# Anomaly Detection in Graphs and Time Series: Algorithms and Applications

Bryan Hooi

April 2019
CMU-ML-19-100

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

**Thesis Committee:**
Christos Faloutsos, Chair
David Choi
Leman Akoglu
Vipin Kumar, University of Minnesota

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



This research was supported by the National Science Foundation under Grant No. CNS-1314632, IIS-1408924, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, and in part by the Defense Advanced Research Projects Agency (DARPA) under award no. FA8750-17-1-0059 for the RADICS program. 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:** anomaly detection, unsupervised, graph, time series, dynamic graphs, fraud detection, tensor

*To my family, who have always been there for me. I love you all.*

## Abstract

How can we detect fraudsters in large online review networks, or power grid failures using electrical sensor data? With the increasing availablility of web-scale graphs and high-frequency sensor data, anomaly detection in massive datasets has seen growing focus. Social networks such as Facebook and Twitter contain up to billions of users. Similarly, large scale sensor data includes networks of traffic speed detectors that span freeway systems in major metropolitan areas, networks of voltage sensors spanning the electrical power grid, as well as numerous types of industrial, weather and environmental sensors. These datasets have created an increasing need for scalable algorithms that can automatically analyze this data and flag users or events which are anomalous or of interest.

This thesis focuses on these problems, by developing scalable, principled algorithms that detect unusual behavior or events. We focus on the use of *connectivity* and *temporal* information, which allow us to detect anomalies in large graphs such as social networks, as well as for sensor datasets such as traffic data, which contain multiple sensors varying over time, that are also arranged on a graph.

First, we focus on *static graphs*, in which only connectivity information is present. For example, how can we detect fraudsters in a user-product review graph, or a Twitter follower-followee graph? We first propose a probabilistic approach that evaluates how surprising a dense subgraph is, while avoiding Erdos-Renyi assumptions of existing methods. We then consider how to detect dense subgraphs in a way that prevents anomalous users from evading detection by manipulating their features. Our approach improves detection accuracy by up to *70%* F-measure over comparable baselines, and detects a Twitter subgraph of more than *4000* accounts, a majority of which used follower-buying services.

Next, we consider *time series*: for example, how can we detect anomalous events, such as electrical component failures in power grid time series? We develop algorithms for modelling and detecting anomalies in discrete time-series data, such as ratings, where a set of users rate a set of products; and real-valued power grid data, in which we use physics-based circuit models to accurately model and detect anomalies. Then, for mixed categorical, numeric and ordinal data, we propose an online nonparametric anomaly detection approach, that detects anomalies with *61%* higher F-measure than related baselines.

Finally, merging graphs and time series, we consider *graphs with sensors*. Consider a set of sensors arranged in a graph, each collecting data over time: for example, traffic speed sensors, which are arranged on a road network, or voltage sensors on a power grid network. We develop algorithms for detecting anomalous events or large changes happening on a subset of the graph nodes, such as traffic accidents or power line failures. Additionally, we propose algorithms for near-optimally selecting locations for new sensors to be placed on a power grid graph, improving the detection of electrical component failures by *59%* or more F-measure.

# Acknowledgments

First, I want to thank my family for their love and constant encouragement over the years. Thanks for always Skyping with me and keeping me connected to what's going on at home; for giving me advice, encouragement, and prayers; for encouraging my interest in computer science; for pointing me toward important real-world problems; and for supporting me over the years in numerous ways.

I cannot overstate how thankful I am to my advisor, Christos Faloutsos. Throughout my time in graduate school, I really appreciated his kindness and excitement for research, and how he genuinely cares about the personal well-being and growth of his students, constantly guiding us with a wealth of patience, insight and experience. During our meetings, he is always incredibly enthusiastic and energetic, which helped to make research an enjoyable experience for me, even during late-night paper writing sessions before a conference deadline. Christos has always helped greatly in preparing me for an independent academic career, by involving me in research proposal writing, student mentorship, giving me guidance on job search, and numerous very helpful suggestions on giving research presentations, paper writing and so on. Even during the times when he was on sabbatical, he still spent hours meeting me and other students remotely on Skype on the weekends to give us advice on research, discuss new ideas, always with endless patience. He has been my role model, and I hope to emulate his kindness and care for others' well-being, as I continue working with others.

I also want to thank my other thesis committee members: Leman Akoglu, David Choi, and Vipin Kumar. Their guidance, questions and comments throughout the process were invaluable to me in shaping the direction of the thesis, and helped me in shaping my understanding of anomaly detection. I especially thank Leman for being a wonderful collaborator in research - I certainly learned a lot from our research discussions and group meetings, particularly from your insightful and detail-oriented way in which you study and understand papers. I am also very thankful to Susan Holmes, who was my undergraduate research advisor: thank you for all you taught me about research, and for your kind, patient, and enthusiastic advising as I worked on my undergraduate thesis.

I am also very grateful to my friends in the data mining group: graduate school has been a much more fun, productive and enriching experience for me thanks to the chance to discuss ideas and do research with all of you. Alex Beutel, Neil Shah, Vagelis Papalexakis, Danai Koutra, Aditya Prakash: thank you for the constant mentoring and guidance you all gave to myself and the rest of the group; for guiding us around during conferences and making conferences a much more fun experience, for your tips on all aspects of graduate school life, and for being good examples through your work for how to do excellent research. I also greatly thank Kijung Shin, Dhivya Eswaran, Hyun Ah Song, Hemank Lamba, Miguel Araujo, Namyong Park, Minji Yoon, and Shubhranshu Shekhar for making the data mining group fun, and always being amazing collaborators: thanks for being great to hang

out with and bounce research ideas off; I am very grateful for being able to work with all of you. I have also learned a lot about research from my collaborators as well as visitors over the years at CMU: Meng Jiang, Shenghua Liu, Srijan Kumar, Stephan Günnemann, Tsubasa Takahashi, Rohan Kumar, and Yasuko Matsubara; as well as my other collaborators: Asim Smailagic, Pedro Costa, Aastha Nigam, Nitesh Chawla, Boleslaw Szymanski, Mohit Kumar, Disha Makhija and others. I greatly enjoyed working with all of you; thanks for patiently guiding and teaching me to do research, especially when I was a new PhD student.

I also thank my office-mates (Kevin Lin, Taylor Pospisil, Collin A. Politsch, Peter Elliott, Yo Joong Choe, Conor Igoe, Tom Yan), as well as my cohort-mates and friends in statistics and computer science: Jisu Kim, Shashank Singh, Natalie Klein, Yotam Hechtlinger, Jining Qin, Lee Richardson, Sangwon Hyun, Nicolas Kim, Qiong Zhang, Chun Kai Ling, William Herlands, Chun-Liang Li, Ian Yen, Darby Losey, Pratik Patil, Yining Wang and many others, for fun discussions, enjoyable times taking classes together, good discussions over meals, and for watching Youtube videos together on all kinds of topics, which were a distraction from research work that I always appreciated. Much thanks also goes to my friends and collaborators in the electrical and computer engineering department: Larry Pileggi, Amritanshu Pandey, Marko Jereminov, and Shimiao Li: thanks for being such enthusiastic, helpful and patient collaborators. I knew nothing about electrical engineering when we started to work together, but you all always managed to make meetings fun and productive anyway, and I learned a lot from all of you.

Last but certainly not least, I greatly appreciate the wonderful administrative support from Marilyn Walgora, Diane Stidle, Ann Stetser, Adrienne McCorkle, Margie Smykla, Todd Seth, and Tony Mareino: thanks for always being amazingly helpful and even going the extra mile in so many ways, whether in organizing conference trips, accommodating special food requests for group meetings, and many others; I really enjoyed working with all of you. Best wishes to all of you and happy retirement Marilyn.

I could not have done it without all of you: thank you for making this a happy and formative period in my life.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

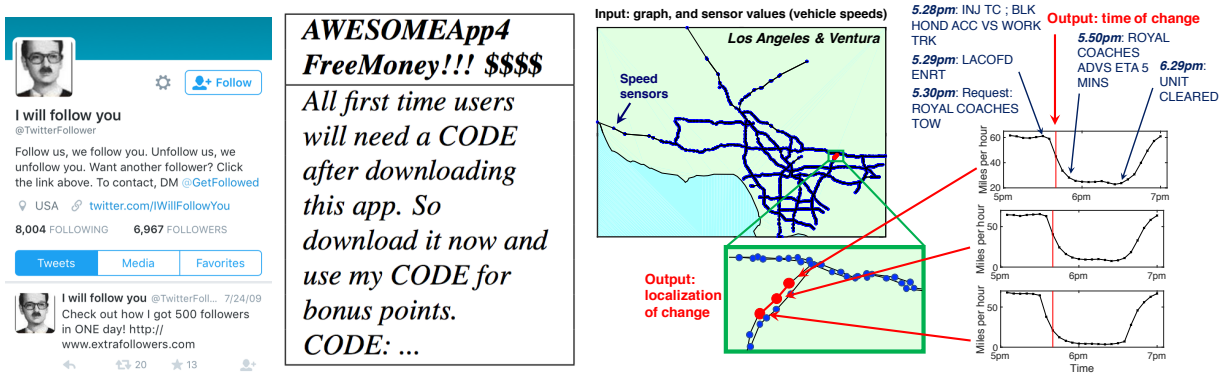
> A capacity for surprise is an essential aspect of our mental life, and surprise itself is the most sensitive indication of how we understand our world and what we expect from it.
>
> Daniel Kahnemann, *Thinking, Fast and Slow*

Our brains contain unconscious but sophisticated machinery for detecting deviations from normality and responding to them quickly. In *Thinking, Fast and Slow*, Kahnemann writes: "Studies of brain responses have shown that violations of normality are detected with astonishing speed and subtlety. In a recent experiment, people heard the sentence 'Earth revolves around the trouble every year.' A distinctive pattern was detected in brain activity, starting within two-tenths of a second of the onset of the odd word. Even more remarkable, the same brain response occurs at the same speed when a male voice says, 'I believe I am pregnant because I feel sick every morning' ... A vast amount of world knowledge must instantly be brought to bear for the incongruity to be recognized [KE11]."

The challenge of quickly noticing deviations from normality is important not just for humans, but also in computer science. In many applications, extremely high-volume data arrives constantly, and the challenge for algorithms is to perform the role that attention does in our brains: of rapidly identifying anomalies in order to surface and respond to them as quickly as possible. Hence, with the increasing availablility of web-scale graphs and high-frequency sensor data, anomaly detection in massive datasets has seen growing focus. Social networks such as Facebook and Twitter contain up to billions of users. Similarly, large-scale sensor data involves numerous types of weather and environmental sensors, networks of voltage sensors spanning the electrical power grid, and networks of traffic speed detectors that span freeway systems in major metropolitan areas.

These datasets are creating an increasing need for scalable algorithms that can automatically analyze this data and flag users or events which are anomalous or of interest. For example, in online commerce and social networks, we are often interested in automatically flagging potentially fraudulent users, spam, network intrusions or other forms of attacks. In large-scale sensor data, we are often interested in unusual events or deviations from normal behavior, such

as traffic accidents, major weather events, or the breakdown of electrical components in power systems data.




(a) Fake followers (b) Fake reviews (c) Traffic accidents

Figure 1.1: Applications of our anomaly detection approaches. (a) **Social networks:** we detect a large group of fake followers in Twitter. (b) **Online reviews:** we detect fake reviews in an online application market. (c) **Traffic:** we detect ground truth traffic accidents (in red) in a dataset containing traffic speed sensors on a road network.

Figure 1.1 shows examples of anomalies in different domains detected by our approaches. In Figure 1.1a, we use connectivity information to detect a large group of fake followers in Twitter. In Figure 1.1b, using temporal information, we detect fake reviews such as this review which advertises 'codes' instead of reviewing the product: we detect these using the temporally bursty nature of groups of fake reviews. In Figure 1.1c, we combine connectivity and temporal information to detect traffic accidents based on traffic speed data: traffic accidents affect a small localized set of nodes on the road traffic graph, and cause a drop in traffic speed over a short time frame. In this figure, nodes represent speed sensors. Red nodes indicate the 3 nodes detected by our algorithm. The 3 plots on the right show the drops in traffic speed on these sensors, which was detected by our method as a change point, and the dark blue text shows ground truth traffic reports, which indicate that a traffic accident occurred.

There is a rich literature in anomaly (or outlier) detection, which defines anomalies as "observations which appear to be inconsistent with the rest of the observations" [BL74]. This thesis studies anomaly detection using graphs and time-series data: we aim to detect anomalies in large graphs such as social networks, as well as time-series sensor datasets, which often involve multiple sensors arranged on a graph and varying over time. Such data are commonplace in modern real-world applications (e.g. arising from online systems), but does not fit neatly into the traditional outlier detection setting, where observations are treated as independent from one another. Hence, the questions we aim to answer are:

- **Q1. Graphs:** How can we identify subgraphs with unusual connectivity in a graph?
- **Q2. Time Series:** How can we use temporal information to detect time periods containing unusual activity?
- **Q3. Graphs with Sensors:** Given time-series sensors arranged on a static graph, how can we detect the time and location of unusual events?

## 1.1 Overview and Contributions

This thesis is organized into three main parts. In Part I, we study how to use connectivity (i.e. graph) information to detect unusual behavior. Part II we focuses on using time-series information. Finally, Part III considers how to combine connectivity and temporal information, in the setting of graphs with sensors.

Table 1.1 provides an overview of this thesis.

Table 1.1: Overview of this thesis.

| | Setting | Goals | Method |
|---|---|---|---|
| **Graphs (Part I)** | Graphs (non-adversarial) | Anomaly Detection | TellTail [PDF] |
| | Graphs (adversarial) | Anomaly Detection (Robust) | Fraudar [PDF] |
| **Time-Series (Part II)** | Time-Series (categorical) | Anomaly Detection | BirdNest [PDF] |
| | Time-Series (real-valued) | Anomaly Detection, Forecasting | StreamCast [PDF] |
| | Time-Series (mixed) | Anomaly Detection | BNB [PDF] |
| | Time-Series (matrix-valued) | Anomaly Detection, Forecasting | SMF [PDF] |
| **Graphs with Sensors (Part III)** | Graphs with Sensors | Anomaly Detection | ChangeDAR [PDF] |
| | Graphs with Sensors | Anomaly Detection, Sensor Placement | GridWatch [PDF] |

### Summary of Technical Contributions

- **Graphs** (Part I): Our TellTail provides a probabilistic score for how measuring how abnormal a subgraph is, while Fraudar considers how to detect abnormal subgraphs in a robust way against adversarial manipulation, improving detection accuracy by up to *70%* F-measure over comparable baselines, and detecting a Twitter subgraph of more than *4000* accounts, a majority of which used follower-buying services.
- **Time Series** (Part II): BirdNest performs anomaly detection in categorical (e.g. ratings) data, while StreamCast considers real-valued (e.g. electrical sensor) data. Then, for mixed categorical, numeric and ordinal data, we propose an online nonparametric anomaly detection approach, BNB, that detects anomalies more accurately than baselines, by *61%* F-measure. Lastly, SMF performs forecasting and anomaly detection in a time series of matrices, capturing seasonality and drift.
- **Graphs with Sensors** (Part III): ChangeDAR detects the time and location of sudden changes in a localized region of the graph. We then develop GridWatch, an anomaly detection approach for power grid data, which also studies how to near-optimally select locations for new sensors to be placed on a power grid graph, improving the detection of component failures by *59%* or more F-measure.

