anomaly-based data mining for intrusions. In *KDD*, pages 386–395, 2002. 109

Mukund Seshadri, Sridhar Machiraju, Ashwin Sridharan, Jean Bolot, Christos Faloutsos, and Jure Leskovec. Mobile call graphs: beyond power-law and lognormal distributions. In *KDD*, pages 596–604, 2008. ISBN 978-1-60558-193-4. 17, 46, 58, 104

Petteri Sevon and Lauri Eronen. Subgraph queries by context-free grammars. *J. Integrative Bioinformatics*, 5(2), 2008. 111

Dafna Shahaf and Carlos Guestrin. Connecting the dots between news articles. In *KDD*, pages 623–632, 2010. 111

Jitesh Shetty and Jafar Adibi. Discovering important nodes through graph entropy: The case of enron email database. In *LinkKDD*, pages 74–81, 2005. 108

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000. 110

Peter Shoubridge, Miro Kraetzl, Walter D. Wallis, and Horst Bunke. Detection of abnormal change in a time series of graphs. *Journal of Interconnection Networks*, 3(1-2):85–101, 2002. 109

W Shrum, N. H. Cheek, and S. M. Hunter. Friendship in school: Gender and racial homophily. *Sociology of Education*, 61:227–239, 1988. 179

Arno Siebes, Jilles Vreeken, and Matthijs van Leeuwen. Item sets that compress. In *SDM*, 2006. 108, 109, 148, 152

G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Powerlaws and the as-level internet topology, 2003. 15, 74

G. Siganos, S. L. Tauro, and M. Faloutsos. Jellyfish: a conceptual model for the as internet topology. *Journal of Communications and Networks*, 2006. 11

Koen Smets and Jilles Vreeken. The odd one out: Identifying and characterising anomalies. In *SDM*, 2011. 108, 109, 148, 157

Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008. 111

Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graph. *ICDM*, November 27-30 2005. 104, 108

Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696, 2007. 109, 110

Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *KDD*, pages 717–726, 2007. 8, 46

Nikolaj Tatti and Jilles Vreeken. Finding good itemsets by packing data. In *ICDM*, 2008. 109

SL Tauro, C. Palmer, G. Siganos, and M. Faloutsos. A simple conceptual model for the Internet topology. 2001. 11

Stephen Hardy Tejinder S. Randhawa. *Network Management in Wired and Wireless Networks*. Springer-Verlag New York, LLC, 2003. 59

Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In *SIGMOD*, pages 567–580, 2008. 110

Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006. 111

Hanghang Tong, Christos Faloutsos, and Yehuda Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007. 111

Hanghang Tong, Spiros Papadimitriou, Christos Faloutsos, Philip S. Yu, and Tina Eliassi-Rad. Basset: Scalable gateway finder in large graphs. In *PAKDD*, pages 449–463, 2010. 112

Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM*, 2008. 15, 74, 125

`http://www-personal.umich.edu/~mejn/netdata/`. *Book citations on U.S. politics*. 140

Nicholas Valler, B. Aditya Prakash, Hanghang Tong, Michalis Faloutsos, and Christos Faloutsos. Epidemic spread in mobile ad hoc networks: Determining the tipping point. In *Networking (1)*, pages 266–280, 2011. 28

Pedro O. S. Vaz de Melo, Leman Akoglu, Christos Faloutsos, and Antonio Alfredo Ferreira Loureiro. Surprising patterns for the call duration distribution of mobile phone users. In *ECML PKDD*, 2010. 3, 9

Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011. 109, 148

Q. H. Vuong. Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica*, 57:307–333, 1989. 52

Mengzhi Wang, Tara Madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE*, pages 507–516, 2002. 13, 77, 91

Ye Wang, Srinivasan Parthasarathy, and Shirish Tatikonda. Locality sensitive outlier detection: A ranking driven approach. In *ICDE*, pages 410–421, 2011. 108

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, (393):440–442, 1998. 7, 75, 76, 79

H. S. Wilf. The eigenvalues of a graph and its chromatic number. *J. London Math. Soc.*, 42:

330:332, 1967. 29

D. Willkomm, S. Machiraju, J. Bolot, and A. Wolisz. Primary users in cellular networks: A large-scale measurement study. In *DySPAN*, pages 1–11, 2008. 17, 58, 61

Weng-Keen Wong, Andrew W. Moore, Gregory F. Cooper, and Michael M. Wagner. Bayesian network anomaly pattern detection for disease outbreaks. In *ICML*, 2003. 108

Evdokia Xekalaki. The bivariate yule distribution and some of its properties. *Statistics*, 17(2): 311–317, 1986. 51

Rongjing Xiang, Jennifer Neville, and Monica Rogati. Modeling relationship strength in online social networks. In *WWW*, pages 981–990, 2010. 46

Kenji Yamanishi and Jun ichi Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *KDD*, pages 676–681, 2002. 109

Xintian Yang, Sitaram Asur, Srinivasan Parthasarathy, and Sameep Mehta. A visual-analytic toolkit for dynamic interaction graphs. In *KDD*, pages 1016–1024, 2008. 46

S. J. Young and E. R. Scheinerman. Random dot product graph models for social networks. In *WAW*, pages 138–149, 2007. 76

W.W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977. 193

Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997. 186

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996. 108

Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009. 110

Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Clustering large attributed graphs: An efficient incremental approach. In *ICDM*, pages 689–698, 2010. 110

G. K. Zipf. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press, 1932. 84