**Reproducibility:** Our code is open-sourced, and our datasets used are available at `www.andrew.cmu.edu/user/bhooi/code`.

**Summary of Impact**

- **Industry Adoption:** BirdNest was deployed at Flipkart for detecting ratings fraud, who found that all of the top 50 reported suspicious users all involved fraud. ChangeDAR and GridWatch contribute to DARPA's $77 million Rapid Attack Detection, Isolation and Characterization Systems (RADICS) program, for power grid security.
- **Taught in Graduate Classes:** our methods are being taught in graduate classes - e.g. University of Michigan (Mining Large-Scale Graph Data) and Virginia Tech (Data Mining Large Networks and Time-Series).
- **Academic Recognition:** Fraudar received KDD 2016 Best Paper Award and CogX 2017 Award for Best Student Paper in AI, and was invited to the TKDD special issue for "Best of KDD 2016". GridWatch received the Runner-up Best Student Paper of ECML-PKDD 2018.
- **Media Coverage:** Fraudar was covered by NSF (Discovery Files podcast), WESA - Pittsburgh NPR, TechXplore, Stanford Scholar, and Crain's.

Next, we summarize the goals and contributions of each of our proposed methods.

## 1.2 Detailed Chapter Summaries

### 1.2.1 Part I: Graphs

Given a graph, how can we measure how unusual a subgraph is and catch unusual subgraphs? In this part, we first propose TellTail, a probabilistic score which evaluates how abnormal a subgraph is, and an efficient algorithm for optimizing this score. We then consider the *adversarial* case, and propose Fraudar, a robust algorithm for detecting dense subgraphs under 'camouflage,' where adversaries add edges (such as Twitter follows) to evade detection.

**Measuring and Detecting Unusual Subgraphs**

*What is an 'unusual' subgraph?*
*How can we measure how unusual a subgraph is, and detect the most unusual subgraphs?*

Our first goal is to define and measure what constitutes an 'unusual' subgraph. Which is more suspicious: a small but very dense subgraph of 10 nodes and 100 edges, or a larger but less dense subgraph of 1000 nodes and 10000 edges? Can we probabilistically measure how unusual each of these subgraphs is, and give our confidence that they indicate anomalous or fraudulent behavior, while exonerating the users if this is within normal variation?

Dense subgraphs often indicate interesting structure, such as network attacks in network traffic graphs, or protein families in protein interaction networks. However, most existing dense subgraph measures, such as average degree, do not take normal variation into account and hence cannot exonerate normal subgraphs, and can also be biased when comparing subgraphs of different sizes. Previous measures [JBC+16] compute a likelihood-based score for subgraphs, but are based on the simple but unrealistic Erdos-Renyi distribution. Hence, in TellTail, we

propose a probabilistic measure for subgraphs based on a more flexible approach that relies on realistic, empirically-observed patterns in real graphs.

**Contributions**

- *Empirical Observations*: We describe empirical patterns about subgraph density distributions in real graphs.
- *Theoretical Framework*: We probalistically evaluate the surprisingness of a subgraph under realistic assumptions.
- *Effectiveness*: Our approach scales linearly, and outperforms baselines in accuracy of detecting injected and ground-truth subgraphs.

**Detecting Unusual Subgraphs in an Adversarial Setting**

*How can we detect unusual subgraphs in the face of adversaries who are trying to evade detection?*

Next, we consider the *adversarial* setting, in which adversaries are trying to evade detection. In particular, consider online commercial settings such as Facebook, Amazon and Twitter, where the data takes the form of a bipartite graph (e.g. likes, reviews, follows, or tweets). A common goal is to detect link fraud, involving groups of fake user accounts which are used to produce a large number of edges (e.g reviews). Existing approaches often try to detect this by searching for dense subgraphs which are sparsely connected to the rest of the graph, e.g. using spectral algorithms.

However, fraudsters performing link fraud can evade many forms of detection using *camouflage*, which involves adding reviews or follows with honest targets so that they look "normal." Furthermore, some fraudsters use *hijacked accounts* from honest users, to make the camouflage more organic and hence harder to detect. Hence, in FRAUDAR, our focus is to spot fraudsters in the presence of camouflage or hijacked accounts.

**Contributions**

- *Camouflage-resistance*: We show theoretically and empirically that our approach performs well in the presence of camouflage.
- *Theoretical guarantee*: Our approach provides an upper bound on the effectiveness of fraudsters.
- *Effectiveness*: FRAUDAR improves detection accuracy by up to *70%* F-measure over the top baseline. In real-world experiments with a Twitter graph of 1.47 billion edges, FRAUDAR successfully detected a Twitter subgraph of more than *4000* accounts, a majority of which used follower-buying services.

### 1.2.2 Part II: Time Series

Time series data arising from online systems often involves discrete events (such as likes, follows, reviews, page views etc.) accompanied by timestamps, while the more traditional time series sensor setting involves real-valued time series data. For the former, we develop an anomaly

detection algorithm for ratings data (i.e. ratings submitted by users for products) in order to catch fraudsters who manipulate ratings [HSB+16a]. For the latter, we develop a forecasting and anomaly detection algorithm for power systems data that exploits domain knowledge via a physics-based model of power systems data [HSP+18]. Next, for mixed data (i.e. combined categorical, numeric or ordinal data), we develop an online nonparametric change detection approach [HF19]. Finally, we propose an online approach for forecasting matrix-valued time series (where we observe a matrix at each time point), and further show that we can efficiently perform anomaly detection in a computationally efficient manner, without forecasting every entry in the matrix [HSLF19].

### Anomaly Detection in Categorical Time Series

*How can we detect unusual users in categorical data involving a series of events (e.g. reviews)?*

We first consider categorical temporal data arising from online settings, which often involves a series of discrete events, such as reviews. Review fraud is a pervasive problem in online commerce, in which fraudulent sellers write or purchase fake reviews to manipulate perception of their products and services. Fake reviews are often detected based on several signs, including 1) they occur in short bursts of time; 2) fraudulent user accounts have skewed rating distributions. We use a Bayesian inference approach, which allows a principled approach for taking into account the number of reviews each user has, as well as combining temporal and rating information.

Hence, in this chapter, we first formulate our model, 'Bayesian Inference for Rating Data' (BIRD) model, a flexible Bayesian model of user rating behavior. Based on our model we formulate a likelihood-based suspiciousness metric, 'Normalized Expected Surprise Total' (NEST).

**Contributions**

- *Theoretically sound user behavior model*: we define a Bayesian model for the data based on a mixture model which captures different types of user behavior.
- *Suspiciousness metric*: we define a likelihood-based metric which measures how muc a user deviates form normal behavior.
- *Effectiveness*: our approach runs in linear time and successfully spots fraud in real-world graphs, which precision of over $84\%$ on the top $250$ Flipkart users flagged by our algorithm.

### Anomaly Detection in Real-Valued Time Series

*How can we detect unusual events in real-valued time series data? (e.g. sensor data)*

Next, we consider the more traditional real-valued, multivariate time series sensor setting: e.g. power grid data, which consists of voltage, current, and environmental temperature data over time. Our goal is to forecast the power consumption of a location for the next few days, which can then be used for anomaly detection, by flagging anomalies when the forecast differs

from reality by more than a fixed threshold. An additional challenge is to take into account 'what-if-scenarios:' e.g. what if the temperature increases by $10°C$, the number of appliances in the grid increase by $20\%$, and voltage levels increase by $5\%$? Such scenarios are crucial for future planning, to ensure that the grid remains reliable even under extreme conditions.

**Contributions**

- *Domain knowledge infusion*: we propose a novel *Temporal BIG* model that extends the physics-based BIG model, allowing it to capture changes over time, trends, and seasonality, and temperature effects.
- *Effectiveness*: StreamCast forecasts multiple steps ahead and outperforms baselines in accuracy by $27\%$ or more.
- *Scalability*: Our algorithm is online, requiring constant update time per new data point, and bounded memory.

### Anomaly Detection in Mixed Time Series

*How can we detect unusual events in mixed categorical, numeric and ordinal time-series data?*

Next, we consider multivariate time series containing mixed data types. A key challenge in this setting is that we cannot make strong assumptions about the type of model that the data follows, and how it changes after the event occurs: i.e. we want our approach to be *nonparametric*. In this case, how do we detect changes in the behavior of the time series: for example, the onset of illnesses or complications in patients?

We propose BNB (Branch and Border), an online, nonparametric change detection method that detects multiple changes in multivariate data. Unlike most existing methods which fit particular distributions (e.g. Gaussian, Poisson), BNB approaches change detection by treating changes as time points where relatively simple random partitions of the data space can cleanly separate the points before and after the change.

**Contributions**

- *Generality*: as BNB is nonparametric, it works well for data sources with any distribution, including numerical, categorical, and ordinal data.
- *Effectiveness*: BNB achieves $70\%$ or more increased F-measure over comparable baselines in experiments averaged over $11$ datasets.
- *Scalability*: BNB scales linearly in the number of time ticks and dimensions, and is online, thus using bounded memory and bounded time per iteration, regardless of the stream length.

### Anomaly Detection in Matrix-Valued Time Series

*Given a stream of matrices, how can we fit a model that captures seasonality and drift over time, and use it to forecast future data, and detect anomalous events?*

Finally, we consider matrix-valued time series. An example is taxi ride data: at each time point, we have a matrix whose rows represent the starting location of the ride, and columns represent the ending location, where each taxi ride is an element in this matrix. Our goal is to learn a model which captures seasonal patterns (e.g. rides toward office areas are more frequent in the morning) and drift (e.g. population growth). We propose a matrix factorization-like model that provides interpretable components, and forecasts future data more accurately than existing approaches. In addition, we show that in the large, sparse setting, we can perform anomaly detection efficiently, without needing to forecast every observation in the matrix.

**Contributions**

- *Model:* we propose a novel matrix factorization model incorporating seasonal patterns and drift, an online fitting algorithm, and an efficient anomaly detection approach.
- *Effectiveness:* in experiments, SMF has lower forecasting error than baselines by $13\%$ to $60\%$, and provides interpretable results in case studies on real data.
- *Scalability:* SMF is online, and scales linearly. In experiments, it was $12$ to $103$ times faster than seasonal baselines.

### 1.2.3 Part III: Graphs with Sensors

We now consider data from a set of sensors placed on a graph, where each sensor provides time series data. An example is electrical sensors placed on nodes of the power grid, where we would like to automatically detect anomalous events, such as due to the failure of electrical components. Another example is traffic sensors which measure the speed of vehicles passing over them, which we want to use to detect traffic accidents.

**Anomaly Detection in Graphs with Sensors**

*Given time-varying data from sensors arranged over a graph, how can we detect the time and location of sudden changes occurring on the graph?*

For this setting of sensors placed on a graph, our first approach considers the *change detection* problem, where we want to detect significant and sudden changes that occurred over a connected subgraph of sensors. For example, after a traffic accident, we would expect traffic speeds to drop sharply at a subgraph of roads around the accident. Similarly. sharp changes in power grid sensor values occur during a power grid failure. Hence, our ChangeDAR (Change Detection And Resolution) algorithm detects localized changes on the graph in an online manner, and reports when and where the change occurred in the graph.

**Contributions**

- *Algorithm*: We propose novel information theoretic optimization objectives for scoring and detecting localized changes, and propose two algorithms, ChangeDAR-S and ChangeDAR-D respectively, to optimize them.
- *Theoretical Guarantees*: We show that both methods provide constant-factor approximation guarantees.

- *Effectiveness*: In experiments, ChangeDAR detects traffic accidents and power line failures with $75\%$ higher F-measure than comparable baselines.
- *Scalability*: our methods are online and near-linear in the graph size and the number of time ticks.

### Sensor Placement in Graphs with Sensors

*How can we select positions for sensors over a power grid graph, in order to detect anomalous events?*

In the power grid setting, maintaining the reliability of the electrical grid is a major challenge, and an important part of achieving this is to place sensors in the grid, and use them to detect anomalies. Hence, focusing on the power grid setting, we now consider the follow-up question of selecting where to place sensors over a graph, in order to have the highest probability of detecting an anomaly using these sensors. In addition, using the chosen sensors, we then propose an online algorithm for detecting anomalies, in order to provide real-time feedback on the most anomalous parts of the graph.

**Contributions**

- *Online anomaly detection*: We propose GridWatch, a novel, online anomaly detection algorithm with higher accuracy than existing approaches.
- *Effectiveness*: Our sensor placement algorithm is provably near-optimal, and both our algorithms outperform existing approaches in accuracy by $59\%$ or more (F-measure) in experiments.
- *Scalability*: Our algorithms scale linearly, and our detection algorithm is online, requiring bounded space and constant time per update.

# Chapter 2

# Background

In this chapter, we introduce the main definitions in graph theory and time series analysis that are useful in understanding this thesis.

## 2.1 Graphs

We start with the definition of a graph, along with the common types of graphs (bipartite, directed, weighted).

**Graph:** a *graph* (or *network*) is a data structure used to model interactions. Graphs consist of a set of *nodes*, and a set of *edges* between pairs of nodes. For example, the nodes could represent people, and the edges could represent friendships between them. Mathematically, a graph $\mathcal{G}$ is an ordered pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges.

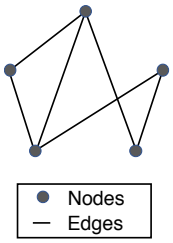



Figure 2.1: A graph.

**Bipartite Graph:** a graph whose nodes can be divided into two disjoint sets $\mathcal{U}$ and $\mathcal{W}$ such that all edges connect a node in $\mathcal{U}$ and a node in $\mathcal{W}$. For example, given a product review website (such as Amazon), $\mathcal{U}$ can be the set of users, and $\mathcal{W}$ the set of products, where an edge between a user and a product represents a review by that user for that product.

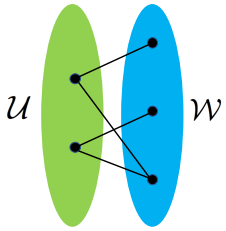



Figure 2.2: A bipartite graph.

**Weighted Graph:** a graph whose edges have a numeric weight associated with them. For example, in a product review graph, edge weights may indicate the rating that the user gave the product. In Figure 2.3, thickness is used to indicate edge weight.

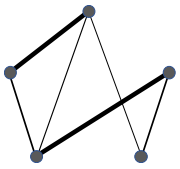


Figure 2.3: A weighted graph.

**Subgraph:** a *subgraph* of a graph $\mathcal{G}$ is a graph formed by a subset of the nodes and edges of $\mathcal{G}$.

**Induced Subgraph:** given a graph $\mathcal{G}$, the *induced subgraph* of a subset $\mathcal{V}'$ of nodes is the subgraph with node set $\mathcal{V}'$, and edge set equal to all edges connecting pairs of nodes in $\mathcal{V}'$.

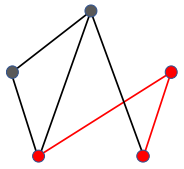


Figure 2.4: A subgraph (in red) induced by the 3 red nodes.

**Neighbors:** two nodes are called *neighbors* if there is an edge connecting them.

**Neighborhood:** the *neighborhood* of a node is the induced subgraph consisting of the node and all its neighbors.

**Degree:** the *degree* of a node is the number of neighbors of the node.

## 2.2 Time Series

**Time Series:** a *time series* $X_1, X_2, \cdots$ consists of measurements of a variable collected over time. Usually, the measurements are made at regular time intervals. A time series is *univariate* if $X_i \in \mathbb{R}$, while *multivariate* time series allow $X_i \in \mathbb{R}^d$.

**Online Algorithm:** an *online* (or *streaming*) algorithm is one that processes its input incrementally, i.e. in the order that the algorithm receives its input. As each new data point is received, the algorithm should compute a partial solution based on the input it has been given. This contrasts with *offline algorithms*, which are only required to output an answer after receiving the full input data.

## 2.3 Graphs with Sensors

**Graphs with Sensors:** In the settings we consider, a *graph with sensors* consists of a static graph, with time-series sensors placed on a subset of its nodes or edges. Some sensors can be missing (thus, this naturally covers the case where sensors may be placed only on nodes, or only on edges).

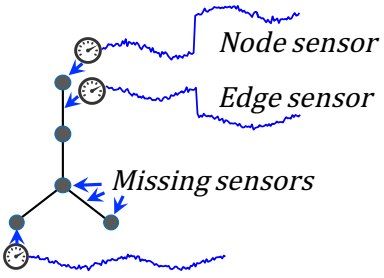



Figure 2.5: A graph with sensors.

# Part I

# Graphs

# Overview: Graphs

*Given a graph, how can we measure how unusual a subgraph is?*
*How can we catch unusual subgraphs?*

Static graphs, which contain only connectivity information, are commonly used in practice when temporal information is not present. In this part, we consider both how to measure how unusual a subgraph is, as well as how to detect subgraphs with unusual connectivity, such as network intrusion attacks, and link spam in social networks. Hence, TellTail focuses on providing a **probabilistic score** for how abnormal a subgraph is: unlike previous work, we evaluate how unusual a subgraph is under more realistic assumptions than existing Erdos-Renyi approaches. Fraudar provides an algorithm for detecting fraud in review graphs and social networks in a way that is robust against certain types of **adversarial manipulation**, aiming to catch fraudsters who try to conceal their presence by adding edges toward honest products.

# Chapter 3

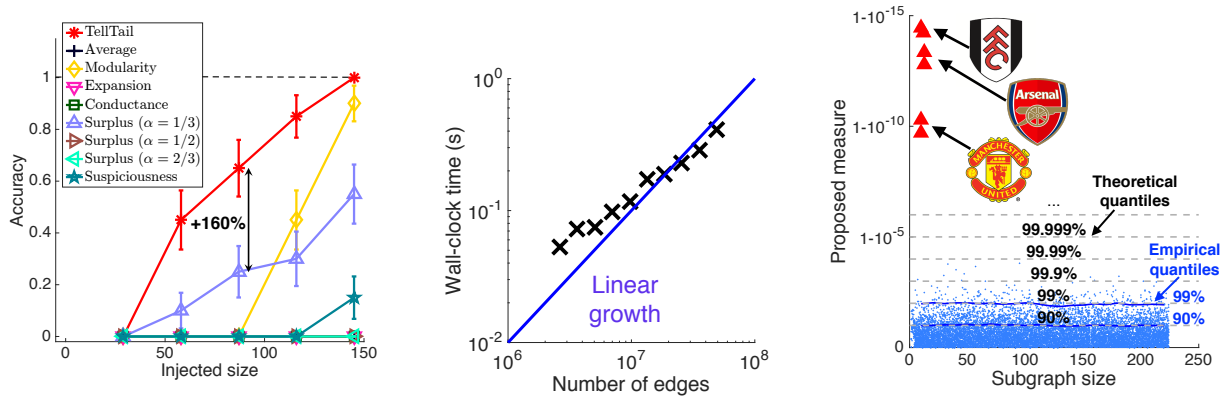
# TellTail: Scoring Dense Subgraphs

In this chapter, we propose probabilistic anomaly detection methods for graphs. Given a phone call graph, or a graph of following relationships on Twitter, how can we design a principled measure for identifying surprisingly dense subgraphs? If 50 users each reviewed almost all the same 500 products several times each, can we measure our confidence that this indicates anomalous or fraudulent behavior, while exonerating the users if this is within normal variation? Our TellTail approach provides a probabilistic measure with theoretical consistency guarantees, and provides near-linear time detection, while avoiding Erdos-Renyi assumptions common in the literature.

## 3.1 Introduction

Given an undirected, possibly weighted graph, how can we measure how surprising or anomalous a subgraph is? How can we do this in a way that exonerates subgraphs that are within the range of normal variation, but catches only subgraphs which are truly surprising? Dense subgraph detection is useful for detecting social network communities, protein families [SHK+10], follower-boosting on Twitter, and rating manipulation [HSB+16b]. In these situations, it is useful to measure how surprising a dense subgraph is, to focus the user's attention on surprising or anomalous subgraphs.

Many measures exist for identifying dense subgraphs, such as average degree, modularity, etc. However, knowing that a subgraph has, for example, an average degree of $10$, does not tell us how surprising it is, since we do not know if this is plausible under normal variation. To quantify how surprising a subgraph is, we need a *probabilistic* measure, that tells us, e.g., that the probability of a random subgraph of the same size being as dense as this one is $10^{-5}$. Such a probabilistic measure is useful for decision making: if a subgraph is very unlikely under normal variation, we can more confidently take action such as investigating these users.




(a) Accuracy (UCForum dataset) (b) Quasi-linear runtime (c) Accurate confidence estimates

Figure 3.1: (a) **Accuracy:** TellTail outperforms competitors in accuracy in detecting injected blocks. (b) **Scalability:** Our search algorithm scales quasi-linearly. (c) **Accurate confidence estimates:** our measure clearly separates ground truth subgraphs (red triangles) from random subgraphs (blue dots) while providing confidence estimates.

The key challenge in this process is to accurately model subgraph densities under normal behavior. In other words, what is a good null model? The simplest approach would be to assume an Erdos-Renyi model, as in [JBC+16]. However, the Erdos-Renyi model does not accurately model density patterns of real graphs: it ignores the clustering structure of graphs, as well as dense subgraphs caused by high degree nodes, such as 'hyperbolic communities' [AGMF14] observed in real graphs.

Instead, our approach uses a novel application of extreme value theory to the dense subgraph problem, with the goal of probabilistically measuring how 'extreme' a dense subgraph is. Extreme value theory is an elegant statistical approach for modelling the distribution of extreme (or rare) events. It was initially proposed to model the distribution of flood heights, with the goal of finding the minimal dike height which would be tall enough to withstand floods with high probability. Since then, it has been applied to many types of extreme events, such as earthquakes, financial crashes, and network attacks. Extreme value theory allows us to accurately estimate the extreme **tail** of a distribution without making strong assumptions about the distribution itself.

We use extreme value theory for modelling the top most dense subgraphs for two reasons: firstly, it provides more accurate modelling. Like flood heights, we are mainly interested in the densest few subgraphs, which is an inherently extremal process. Indeed, modelling the overall distribution itself in full can mislead us about the tail of the distribution, particularly in realistic settings where the extreme values are outliers whose distribution deviates from that of the normal values. We show in Section 3.5.3 that our approach models real-world dense subgraph patterns more accurately than other models. Secondly, the full distribution of subgraph densities in real graphs is hard to theoretically characterize explicitly, which is why it has only been

done for the (unrealistic) Erdos-Renyi model so far. Thus, extreme value theory allows us to theoretically characterize dense subgraph patterns without restrictive assumptions.

To improve its practicality, our method further makes use of two novel empirical findings about the distribution of subgraph densities in real graphs. We use these empirical findings in our measure which assesses how surprising a subgraph is, then propose a fast pruning-based algorithm for detecting dense subgraphs using this measure, in quasi-linear time. Figure 3.1a shows that our measure outperforms baselines in its accuracy of identifying injected subgraphs. Figure 3.1b shows that our search algorithm is fast and scales quasi-linearly: it took $0.41$s per subgraph to find, on a graph with $49$ million edges, on a laptop computer. Figure 3.1c shows that on a Twitter dataset of football clubs, our measure separates ground truth football clubs from random subgraphs, while providing confidence estimates. The 'empirical quantiles' plot the $90\%$ and $99\%$ quantiles of the measure on random subgraphs, matching the theoretical quantiles and verifying that our confidence estimates are accurate.

Our contributions are:

- **Theoretical underpinnings:** we propose a probabilistic framework (Definition 3.5) for finding dense subgraphs based on extreme value theory. Theorem 3.2 provides a guarantee of consistency.
- **Discoveries:** We make 2 novel empirical observations on dense subgraph patterns in real graphs, which we use to speed up our measure.
- **Effectiveness:** Our approach outperforms baselines in finding injected (Figure 3.7) and ground truth dense subgraphs (Table 3.4 and 3.5).
- **Algorithm:** Our search algorithm scales quasi-linearly, and we further speed it up using a safe pruning step (based on Theorem 3.1).

**Reproducibility**: Our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/telltail/`.

## 3.2 Related Work

**Measures based on internal connectivity:** average degree [Gol84] is a common measure that can be optimized exactly [Gol84] or via $1/2$-approximation factor greedy algorithms [Cha00]. Variants allow for size restrictions [AC09] or local subgraphs [And10]. Other measures include edge density, which underlies quasi-cliques [ARS02], edge surplus [TBG+13], triangle and k-clique density [Tso15], discounted average degree [YH18], and minimum internal degree, which defines $k$-cores [SERF16]. Related measures underlie k-plexes [SF78] and k-trusses [Coh08]. [ANMJZ12] evaluates density by subtracting the expected density of similar clusters.

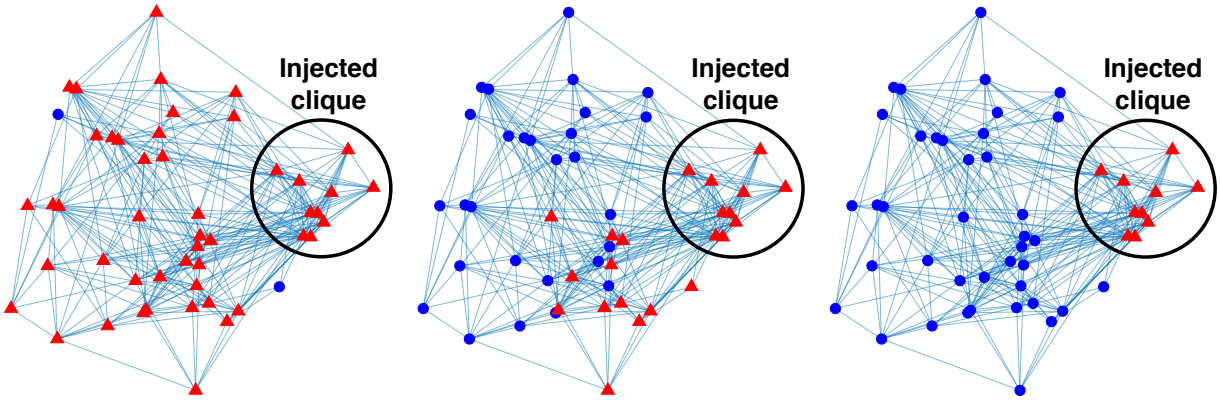
**Measures based on internal and external connectivity:** external connectivity refers to the edges between the subgraph and the rest of the graph. These measures find subgraphs which are dense internally but sparsely connected to the rest of the graph. These include modularity [New06], Maximum, Average, and Flake-ODF [FLG00], local density [QLCZ15], and the family of cut-based measures, such as expansion [RCC+04] and conductance [SM00]. [KVV04] proposes a bicriteria related to conductance for analyzing spectral clustering.

**Model-based measures:** many generative models for graphs exist, such as scale-free graphs [LADW05], Kronecker graphs [LCK+10], and so on. However, computing surprisingness under these models is often infeasible because the models do not allow for tractable analysis. For the Erdos-Renyi model, [BE76] analyzes cliques in random graphs.

Closely related to our approach is Jiang (2016) [JBC+16] which defines the *suspiciousness* subgraph measure: for a subgraph $S$ with $e(S)$ edges, the suspiciousness is $-\log P(Y = e(S))$, where $Y$ has a Poisson distribution with mean equal to the expected mass of $S$ under an Erdos-Renyi model. Effectively, the score of a subgraph is (approximately) the negative log probability of how likely we are to achieve the same mass assuming edges are drawn randomly from an Erdos-Renyi model. In comparison, our approach uses a more empirical approach of studying subgraph distributions of real graphs, showing that they are well-modeled by a Generalized Pareto (GP) distribution.

Also closely related is van Leeuwen (2016) [vLDBSM16], which proposes a subgraph interestingness measure. Their general framework evaluates how subjectively interesting a subgraph is relative to a user's prior beliefs. Given such priors, subjective interestingness of a pattern is the ratio of *information content* (negative log probability of the pattern being present under the user's belief state) over the *description length* (length of pattern's description). In comparison, our approach also evaluates subgraphs in the context of a null distribution, but takes a more empirical approach, with our choice of null distribution motivated by empirical patterns in real graphs, rather than the user's prior beliefs.

## 3.3 Background: Generalized Pareto




(a) Average Degree measure (b) Modularity measure (c) TellTail measure

Figure 3.2: **TellTail avoids size-bias:** in an Erdos-Renyi graph with $n = 50$ and $p = 0.2$, with an injected clique of size 10 (circled). The subgraph detected under each measure is labeled in red triangles.

The Generalized Pareto (GP) distribution is a 3 parameter distribution used within extreme value theory [CBTD01]. Its key properties are its flexibility in smoothly interpolating between light- and heavy-tailed regimes, and its 'universality' property, Property 3.1.

The CDF of the GP distribution [CBTD01] is:

$$GPD_{\mu,\sigma,\xi}(x) = \begin{cases} 1 - (1 + \frac{\xi(x-\mu)}{\sigma})^{-1/\xi} & \text{if } \xi \neq 0 \\ 1 - \exp(-\frac{x-\mu}{\sigma}) & \text{if } \xi = 0 \end{cases} \tag{3.1}$$

for $\sigma > 0$. Its support (i.e. the set of $x$ with nonzero density) is $x \geq \mu$ when $\xi \geq 0$, and $\mu \leq x \leq \mu - \sigma/\xi$ when $\xi < 0$. Its three parameters are its **location** parameter $\mu$, its **scale** parameter $\sigma$, and its **shape** parameter $\xi$. The GP distribution generalizes several well-known distributions:

- For $\xi > 0$ it is a Pareto distribution with shape $\alpha = 1/\xi$;
- For $\xi = 0$ and $\mu = 0$ it is an exponential distribution with mean $\sigma$.

Note that Pareto distributions exhibit heavy-tailed decay (i.e. power law tails) while exponential distributions exhibit light-tailed decay (i.e. exponential tails), so the GP distribution interpolates smoothly between the two regimes.

### Properties

Given any random variable $X$, let $t \in \mathbb{R}$ and define a new distribution $F_t$ representing the upper tail of $X$ past threshold $t$; i.e. $F_t$ is the distribution of $X - t$ conditioned on $X > t$.

**Definition 3.1**

The **conditional excess at threshold** $t$ has distribution (i.e. CDF) $F_t(y) = P(X - t \leq y | X > t)$.

The universality property, or Pickands- Balkema-de Haan theorem [BDH74], states that the GP distribution can approximate the tails of an arbitrary distribution:

Let $F$ be any distribution function from a broad class of distributions. (This class includes almost all commonly used distributions; see [EKM99].)

**Property 3.1: Universality**

There exists $\xi, \sigma(t)$ that approximates the tail of $F$ arbitrarily closely: i.e.:

$$\lim_{t \to t_{\max}} \sup_x |F_t(x) - \text{GPD}_{0,\sigma(t),\xi}(x)| = 0, \tag{3.2}$$

where $t_{\max}$ is the right endpoint of $F$ ($t_{\max}$ can be $\infty$).

Intuitively, GP distributions can closely model upper tails arising from many distributions. This justifies using GP distributions to model the upper tail of subgraph mass distributions, as we will do.

## 3.4 Problem Definition

Table 8.2 summarizes the symbols used in this chapter.

| Symbol | Interpretation |
|---|---|
| $G(V, E)$ | Graph, with its vertex and edge set |
| $n$ | Number of nodes |
| $m$ | Number of edges |
| $S$ | Subset of nodes |
| $k$ | Size of subset (i.e. $\|S\|$) |
| $A$ | Adjacency matrix |
| $d$ | $n \times 1$ vector of node degrees |
| $B$ | Modularity matrix: $B = A - d \cdot d^T/(2m)$ |
| $f(S)$ | Desired output: the surprisingness of $S$ |
| $e(S)$ | Number of edges in induced subgraph of $S$ (also called its **mass**) |
| $\tilde{e}(S)$ | Adjusted mass of induced subgraph of $S$ i.e. its mass minus its expected mass |
| $d(S)$ | Sum of (weighted) degrees of nodes in $S$ |
| $\mu, \sigma, \xi$ | Location, scale, shape parameters of GP distribution |
| $\epsilon$ | GP distribution tail cutoff probability |
| $r$ | No. of repetitions in TellTail-Search |
| $t$ | No. of neighborhoods in TellTail-Search+ |

Table 3.1: Symbols and Definitions

Given a graph $G = (V, E)$, our goal is to estimate how surprising a subgraph induced by $S \subseteq V$ is by defining a scoring function $f$ from subsets of $V$ to $\mathbb{R}$. Our scoring function should quantify the **probability** of observing a subgraph at least as dense as this one:

**Problem 3.1: Scoring Function**

**Given:** graph $G = (V, E)$, subgraph induced by $S \subseteq V$
**Output:** $f(S)$, an estimate of how surprising the mass of $S$ is compared to the distribution of subgraphs of the same size.

## 3.5 Proposed Approach

### 3.5.1 Introductory Example

Consider an Erdos-Renyi graph $G$ of size $n = 50$ with edge probability $p = 0.2$. We select 10 random nodes and add all the edges between them to $G$, creating an injected clique, as shown in Figure 3.2, with the injected clique circled. To detect the clique, we optimize each one of three measures (average degree, modularity, and TellTail) using a standard greedy local search approach [JBC+16], plugging in the measure in question. Figure 3.2a shows that the highest average degree subgraph (red triangles) consists of almost all the nodes; Figure 3.2b shows that the highest modularity subgraph contains around half the nodes; and Figure 3.2c shows that under TellTail, the densest subgraph coincides with the injected clique.

Why does this happen? Consider average degree, $2e(S)/|S|$. The average degree of the entire graph (i.e. setting $S = V$) is $2|E|/|V|$, while the average degree of any subgraph of $k$ nodes cannot exceed $k - 1$, even for a clique. Hence, average degree is biased in favor of larger subsets, and in Figure 3.2a, the highest average degree subgraph can be almost the entire graph. A similar problem holds for modularity, which tends to select subgraphs of size close to $n/2$, as also observed by Leskovec et al. [LLM10]. This problem is not specific to average degree and modularity: the key issue is that many measures are **size-biased**, meaning that the values they produce on subgraphs of different sizes are not comparable across sizes in any principled way. Hence a highly surprising subgraph may be given a lower measure value than a less surprising subgraph, due to difference in sizes.

In contrast, our proposed measure TellTail handles this problem by **controlling for size**: it evaluates the surprisingness of a subgraph by comparing it to the population of subgraphs of the same size. Specifically, it estimates the probability that a random subgraph of the same size would have equal or higher density than the given subgraph. Since the measure values are in 'units' of probability, they can be compared across sizes in a principled way.

### 3.5.2 Subgraph Mass Profiles

We start by introducing the **subgraph mass profile**, which we use to study empirical distributions of subgraph masses in real graphs.

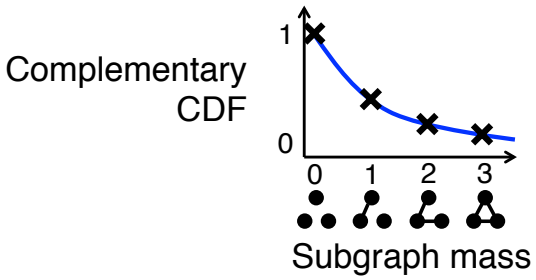
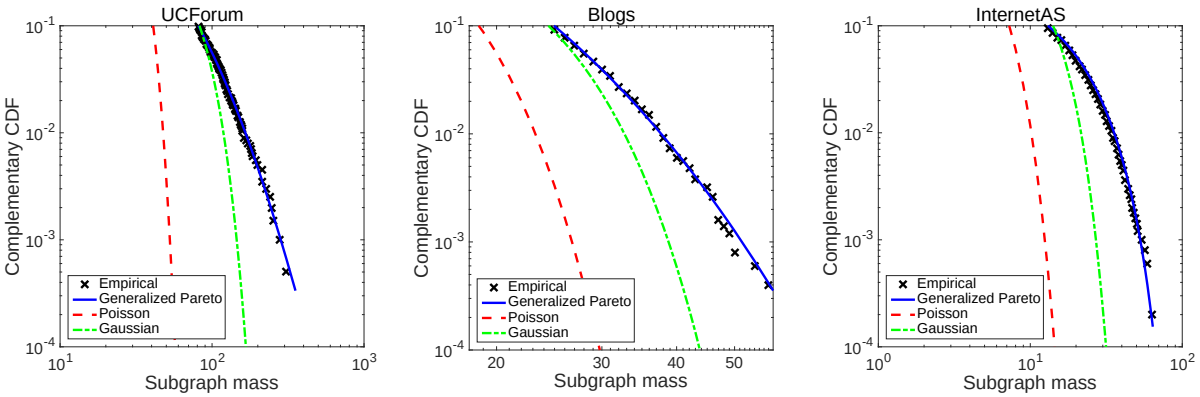



Figure 3.3: A subgraph mass profile, for size $k = 3$. The curve is 1 – CDF of the mass of a random subgraph of $k$ nodes.




(a) UCForum dataset ($n = 899$) (b) Blogs dataset ($n = 1.2$K) (c) InternetAS dataset ($n = 34$K)

Figure 3.4: **The GP distribution closely fits mass distributions of real graphs:** Black crosses indicate the empirical distribution of subgraph masses for subgraphs of size $k = \lfloor\sqrt{n}\rfloor$, in the form of its complementary CDF. The colored curves are the best fit GP, Poisson, and Gaussian to the empirical distribution.

Figure 3.3 illustrates a subgraph mass profile. For each subgraph mass, the curve plots the probability that a randomly chosen subset of size $k$ will have at least that mass. In other words, it is the CCDF (i.e. $1-$ CDF) of the mass of a random subgraph of size $k$.

Next, we will estimate these profiles in real graphs, and study the resulting empirical patterns.

### 3.5.3 Empirical Observations

Figure 3.4 shows the empirical distribution of subgraph masses in three real graphs. The crosses show the empirical CCDF of the mass of 5000 random subgraphs of size $k = \lfloor\sqrt{n}\rfloor$. The 3 colored lines are maximum likelihood fits of 3 distributions to these masses.

The wide gap between the Poisson curve and the crosses indicates that the Poisson distribution greatly underestimates how many dense subgraphs we should observe. This makes sense, as [JBC+16] shows that approximately Poisson mass distributions occur under an Erdos-Renyi graph model. However, the Erdos-Renyi model ignores the clustering effects present in real graphs. Hence, in real graphs, much denser subgraphs exist than in Erdos-Renyi models.

Figure 3.4 shows that Gaussian distributions also decay too quickly, while the Generalized Pareto (GP) distribution fits the empirical distribution very closely. For space reasons, plots for our 5 other datasets are in our supplementary document [sup]. To summarize:

**Observation 3.1**

The upper tail of the subgraph mass distributions of real graphs closely follows a GP distribution.

## 3.6 Proposed Measures

Hence, we now propose measures which approximate a subgraph mass distribution using a GP distribution, and use it to estimate the surprisingness of each subgraph.

### 3.6.1 TAIL Measure

TAIL, our simplest approach, estimates this GP distribution via sampling. Given a subgraph $S$ to evaluate of size $|S| = k$, first sample $N$ uniformly random subsets of $V$ of size $k$, and compute the mass of each. Fit a GP distribution using maximum likelihood [Gri93] to the largest $\lfloor \epsilon N \rfloor$ of these subgraph masses. Then, the surprisingness of $S$ is the CDF of the fitted GP distribution, evaluated at $e(S)$:

**Definition 3.2: TAIL Measure**

$$f(S) = \text{GPD}_{\hat{\mu},\hat{\sigma},\hat{\xi}}(e(S))$$

where $\hat{\mu}, \hat{\sigma}, \hat{\xi}$ are the maximum likelihood GP fit [Gri93].

### 3.6.2 Adjusting for Degree: the TAILDC Measure

How do we identify subgraphs that are not just dense, but also sparsely connected to the rest of the graph? For example, in a user-product review graph, subgraphs which are dense internally and sparse externally could suggest the presence of lockstep behavior.

Instead of using the mass $e(S)$ of subset $S$, we compute its **adjusted mass**, denoted by $\tilde{e}(S)$, which is its mass minus the expected value of its mass assuming edges are connected randomly. Similar to the computation of the modularity measure [New06], the expected value of $e(S)$ is equal to $d(S)^2/(4m)$, where $d(S)$ is the sum of degrees of the nodes in $S$.

**Definition 3.3: Adjusted mass**

The adjusted mass of $S$ is its mass minus the expectation of its mass, and is equal to:

$$\tilde{e}(S) = e(S) - d(S)^2/(4m)$$

TAILDC differs from TAIL only in that it uses adjusted mass instead of mass:

**Definition 3.4: TailDC Measure**

$$f(S) = \text{GPD}_{\hat{\mu},\hat{\sigma},\hat{\xi}}(\tilde{e}(S))$$

where $\hat{\mu}, \hat{\sigma}, \hat{\xi}$ are maximum likelihood GP parameters for the distribution of adjusted masses.

### 3.6.3 Dense Subgraph Power Laws

Tail and TailDC require a time-consuming sampling step. How do we speed them up? In this subsection, we first observe near-power law empirical patterns. We will use these patterns to greatly speed up TailDC by eliminating the random sampling step.

Let $\mu(k)$ be the GP location parameter as a function of subgraph size $k$ (and similarly for $\sigma(k)$). For GP distributions fitted to adjusted mass $\tilde{e}(S)$, we observe power law patterns for $\mu(k)$ and $\sigma(k)$:

$$\mu(k) = \mu_0 k^{\alpha} \tag{3.3}$$

$$\sigma(k) = \sigma_0 k^{\beta}. \tag{3.4}$$

where $\mu_0, \sigma_0, \alpha$ and $\beta$ are constants to be determined.

Plotting $\mu(k)$ and $\sigma(k)$ against $k$ on a log-log plot, this takes the form of a straight line. Figure 3.5 shows such plots for the `InternetAS` graph (dataset information is in Table 10.3). $\mu(k)$ and $\sigma(k)$ closely follow the near-power law patterns in Eq. (3.3) and (3.4), with $R^2$ of $1.00$ and $0.99$ respectively (where $R^2 = 1$ indicates a perfect linear fit). Table 3.2 shows that the same pattern holds across many datasets, with $R^2$ values very near $1$, and additionally, $\alpha$ and $\beta$ are close to $0.9$ and $0.8$ respectively, while $\xi$ is close to $-0.1$.

**Observation 3.2: Dense Subgraph Power Laws**

In real graphs, the GP parameters of adjusted subgraph mass distributions closely follow $\mu(k) = \mu_0 k^{0.9}$ and $\sigma(k) = \sigma_0 k^{0.8}$.

### 3.6.4 TellTail: Fast Scoring using Power Laws

Using Observation 3.2, we now propose our main TellTail measure, which avoids the sampling steps in Tail and TailDC, providing large speedup. TellTail is a simple, closed-form measure that runs in $O(m)$ time. Assume that the measure is used to compare the surprisingness of different subsets (e.g. in an algorithm for finding the most surprising $S$), so we can ignore strictly increasing transformations of the measure (e.g. multiplication by a constant), as they do not change the relative order of different subsets.

As in TailDC, the surprisingness of $S$ is the CDF of a GP distribution, evaluated at the adjusted mass of $S$, as given in (3.1). As observed in Section 3.6.3, $\xi$ remains fairly constant empiri-

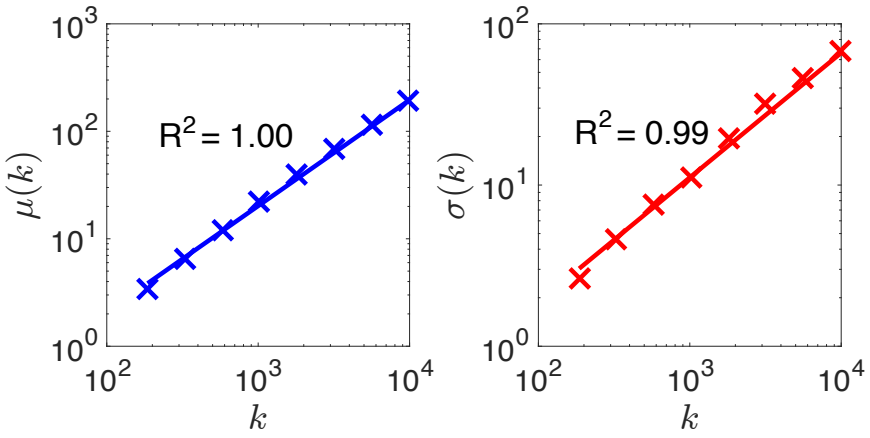



Figure 3.5: **Power law patterns in GP parameters**: in the `InternetAS` graph.

| Dataset | $\mu$ slope ($\alpha$) | $R^2$ | $\sigma$ slope ($\beta$) | $R^2$ | $\xi$ |
|---|---|---|---|---|---|
| `UCForum` | 0.87 | 0.99 | 0.63 | 0.98 | -0.02 |
| `Email` | 0.82 | 0.99 | 0.74 | 0.98 | -0.12 |
| `Blogs` | 0.87 | 0.99 | 0.76 | 0.99 | -0.05 |
| `Petster` | 0.85 | 0.99 | 0.78 | 0.99 | -0.11 |
| `PGP` | 0.86 | 0.99 | 0.81 | 0.99 | -0.11 |
| `AstroPh` | 0.88 | 0.99 | 0.85 | 0.99 | -0.12 |
| `DBLP` | 0.87 | 0.99 | 0.81 | 0.99 | -0.13 |
| `InternetAS` | 0.98 | 1.00 | 0.76 | 0.99 | -0.07 |

Table 3.2: **Power law patterns across multiple datasets:** as a function of $k$, $\mu(k)$ and $\sigma(k)$ closely follow the power-law patterns, (3.3) and (3.4). Dataset information is given in Table 10.3.

cally, so we approximate $\xi$ by treating it as a constant as a function of $k$. Then $\text{GPD}_{\xi,\mu,\sigma}(\tilde{e}(S))$ is a function of $\frac{\tilde{e}(S)-\mu}{\sigma}$; moreover, since the function is a CDF, it has to be a strictly increasing function of $\frac{\tilde{e}(S)-\mu}{\sigma}$. Thus, we can simplify our measure by using $\frac{\tilde{e}(S)-\mu}{\sigma}$ in place of $\text{GPD}_{\xi,\mu,\sigma}(\tilde{e}(S))$.

Substituting the power law patterns (3.3) and (3.4):

$$\frac{\tilde{e}(S) - \mu(k)}{\sigma(k)} = \frac{\tilde{e}(S) - \mu_0 k^{\alpha}}{\sigma_0 k^{\beta}}$$

The constant factor $\sigma_0$ can be ignored, resulting in our expression for TellTail:

**Definition 3.5: Proposed TellTail Measure**

The surprisingness of subgraph $S$ with adjusted mass $\tilde{e}(S)$ and size of $k$ nodes is:

$$\boxed{f(S) = \frac{\tilde{e}(S) - \mu_0 k^\alpha}{k^\beta}} \tag{3.5}$$

Following Observation 3.2, we set $\alpha = 0.9$ and $\beta = 0.8$.

**Estimating $\mu_0$**

Let $A$ be the adjacency matrix, $d$ be the column vector of node degrees, $B = A - d \cdot d^T/(2m)$, and $s = \lfloor n/2 \rfloor$. For brevity, the derivation of $\mu_0$ under a normal approximation is in our supplementary document [sup], and the final value for $\mu_0$ is:

$$\begin{aligned} \mu_0 = s^{-\alpha}(p_2 S_1 + z_{1-\epsilon}(p_2 S_2 + p_3(S_3 - 2S_2) \\ + p_4(S_1^2 + S_2 - S_3) - (p_2 S_1)^2)^{1/2}) \end{aligned} \tag{3.6}$$

where: $S_1 = \sum_{i<j} B_{ij}, S_2 = \sum_{i<j} B_{ij}^2, S_3 = \sum_{i=1}^n (\sum_{j=1}^n B_{ij})^2, p_r = \prod_{j=1}^r (s-j+1)/\prod_{j=1}^r (n-j+1)$, and $z_{1-\epsilon}$ is the $(1-\epsilon)$-quantile of a standard normal distribution.

### 3.6.5 Basic Optimization Algorithm

TellTail-Search, our basic approach for optimizing TellTail, uses randomized local search. It is given in Algorithm 3.1, if we omit the pruning steps (lines 1 to 3 and 9; in green).

### 3.6.6 Improved Algorithm: TellTail-Search+

We now show that the form of our TellTail measure allows us to safely ignore less important nodes, while guaranteeing that we never ignore nodes that are in the subset that optimizes TellTail. This allows us to improve the accuracy and speed of our search algorithm by pruning away (i.e. removing) some of the nodes.

Recall that $B = A - d \cdot d^T/(2m)$ is the modularity matrix.

**Definition 3.6: Deviation of a node**

The **deviation** of a node is the sum of the modularity matrix $B$ between it and its neighbors:

$$\text{Dev}_i = \sum_{i,j \in E} B_{ij} \tag{3.7}$$

Intuitively, deviation measures how much a node can 'contribute' to the adjusted mass of a subset. Let $S^* = \arg\max_{S \subseteq V} \textsc{TellTail}(S)$ and $s = \lfloor n/2 \rfloor$. Define:

$$\Delta(S) = (s^\beta - (s-1)^\beta)\textsc{TellTail}(S) + \mu_0(s^\alpha - (s-1)^\alpha) \tag{3.8}$$

Our pruning is based on the following theorem.

**Theorem 3.1: Safe Pruning**

For any node $i$, if $i \in S^*$, we have:

$$\text{Dev}_i \geq \Delta(S^*) \geq \Delta(S) \; \forall \; S \subseteq V \tag{3.9}$$

Thus, we can prune nodes with deviation less than $\Delta(S)$, for **any** $S$.

*Proof.* Define $k_* = |S^*|$, and $\gamma = k_*^\alpha - (k_* - 1)^\alpha$, and $\delta = k_*^\beta - (k_* - 1)^\beta$.

Since $\text{Dev}_i$ is the sum of positive modularity terms along the $i$th row of $B$, $\text{Dev}_i$ is an upper bound for how much adjusted mass the $i$th row can contribute to $\tilde{e}(S^*)$. Hence:

$$\textsc{TellTail}(S^*) \geq \textsc{TellTail}(S^* \setminus \{i\}) \tag{3.10}$$

$$\implies \frac{\tilde{e}(S^*) - \mu_0 \cdot k_*^\alpha}{k_*^\beta} \geq \frac{\tilde{e}(S^*) - \text{Dev}_i - \mu_0 \cdot (k_* - 1)^\alpha}{(k_* - 1)^\beta} \tag{3.11}$$

$$\implies \frac{\tilde{e}(S^*) - \mu_0 \cdot k_*^\alpha}{k_*^\beta} \geq \frac{\tilde{e}(S^*) - \text{Dev}_i - \mu_0 \cdot k_*^\alpha + \mu_0 \cdot \gamma}{k_*^\beta - \delta} \tag{3.12}$$

$$\implies -\delta \cdot \tilde{e}(S^*) + \mu_0 \cdot \delta \cdot k_*^\alpha \geq -k_*^\beta \cdot \text{Dev}_i + k_*^\beta \cdot \mu_0 \cdot \gamma \tag{3.13}$$

$$\implies \textsc{TellTail}(S^*) \leq \frac{\text{Dev}_i - \mu_0 \gamma}{\delta} \tag{3.14}$$

$$\implies \text{Dev}_i \geq \delta \cdot \textsc{TellTail}(S^*) + \mu_0 \cdot \gamma \tag{3.15}$$

$$\implies \text{Dev}_i \geq \Delta(S^*) \tag{3.16}$$

The second inequality $\Delta(S^*) \geq \Delta(S) \; \forall \; S \subseteq V$ follows from $\textsc{TellTail}(S^*) \geq \textsc{TellTail}(S)$, since this implies that $\Delta(S^*) \geq \Delta(S) \; \forall \; S \subseteq V$. ■

TellTail-Search+ first uses the neighborhoods of the $t$ highest degree nodes to construct bounds to prune the nodes (lines 1 to 3 of Algorithm 3.1). After each repetition of the search algorithm, we prune the nodes again (line 9).

## 3.7 Theoretical Results

### 3.7.1 Consistency

Our Tail estimate for the surprisingness of subgraphs is consistent: i.e. as the data size increases, the error of $f$ approaches zero. Formalizing this requires the notion of graphons [OR15]:

**Algorithm 3.1:** TellTail-Search+

**Input** : Graph $G$, no. of repetitions $r$, no. of neighborhoods $t$

**for** $i = 1$ **to** $t$ **do**
  Let $N_i$ be the neighborhood of $i$th highest degree node
  Prune away nodes with deviation less than $\Delta(N_i)$
**end**
**for** $i = 1$ **to** $r$ **do**
  Sample $p \sim \text{Uniform}([0, 1])$
  Initialize $S[i]$ to include each node with probability $p$
  **while** *not converged* **do**
    ▷Greedily add or remove a node of $S[i]$
    $S[i] \leftarrow \underset{S' : |S' \cup S[i]| - |S' \cap S[i]| \leq 1}{\arg\max} \text{TellTail}(S')$
  **end**
  Prune away nodes with deviation less than $\Delta(S[i])$
**end**
**Return** $S[j]$, where $j = \arg\max_j \text{TellTail}(S[j])$

**Definition 3.7: Graphon**

A graphon is a symmetric function $\mathcal{G} : [0, 1]^2 \to [0, 1]$. To sample a random graph $G$ of size $n$ from $\mathcal{G}$, for each vertex $i = 1, \ldots, n$, we sample an independent random $u_i \sim \text{Uniform}([0, 1])$. Then edge $(i, j)$ is independently included in $G$ with probability $\mathcal{G}(u_i, u_j)$.

As a simple example, setting $\mathcal{G}$ to a constant value of $p$ corresponds to Erdos-Renyi graphs with edge probability $p$.

Define an arbitrary graphon $\mathcal{G}$ to be the true model underlying the graph $G$, and fix $k$. Define the 'true surprisingness' function $F$ to be the CDF of the mass of a random graph of size $k$ drawn from $\mathcal{G}$. Based on $G$, Tail defines an estimate $\hat{F}$ of $F$, where $\hat{F}$ is the CDF of the maximum likelihood GP distribution in Section 3.6.1. Our consistency theorem states that $\hat{F}$ converges to $F$ in terms of relative error, as $n \to \infty$.

**Theorem 3.2: Consistency**

Let $G$ be a graph drawn from $\mathcal{G}$, and fix $k$. Define any fixed $z > 0$, and let $y_n = -\sigma(\hat{\mu}_n)(1-z)/\xi$. Then as $n \to \infty$, there exists a sequence[a] of $N_n \to \infty, \epsilon_n \to 0$ such that:

$$\frac{1 - \hat{F}(\hat{\mu}_n + y_n)}{1 - F(\hat{\mu}_n + y_n)} \xrightarrow{P} 1$$

where $\xrightarrow{P}$ denotes convergence in probability.

[a] $N_n$, $\epsilon_n$, $\hat{\mu}_n$ denote the original variables $(N, \epsilon, \hat{\mu})$ indexed over runs corresponding to different values of $n$. $\xi$ and $\sigma(\cdot)$ are from (3.2).

This says that $\hat{F}$ converges to $F$, where following [Smi87], convergence is with respect to relative error of the CCDF, $1-F$. Since high surprisingness is associated with very small values of the CCDF, this formulation guarantees that these small values are estimated accurately in terms of relative error.

*Proof.* Fix $N$ and let $n \to \infty$. Consider an $n$-node graph $G \sim \mathcal{G}$ and random subsets $S_1, \ldots, S_N$ of size $k$. For any $i, j$, the probability that $S_i$ and $S_j$ are completely disjoint is $(\frac{n-k}{n})^k \to 1$ as $n \to \infty$ (recall that $k$ is fixed). Extending this to all of the pairs of $i, j$ by union bound, the probability that all of $S_1, \ldots, S_N$ are disjoint goes to 1 as $n \to \infty$. This implies that with high probability, $S_1, \ldots, S_N$ are disjoint and hence are i.i.d. samples of size $k$ from $\mathcal{G}$, and their masses (denoted by $m_1, \ldots, m_N$) are i.i.d. samples from $F$. Note that the event in which $S_1, \ldots, S_N$ are non-disjoint has probability converging to zero, and since the statement we want to prove is about convergence in probability, this event can be ignored.

At this point, $m_1, \ldots, m_N$ is an i.i.d. sample from a distribution $F$, and our algorithm TAIL estimates a GP distribution using maximum likelihood from this sample. [Smi87] shows that under these conditions, the maximum likelihood procedure produces consistent estimators of the tail probabilities of the distribution $F$. Formally:

$$\frac{1 - \hat{F}(\hat{\mu_n} + y_n)}{1 - F(\hat{\mu_n} + y_n)} \xrightarrow{P} 1$$

i.e. $\hat{F}$ converges to $F$, measured in terms of relative error with respect to the CCDF $1 - F$. ■

### 3.7.2 Proposed Monotonicity Axioms

Next, we show that other than speed, TELLTAIL has advantages over sampling-based approaches in that it satisfies two intuitive 'axioms,' one of which TAILDC does not satisfy.

Consider a measure $f(S)$. Two natural axioms are:

**Axiom 3.1: Mass**

All else being equal, higher mass (i.e. number of edges) is more surprising: If $|S| = |S'|$ and $\tilde{e}(S) > \tilde{e}(S')$, then $f(S) > f(S')$.

**Axiom 3.2: Concentration**

All else being equal: subsets of smaller size (i.e. number of nodes) are more surprising: If $\tilde{e}(S) = \tilde{e}(S')$ and $|S| < |S'|$, then $f(S) > f(S')$.

**Theorem 3.3**

TellTail satisfies Axioms 1 and 2.

*Proof.* $f(S) = \frac{\tilde{e}(S) - \mu_0 k^\alpha}{k^\beta}$ is increasing in $\tilde{e}(S)$ for fixed $k$, satisfying Axiom 1. For Axiom 2, as $k$ increases, both $\mu_0 k^\alpha$ and $k^\beta$ increase, making $f(S)$ decrease. Thus for fixed $\tilde{e}(S)$, $f(S)$ is decreasing in $k$, satisfying Axiom 2. ■

TailDC also satisfies Axiom 1, since it fits a (strictly increasing) CDF $\hat{F}$. However, it does not satisfy Axiom 2, as the random sampling makes $f(S)$ not strictly decreasing in $k$. Thus, TellTail improves on the sampling approaches by removing the stochasticity of relying on random samples.

### 3.7.3 NP Completeness

In this subsection, we show that maximizing TellTail is NP-complete. Let $\tilde{e}_G(S)$ denote the adjusted mass of subset $S$ with respect to the graph $G$, i.e. $\tilde{e}_G(S) = e(S) - d(S)^2/(4m)$, all with respect to $G$. Define the following two problems:

**Problem 3.2: Modularity**

Given a graph $G$, does there exist a subset $S$ of its nodes that such $\tilde{e}_G(S) > 0$?

The NP-hardness of modularity maximization was first shown by [BDG+07], though this differs from the formulation here. The NP-hardness of this exact formulation was shown by [DLT15]. They do this by reducing the known NP-hard Partition problem, of partitioning $n$ integers $x_1, \cdots, x_n$ into two subsets of equal sum, to the Modularity problem. To do this, they show that given $x_1, \cdots, x_n$, we can construct a graph such that a subset $S$ of positive

modularity exists iff there exists a partition of $x_1, \cdots, x_n$ into two halves of equal sum, which completes the reduction and establishes the NP-hardness of Modularity.

The problem we are interested in is:

**Problem 3.3: TellTailProb**

Given a graph $G'$, does there exist a subset $S$ of its nodes such that $\textsc{TellTail}_{G'}(S) > 0$?

We now show our main NP-completeness result:

**Theorem 3.4**

TellTailProb is NP-complete.

*Proof.* Given a subset $S$ of the nodes of $G$, we can compute $\textsc{TellTail}(S)$ in polynomial time. Thus, TellTailProb can be *verified* in polynomial time, and is therefore in NP. It remains to establish that TellTailProb is NP-hard.

We do this by reducing Modularity to TellTailProb: given an algorithm $A'$ that solves TellTailProb, we show that it can be used as a subroutine in a polynomial-time algorithm $A$ to solve Modularity. Then, since we know that Modularity is NP-hard, this would imply that TellTailProb is NP-hard as well.

Consider an instance of Modularity with graph $G$. Construct a new graph $G'$ by adding $r$ extra nodes to $G$, in which the extra nodes have no edges attached to them. Note then that for any subset $S$ of the nodes of $G$, the adjusted mass of $S$ is the same when computed with respect to either $G$ or $G'$: this is because in the formula $\tilde{e}(S) = e(S) - d(S)^2/(4m)$, none of the terms ($e(S)$, $d(S)$ or $m$) differ between $G$ and $G'$.

Consider Eq. (3.6) for $\mu_0$. Each of the $p_i$ is at most 1, and the $S_i$ are all constant as we increase $r$, which we can verify from Eq. (6) to (8) of the original paper. Then, since $G'$ has $n+r$ nodes, we have from Eq. (3.6) that $\mu_0 \leq (\frac{n+r}{2})^{-\alpha} B$, for some $B > 0$ that does not depend on $r$.

Set $r > 2(4mn^\alpha B)^{(1/\alpha)}$. Then in $G'$, we have:

$$\begin{aligned}
\mu_0 &\leq \left(\frac{n+r}{2}\right)^{-\alpha} B \\
&< \left(\frac{r}{2}\right)^{-\alpha} B \\
&= (4mn^\alpha B)^{(1/\alpha)\cdot(-\alpha)} B \\
&= \frac{1}{4mn^\alpha}
\end{aligned}$$

Define the algorithm $A(G)$ for Modularity that given $G$, constructs $G'$, runs our subroutine for solving TellTailProb on $G'$, and outputs $A'(G')$. We claim that $A(G)$ correctly solves Modularity. To show this, we consider two cases:

- **case 1:** there exists $S$ such that $\tilde{e}_G(S) > 0$. Then since $\tilde{e}_G(S)$ is a fraction with an integer in the numerator and a denominator of $4m$, we have $\tilde{e}_G(S) \geq 1/(4m)$. Then, recalling that $\tilde{e}_G(S) = \tilde{e}_{G'}(S)$,

$$\begin{aligned}\textsc{TellTail}_{G'}(S) &= \frac{\tilde{e}_G(S) - \mu_0|S|^\alpha}{|S|^\beta} \\ &\geq \frac{\frac{1}{4m} - \mu_0|S|^\alpha}{|S|^\beta} \\ &\geq \frac{\frac{1}{4m} - \frac{1}{4mn^\alpha}|S|^\alpha}{|S|^\beta} \\ &> 0\end{aligned}$$

  Thus $A(G)$ outputs the correct result in this case: $A'(G')$ will return 'true' since $\textsc{TellTail}_{G'}(S) > 0$, so $A(G)$ will return 'true,' which is correct since $\tilde{e}_G(S) > 0$.
- **case 2:** there does not exist such an $S$; i.e. $\tilde{e}_G(S) \leq 0$ for all $S$. Then for all $S$,

$$\textsc{TellTail}_{G'}(S) = \frac{\tilde{e}_G(S) - \mu_0|S|^\alpha}{|S|^\beta} \leq \frac{\tilde{e}_G(S)}{|S|^\beta} \leq 0$$

  Thus $A(G)$ returns 'false', which is the correct result in this case as well.

In conclusion, $A(G)$ is a correct, polynomial time algorithm for Modularity, assuming we have a subroutine $A'$ that solves TellTailProb. Since Modularity is NP-hard by [DLT15], this implies that TellTailProb is NP-hard as well. Since we also showed that TellTailProb is in NP, this implies that it is NP-complete. ■

Our experiments answer the following questions:

- **Q1. Scalability:** how does TellTail+ scale with data size?
- **Q2. Accuracy of Measure:** does TailF accurately identify injected dense subgraphs?
- **Q3. Real-World Effectiveness:** do TailF and TellTail+ accurately find ground-truth communities in real graphs?

Datasets details are in Table 10.3. The first 8 datasets are available online at KONECT [Kun13]. The next 5 are topical Twitter graphs with ground truth communities [GC13]; we filter excessively small communities ($< 10$ nodes). We use `WikiP` [kon17] for testing scalability, and `Twitter` [KLPM10] as a fraud detection case study.

### 3.7.4 Q1. Scalability

To get graphs that follow real-world patterns, rather than using a synthetic generator, we use the real `WikiP` graph and extract subsets of its nodes. Hence, we run TellTail+ on random subsets of the `WikiP` graph containing $\lfloor (0.85)^k \rfloor$ fraction of nodes for $k = 0, \ldots, 9$, averaging each over 10 trials. Figure 3.1b shows that TellTail+ scales near-linearly: the blue line indicates linear growth. TellTail+ is fast, taking $0.41$ seconds (using a laptop computer) on the full graph of $49.0$ million edges.

| | Nodes | Edges | Node Info | Edge Info | Communities |
|---|---|---|---|---|---|
| UCForum [LKF07] | 899 | 34K | User/Forum | Post | - |
| Email [CHC+07] | 1.1K | 5K | User | Email | - |
| Blogs [AG05] | 1.2K | 19K | Blog | Hyperlink | - |
| Petster [Kun13] | 2.4K | 17K | User | Friendship | - |
| PGP [BPSDGA04] | 11K | 24K | User | Interaction | - |
| DBLP [Ley02] | 13K | 50K | Publication | Citation | - |
| AstroPh [LKF07] | 18K | 198K | Author | Coauthor | - |
| InternetAS [ZLMZ05] | 34K | 171K | A.S. | Connection | - |
| Football [GC13] | 248 | 3.8K | User | Follow | 17 |
| PoliticsIE [GC13] | 348 | 17K | User | Follow | 4 |
| PoliticsUK [GC13] | 419 | 27K | User | Follow | 3 |
| Olympics [GC13] | 464 | 11K | User | Follow | 18 |
| Rugby [GC13] | 854 | 36K | User | Follow | 8 |
| WikiP [kon17] | 1.6M | 49M | Page | Link | - |
| Twitter [KLPM10] | 41.7M | 1.47B | User | Follow | - |

Table 3.3: Datasets used in our experiments

### Pruning Effectiveness

How effective are the pruning steps in TellTail+ in reducing the size of the graph? Figure 3.6 shows that TellTail+ reduces the number of nodes by a factor of 45, 18 and 52 on the largest InternetAS, WikiP and Twitter datasets respectively.

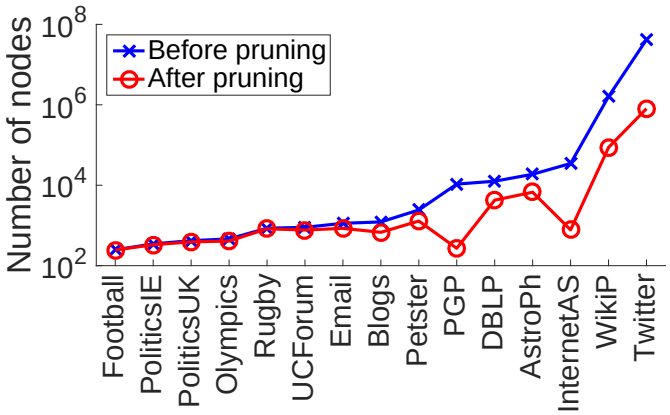



Figure 3.6: **TellTail+ reduces graph size.**

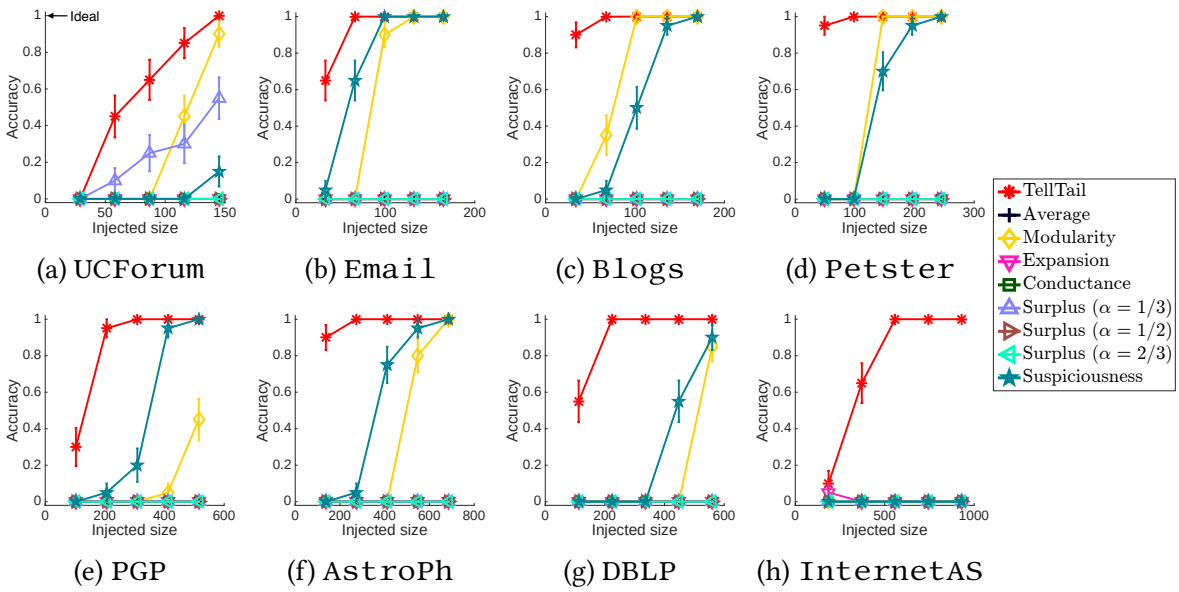



Figure 3.7: **TailF outperforms baselines in accuracy** for identifying injected subgraphs.

### 3.7.5 Q2. Accuracy of Measure

In this section, we evaluate the accuracy of our TailF measure in distinguishing injected dense subgraphs from normal subgraphs. Note that our goal here is to evaluate its accuracy as a measure, against baseline measures, rather than against detection algorithms. Hence, our baseline measures are chosen to provide a mix of standard internal density measures (average degree, modularity), measures which consider internal and external density (expansion, conductance) and recently proposed dense subgraph measures (edge surplus [TBG+13], suspiciousness [JBC+16]). Define $c(S) = |\{(u, v) \in E : u \in S, v \notin S\}|$.

- *Average Degree* [RCC+04]: $f(S) = 2e(S)/|S|$ is the average internal degree of the nodes in $S$.
- *Modularity* [New06]: $f(S) = \frac{1}{m}e(S) - (\frac{d(S)}{2m})^2$ is the fraction of links in $S$ minus its expected value.
- *Expansion* [RCC+04]: $f(S) = c(S)/|S|$ is the number of edges per node that cross the boundary of $S$.
- *Conductance* [SM00]: $f(S) = \frac{c(S)}{2e(S)+c(S)}$ is the fraction of the edge volume of $S$ crossing its boundary.
- *Edge Surplus* [TBG+13]: $f(S) = e(S) - \alpha\binom{|S|}{2}$. Following [TBG+13] we use $\alpha = 1/3$, but also include $1/2$ and $2/3$.
- *Suspiciousness* [JBC+16]: $f(S) = -\log P(Y = e(S))$: $Y$ is the mass of $S$ under an Erdos-Renyi model.

While edge density $e(S)/\binom{|S|}{2}$ is intuitive, it does not directly work as a single edge attains the maximum possible density of 1. Average Degree, Modularity and Edge Surplus can be considered as different approaches for adjusting edge density to alleviate this problem.

For each graph, in each trial we inject a dense subgraph, whose nodes are uniformly sampled at random from the graph's existing nodes, and whose size is one of $\lfloor\sqrt{n}\rfloor, 2\lfloor\sqrt{n}\rfloor, \cdots, 5\lfloor\sqrt{n}\rfloor$ (we obtain separate results for each of these sizes). We choose these values as it tends to be the range where performance varies meaningfully. We inject edges uniformly at random to this subgraph, adding additional density equal to twice the average density of the whole graph (i.e. density $2m/\binom{n}{2}$). A good measure should distinguish 'unnatural' (injected) subgraphs from 'natural' (non-injected) subgraphs. Thus, in each trial we generate $500$ random null subgraphs: their sizes follow an evenly distributed grid of sizes between $1$ and $n$, then the nodes of each null subgraph are chosen uniformly at random. A measure is successful on a trial if it gives a higher value to the injected subgraph than to all the null subgraphs. We repeat each trial $20$ times to generate error bars (representing $1$ standard deviation).

Figure 3.7 shows accuracy (i.e. the fraction of trials where the injected subgraph received the highest score) against injected size. TailF clearly outperforms the baselines, with accuracy near $1$ for many cases. Why does this happen? In Section 3.5.1 we introduced **size-bias**: many measures do not compare across sizes in a balanced manner: e.g. average degree chooses large subsets of almost the whole graph, while modularity chooses subsets of around half the graph size. TailF evaluates subgraphs of each size using an accurate GP model, allowing fair comparison across sizes.

### 3.7.6 Q3. Real-World Effectiveness

We next evaluate TailF on topically curated Twitter graphs with ground truth dense communities corresponding to sports teams or political parties manually labelled by [GC13]. On each dataset, we randomly generate a pool of null subgraphs: for each size (twice), we generate $200$ random subgraphs of that size and add the community with highest mass into the pool. This ensures that the null subgraphs are reasonably dense. We then compute each measure on the ground truth and null subgraphs. For each measure, its accuracy is its precision at $k$, i.e. the fraction of ground truth communities in the top $k$ subsets according to the measure, where $k$ is the true number of ground truth communities. Table 3.4 shows that TailF clearly outperforms the baselines, and has more consistent performance across datasets. Many of the baselines have highly variable performance, e.g. some high and some low accuracies, which suggests that they are not comparing subgraphs of different sizes in a balanced manner.

#### Effectiveness of our Search Algorithms

We now evaluate TellTail and TellTail+ on detecting ground truth communities in the same topical Twitter graphs. We use the same baselines, each optimized using standard local search [JBC+16]. Note that local search is a standard approach for optimizing most of these metrics: [Cha00, New06, TBG+13, JBC+16]. We run each algorithm $10$ times and choose the subset which it gave the highest score to. We evaluate each method based on the largest Jaccard similarity between its output and any ground truth community.

The results are shown in Table 3.5. Averaging over the 5 graphs, TellTail and TellTail+ outperforms the best-performing baseline by $0.34$ and $0.31$ respectively. This occurs likely due

| | TailF | Average | Modularity | Expansion | Conduct. | Surp.(1/3) | Surp.(1/2) | Surp.(2/3) | Susp. |
|---|---|---|---|---|---|---|---|---|---|
| Football | **1.00** | 0.00 | 0.96 | 0.71 | 0.55 | 0.99 | **1.00** | **1.00** | 0.68 |
| PoliticsIE | **1.00** | 0.00 | 0.94 | 0.87 | 0.86 | 0.16 | 0.75 | 0.86 | 0.38 |
| PoliticsUK | **0.96** | 0.46 | 0.87 | 0.54 | 0.87 | 0.46 | 0.71 | 0.87 | 0.29 |
| Olympics | **1.00** | 0.00 | 0.85 | 0.74 | 0.67 | 0.99 | **1.00** | **1.00** | 0.63 |
| Rugby | **0.97** | 0.00 | 0.85 | 0.90 | 0.80 | **0.97** | 0.94 | 0.94 | 0.55 |

Table 3.4: **TailF outperforms baselines** in identifying ground truth communities.

to their use of TailF, which also performs better in identifying ground truth communities in Table 3.4.

| | TellTail | TellTail+ | Average | Modularity | Expansion | Conduct. | Surp.(1/3) | Surp.(1/2) | Surp.(2/3) | Susp. |
|---|---|---|---|---|---|---|---|---|---|---|
| Football | **0.78** | 0.67 | 0.11 | 0.19 | 0.06 | 0.11 | 0.13 | 0.15 | 0.20 | 0.13 |
| Politics IE | **0.90** | 0.82 | 0.42 | 0.39 | 0.02 | 0.41 | 0.42 | 0.41 | 0.40 | 0.39 |
| Politics UK | 0.82 | **0.84** | 0.81 | 0.75 | 0.01 | 0.36 | 0.51 | 0.53 | 0.81 | 0.76 |
| Olympics | 0.68 | **0.72** | 0.20 | 0.21 | 0.11 | 0.15 | 0.22 | 0.29 | 0.44 | 0.22 |
| Rugby | 0.47 | **0.49** | 0.43 | 0.24 | 0.36 | 0.17 | 0.25 | 0.03 | 0.03 | 0.28 |

Table 3.5: **TellTail and TellTail+ outperform baselines** in identifying ground truth communities.

### 3.7.7 Case Study on Twitter Data

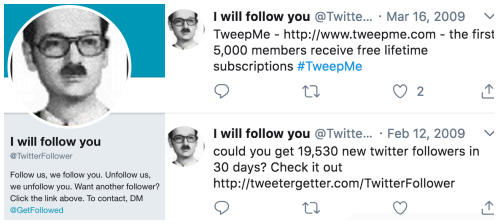



Figure 3.8: **TellTail+ detects a follower-boosting scheme:** we detect a large group of accounts, 85% of which are directly connected to the pictured account.

We apply TellTail+ on a Twitter following graph of 41.7M users and 1.47B follows [KLPM10]. TellTail+ detects a group of 4334 users with edge density 75%, which is highly suspicious in itself.

To further analyze the users in this block, we first take a random sample of 400 of these users. We then exclude users whose accounts were deleted, suspended, or not searchable, resulting in 280 remaining users. Of these, we use a script to search for which of these users have made tweets linking either 'tweepme.com' or 'tweetergetter.com,' which are the names of two known follower-buying services in which users who purchase the service follow one another (which would explain the unusually dense subgraph). We find that 125 (45%) of these users have made such tweets, and still remain undeleted in the 10 years since this dataset was collected.

The true number of users in this group engaging in follower-buying services is likely higher: users may have cleared their old tweets, been suspended or deleted, or use other follower-buying services. Indeed, we find that 85% of the users in this block are directly adjacent to the user in Figure 3.8, which seems to be a user who was created for the purpose of reciprocal following and amplifying follower-buying schemes: this user (who still exists without being suspended) advertises over 20 follower-boosting services, such as 'TweepMe' and 'TweeterGetter'.

## 3.8 Conclusion

In this chapter, we introduced principled scoring functions for dense subgraphs. To sum up, a persistent question in dense subgraph settings is: how can we fairly compare a very dense subgraph of size 10, to a less dense subgraph of size 10000? Which is more surprising, and thus preferable to bring to the user's attention? Our answer is to evaluate each subgraph against the empirical distribution of subgraphs of the same size, producing a probabilistic score, and then to compare these scores. This allows us to treat subgraphs of all sizes fairly, rather than being 'size-biased' (i.e. giving high scores more often to subgraphs of particular sizes). Our approach for doing this is based on the GP distribution, which closely fits the empirical distribution of subgraph masses.

Our contributions are as follows:

- **Theoretical underpinnings:** we propose a probabilistic framework (see Definition 3.5) for finding dense subgraphs based on extreme value theory. Theorem 3.2 provides a guarantee of consistency.
- **Discoveries:** We make two novel empirical observations: Observations 1 and 2, on dense subgraph patterns in real graphs, which we use to speed up our TellTail measure.
- **Practicality:** Our approach outperforms existing measures in synthetic (Figure 3.7) and ground truth (Tables 3.4 and 3.5) settings.
- **Search algorithm:** Theorem 3.1 allows us to safely prune away graph nodes, which improves the accuracy of our search algorithm, TellTailSearch+.

**Reproducibility**: Our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/telltail/`.

# Chapter 4

# FRAUDAR: Fraud Detection in an Adversarial Setting

Chapter based on work that appeared at KDD16 [HSB+16b] [PDF].

Given a bipartite graph of users and the products that they review, or followers and followees, how can we detect fake reviews or follows? Fraudsters often evade detection using *camouflage*, by adding reviews or follows with honest targets so that they look "normal".

To spot fraudsters in the presence of camouflage or hijacked accounts, we propose FRAUDAR, an algorithm that (a) is camouflage-resistant, (b) provides upper bounds on the effectiveness of fraudsters, and (c) is effective in real-world data.

## 4.1 Introduction

How can we detect if a politician has purchased fake followers on Twitter, or if a product's reviews on Amazon are genuine? More challengingly, how can we *provably* prevent fraudsters who sell fake followers and reviews for various web services from evading our detection systems? In this chapter we focus on precisely this problem – specifically, how can we design a fraud detection system with strong, provable guarantees of robustness?

Given the rise in popularity of social networks and other web services in recent years, fraudsters have strong incentives to manipulate these services. On several shady websites, anyone can buy fake Facebook page-likes or Twitter followers by the thousands. Yelp, Amazon and TripAdvisor fake reviews are also available for sale, misleading consumers about restaurants, hotels, and other services and products. Detecting and neutralizing these actions is important for companies and consumers alike. The tell-tale sign of such fraudulent actions is that fraudsters must add many edges, creating unusually *large* and *dense* regions in the adjacency matrix of the graph (see Figure 4.2). Smart fraudsters will also try to 'look normal', by adding links to popular items/idols (like famous singers/actors, or well-liked products) - this behavior is called "*camouflage*" in the recent literature. State-of-the-art algorithms, such as SPOKEN [PSS+10] and NETPROBE [PCWF07] exploit exactly the density signal, but do not account for "camouflage."

We propose FRAUDAR, a novel approach for successfully detecting fraudsters under camouflage, and we give *provable* limits on undetectable fraud. We provide data-dependent limits on the maximum number of edges a group of fraudulent adversaries can have without being detected, on a wide variety of real world graphs.

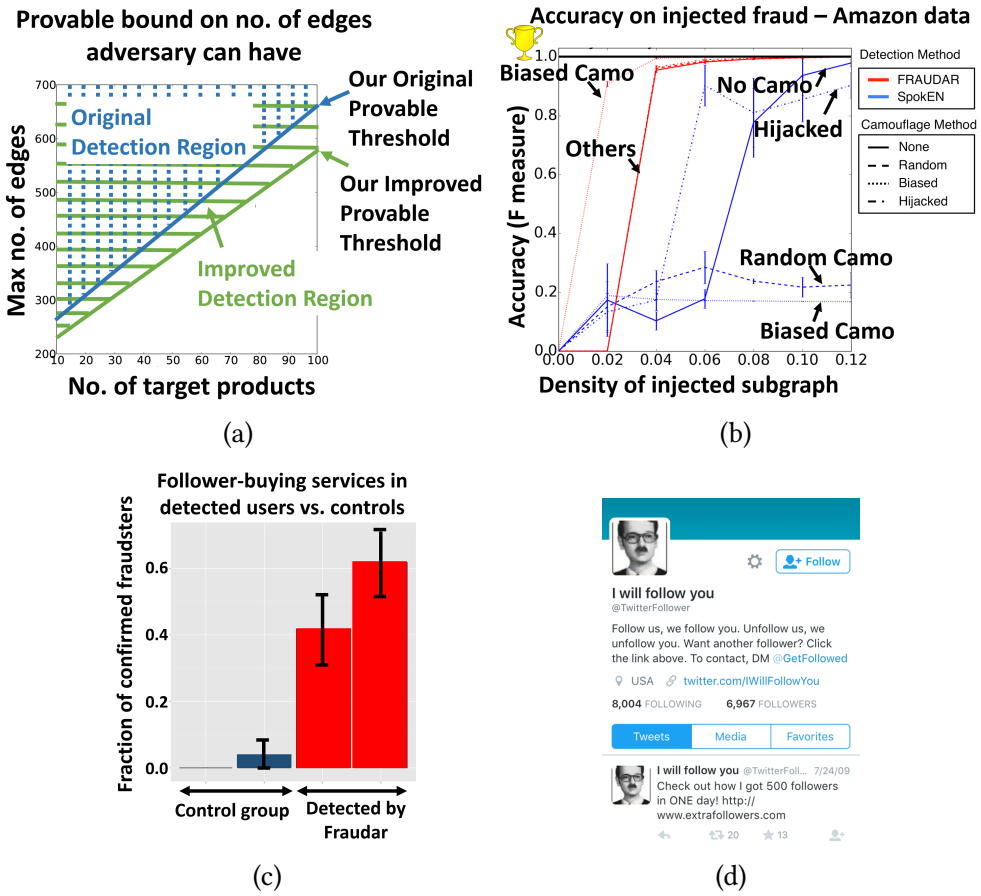



Figure 4.1: **(a) FRAUDAR detects more fraudsters:** the detection region indicates where a fraudster will be guaranteed to be caught by our approach. **(b) Accuracy:** FRAUDAR outperforms baselines. **(c) Confirmed fraudsters**: a large fraction of our detected followees (left red bar) and followers (right red bar) compared to almost none among non-flagged accounts (2 control groups). Confirmation was done by inspecting Tweets that advertise '*TweepMe*' or '*TweeterGetter*'. **(d) Real-life results:** a sample fraudster caught.

As shown in Figure 4.1(a), FRAUDAR provides limits on undetectable fraud, and additionally provides novel optimizations that strengthen this bound.

Moreover, our method outperforms competitors and finds real world fraud on Twitter. In Figure 4.1(b) we find that FRAUDAR detects injected fraud with high accuracy, even in the case of camouflage, where prior methods struggle to detect fraudulent attacks. Additionally, when tested on a Twitter graph from 2009, FRAUDAR finds a $4031$ by $4313$ subgraph that is $68\%$ dense. As shown in Figure 4.1(c-d), we find that *a majority* of the detected accounts had tweets showing

that they used follower-buying services, and had gone undetected by Twitter for the 7 years since the data was collected. Finally, our method is scalable, with near linear runtime in the data size.

Thus, our main contributions are as follows:

- **Metric** we propose a novel family of metrics which satisfies intuitive "axioms" and has several advantages as a suspiciousness metric.
- **Theoretical Guarantees** we provide a provable bound on how much fraud an adversary can have in the graph without being caught, even in the face of camouflage. Additionally, we improve the theoretical bound through novel optimizations that better distinguish fraud and normal behavior in real-world data.
- **Effectiveness** FRAUDAR outperforms state-of-the-art methods in detecting various fraud attacks in real world graphs, and detects a large amount of previously undetected fraudulent behavior on Twitter.
- **Scalability** FRAUDAR is scalable, with near-linear time complexity in the number of edges.

Furthermore, FRAUDAR offers natural extensibility and can easily incorporate more complex relations available in certain contexts such as review text, IP addresses, etc.

## 4.2 Background and Related Work

Fraud detection has received significant focus in recent years. Many existing methods aim to detect fraud through review text [OCCH11, JL08]. However, these approaches are typically not adversarially robust: spammers can carefully select their review texts to avoid detection. Even without knowledge of the detection system, they may mimic normal user reviews as closely as possible. Graph-based approaches detect groups of spammers, often by identifying unexpectedly dense regions of the graph of users and products. Such methods are potentially harder to evade, as creating fake reviews unavoidably generates edges in the graph. Graph-based methods may be classified into global and local methods.

**Global methods:** Building on singular value decomposition (SVD), latent factor models, and belief propagation (BP), these model the entire graph to find fraud. SPOKEN [PSS+10] considered the "eigenspokes" pattern produced by pairs of eigenvectors of graphs, and was later generalized for fraud detection [JCB+14b]. FBOX [SBGF14] builds on SVD but focuses on detecting attacks missed by spectral techniques. Several methods have used HITS [Kle99]-like ideas to detect fraud in graphs [JCB+14a, GGMP04, CSYP12, GVK+12, WGD06]. BP has been used for fraud classification on eBay [PCWF07], and fraud detection [ACF13]. All of these methods have been successful in finding fraud but they offer no guarantees of robustness. [SBGF14] performs adversarial analysis for spectral algorithms, showing that attacks of small enough scale will necessarily evade detection methods which rely on the top $k$ SVD components.

**Local clustering methods:** A different direction for fraud detection focuses on local subgraphs, by analyzing the properties of egonets to detect fraud [CPV01, PAISM14]. COPYCATCH [BXG+13] and GETTHESCOOP [JCB+14b] use local search heuristics to find relevant dense bipartite subgraphs. However, without guarantees on the search algorithm, the algorithms may not be robust to intelligent adversaries.

| *Property* | CopyCatch | CatchSync | BP-based | SpokEn | fBox | GetTheScoop | Fraudar |
|---|---|---|---|---|---|---|---|
| **Detects dense blocks** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✔ |
| **Camouflage-resistant** | ✓ | ? | | | ? | | ✔ |
| **Theoretical guarantees** | | | | | | | ✔ |

Table 4.1: Comparison between Fraudar and other fraud detection algorithms.

**Dense subgraph mining:** Finding dense subgraphs has been an important focus of graph theory communities and has been studied from a wide array of perspectives [GTV11, KK98]. Most closely related to ours is Charikar's work on finding subgraphs with large average degree [Cha00], which shows that subgraph average degree can be optimized with approximation guarantees. Variants have been proposed to efficiently find large, dense subgraphs [Tso15], with approximation guarantees. To our knowledge, however, this is the first work which adapts this theoretical perspective to the challenges of fraud detection and camouflage resistance, and achieves meaningful bounds for our application. Moreover, our work differs from these in its setting of bipartite graphs, and in the use of edge re-weighting to further increase accuracy.
**Social network-based Sybil defense:** Multiple identity or 'Sybil' attacks pose problems of malicious behavior in distributed systems. SybilGuard [YKGF06] and SybilLimit [YGKX08] use a decentralized random walk approach to limit the number of Sybil attackers. SumUp [TMLS09] and Iolaus [MKKSM13] adapt this to content rating settings. However, these systems rely on a separate trust network between users; our setting is fundamentally different as our approach works directly with the user-product bipartite graph.
**Handling camouflage:** [GPW+, VD06] consider fraud detection methods that are robust to camouflage attacks. However, both methods focus on the time-series domain, observing changes in the behavior of fraudsters from system access logs rather than graph data.

A comparison between Fraudar and other fraud detection algorithms is summarized in Table 10.1. Our proposed method Fraudar is the only one that matches all specifications.

## 4.3 Problem Definition

Consider a set of $m$ users $\mathcal{U} = \{u_1, \ldots, u_m\}$ and $n$ objects $\mathcal{W} = \{w_1, \ldots, w_n\}$ connected according to a bipartite graph $G = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$. We can consider the objects to be followees on Twitter or products on Amazon. Table 8.2 gives a complete list of the symbols we use throughout the paper. We now describe our attack model and then our problem definition.

**Attack model** We assume that fraudsters are hired to use users they control to add edges pointing to a subset of nodes in $\mathcal{W}$. For example, a business may pay for followers on Twitter

| Symbol | Interpretation |
|---|---|
| $\mathcal{U} = \{u_1, \ldots, u_m\}$ | Users |
| $\mathcal{W} = \{w_1, \ldots, w_n\}$ | Objects |
| $\mathcal{V}$ | Nodes of bipartite graph: $\mathcal{U} \cup \mathcal{W}$ |
| $G$ | Bipartite graph $G = (\mathcal{V}, \mathcal{E})$ |
| $\mathcal{A}$ | Subset of users |
| $\mathcal{B}$ | Subset of objects |
| $\mathcal{S}$ | Subset of nodes, $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$ |
| $g(\mathcal{S})$ | Density metric |
| $f(\mathcal{S})$ | 'Total suspiciousness' metric (Eq. (4.2)) |
| $\mathcal{X}$ | Current set of nodes in the greedy algorithm |
| $\Delta_i$ | $f(\mathcal{X} \setminus \{i\}) - f(\mathcal{X})$ |
| $\hat{\mathcal{A}}, \hat{\mathcal{B}}$ | Users (resp. objects) returned by FRAUDAR |
| $m_0, n_0$ | No. of users (resp. objects) in fraud block |
| $d_i$ | $i$th column sum of adjacency matrix |
| $\lambda$ | Min. fraction of fraud edges per customer |
| $g_{\log}$ | Logarithmic weighted metric |

Table 4.2: Symbols and Definitions

or positive reviews on Yelp. In general, fraudsters add a large number of edges, inducing a dense subgraph between the fraudster accounts and customers, as shown in the bottom right corner of each subplot of Figure 4.2. This general characteristic of fraud was found to be true in our experiments on real datasets, as well as in many other papers which use dense blocks to detect fraud [BXG+13, PSS+10, JCB+14b, PCWF07, ACF13].

To mask the fraud, fraudster accounts can add arbitrary "camouflage", i.e. edges pointing from their user accounts to any of the nodes in $\mathcal{W}$ that are not customers. We assume that fraudsters have *complete* knowledge of the graph and fraud detection mechanisms, enabling worst-case camouflage for any fraud detection system we create. Examples of the possible types of camouflage are given in Figure 4.2: (a) adding camouflage edges to random honest users, (b) camouflage biased toward high degree nodes, (c) using hijacked accounts, whereby fraudster accounts have realistic patterns of camouflage essentially similar to that of honest users.

While it is trivial for fraud accounts to add edges to any other node, it is more difficult for customer accounts to get honest edges. In particular, we assume that a customer would try to increase their number of incoming edges by a significant portion, and as a result a fraction, $\lambda \in [0, 1]$, of their incoming edges will be from fraudsters. This assumption would manifest itself as customers wanting to boost their follower count to seem *noticeably* more popular or a restaurant wanting a *significant* number of positive ratings to shift its average "number of stars" on Yelp. We will demonstrate how using this real world pattern significantly improves fraud detection both theoretically and in practice.

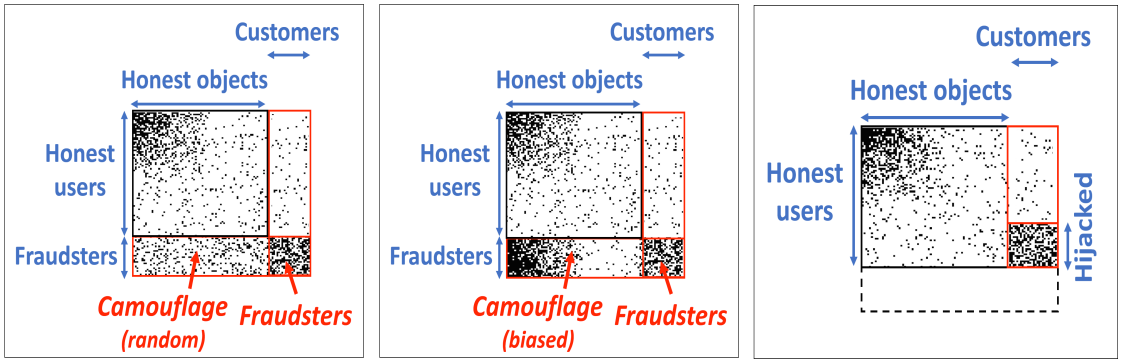



Figure 4.2: **Three examples of possible attacks**: (a) random camouflage; (b) biased camouflage; (c) hijacked accounts.

**Desired properties of detection approach** Our goal is to detect dense subgraphs in $G$, typically indicative of fraudulent groups of users and objects, like in the bottom-right of each subfigure in Figure 4.2.

**Informal Problem 4.1**

Given a bipartite graph, detect attacks so as to minimize the number of edges that fraudsters can add pointing to customers without being detected.

Given that we want our detection algorithm to be able to handle camouflage, we define the requirements for a *camouflage-resistant* algorithm:

**Definition 4.1**

Let $(\mathcal{A}, \mathcal{B})$ be a block consisting of fraudulent users and objects. A density metric $g$ is *camouflage-resistant* if when any amount of camouflage is added by the adversary, $g(\mathcal{A} \cup \mathcal{B})$ does not decrease.

That is, fraudsters cannot make themselves less suspicious by adding camouflage. Our goal is to find a fraud detection approach satisfying the following criteria:

**Problem 4.1: Dense Subgraph Detection**

Design a class of density metrics for bipartite graphs, which can be optimized (1) in near-linear time, (2) within a constant factor of the optimum, and (3) is minimally affected by camouflage edges added by adversaries.

Obtaining theoretical guarantees on the near-optimality of the returned subgraph is important because, as we will later show, it allows us to offer guarantees against *worst-case* fraudsters.

## 4.4 Proposed Method

Given this problem definition and attack model, we now offer FRAUDAR and our theoretical analysis of FRAUDAR.

### 4.4.1 Metric

In this section, we propose a class of metrics $g$ that have particularly desirable properties when used as suspiciousness metrics. Namely, we will show that if $g$ takes the form in (4.1) and (4.2), then it can be optimized in a way that is (a) scalable; (b) offers theoretical guarantees, and (c) is robust to camouflage.

Let $\mathcal{A} \subseteq \mathcal{U}$ be a subset of users and $\mathcal{B} \subseteq \mathcal{W}$ be a subset of objects. Let $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$, and $\mathcal{V} = \mathcal{U} \cup \mathcal{W}$. For the rest of this chapter, we use $g$ to denote the density metric that the algorithm will optimize, i.e. the algorithm will find $\mathcal{S}$ to (approximately) maximize $g(\mathcal{S})$. Note that $g$ has a single argument, which is the union of the users and objects whose suspiciousness we are evaluating.

We propose using density metrics $g$ of the following form:

$$g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|} \tag{4.1}$$

where total suspiciousness $f$ is:

$$\begin{aligned} f(\mathcal{S}) &= f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S}) \\ &= \sum_{i \in \mathcal{S}} a_i + \sum_{i,j \in \mathcal{S} \wedge (i,j) \in \mathcal{E}} c_{ij}, \end{aligned} \tag{4.2}$$

for some constants $a_i \geq 0$ and constants $c_{ij} > 0$.

Intuitively, the node suspiciousness $f_{\mathcal{V}}(\mathcal{S})$ is a sum of constants $a_i$ corresponding to the users and objects in $\mathcal{S}$, which can be thought of as how individually suspicious that particular user or object is. The edge suspiciousness $f_{\mathcal{E}}(\mathcal{S})$ is a sum of constants $c_{ij}$ corresponding to the edges in between $\mathcal{S}$, which can be thought of as how suspicious that particular edge is (e.g. the suspiciousness of the text of a review by user $i$ for object $j$).

There are many advantages to metrics of this form. Firstly, metrics of this form can be optimized in a way that is (a) scalable; (b) offers theoretical guarantees, and (c) is robust to camouflage, as we demonstrate in the rest of this chapter. All 3 of these properties hold due to the particular chosen form in (4.1) and (4.2).

Secondly, metrics of this form obey a number of basic properties (or "axioms") that we would intuitively expect a reasonable suspiciousness metric should meet, as we next show. These basic properties are adapted from the "axioms for suspiciousness metrics," proposed in [JBC+15], to our setting where node and edge weights exist.

**Axiom 4.1: Node Suspiciousness**

A subset consisting of higher suspiciousness nodes is more suspicious than one consisting of lower suspiciousness nodes, if the other conditions are fixed. Formally,

$$|\mathcal{S}| = |\mathcal{S}'| \quad \wedge \quad f_{\mathcal{E}}(\mathcal{S}) = f_{\mathcal{E}}(\mathcal{S}') \quad \wedge \quad f_{\mathcal{V}}(\mathcal{S}) > f_{\mathcal{V}}(\mathcal{S}') \Rightarrow g(\mathcal{S}) > g(\mathcal{S}')$$

**Axiom 4.2: Edge Suspiciousness**

Adding edges within a subset increases the suspiciousness of the subset if the other conditions are fixed. Formally,

$$e \notin \mathcal{E} \Rightarrow g(\mathcal{S}(\mathcal{V}, \mathcal{E} \cup \{e\})) > g(\mathcal{S}(\mathcal{V}, \mathcal{E}))$$

where $\mathcal{S}(\mathcal{V}, \mathcal{E})$ is the subgraph induced by $\mathcal{S}$ in the graph $(\mathcal{V}, \mathcal{E})$.

The **edge density** $\rho(\mathcal{S})$ of an induced subgraph is its number of edges divided by its maximum possible number of edges.

**Axiom 4.3: Size**

Assuming node and edge weights are all equal, larger subsets are more suspicious than smaller subsets with the same edge density. Formally, given $a_i = a \; \forall \; i$, and $c_{ij} = b \; \forall \; (i, j) \in \mathcal{E}$:

$$|\mathcal{S}| > |\mathcal{S}'| \quad \wedge \quad \mathcal{S} \supset \mathcal{S}' \quad \wedge \quad \rho(\mathcal{S}) = \rho(\mathcal{S}') \Rightarrow g(\mathcal{S}) > g(\mathcal{S}')$$

**Axiom 4.4: Concentration**

A subset with smaller size is more suspicious than one with the same total suspiciousness but larger size. Formally,

$$|\mathcal{S}| < |\mathcal{S}'| \quad \wedge \quad f(\mathcal{S}) = f(\mathcal{S}') \Rightarrow g(\mathcal{S}) > g(\mathcal{S}')$$

Density metrics $g$ of the form defined in Equation (4.1) satisfy these properties:

**Theorem 4.1**

The density metric defined in (4.1) satisfies axioms 4.1 to 4.4.

*Proof.* **Axiom 4.1 (Node Suspiciousness)**

$$g(\mathcal{S}) = \frac{f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S})}{|\mathcal{S}|}$$
$$> \frac{f_{\mathcal{V}}(\mathcal{S}') + f_{\mathcal{E}}(\mathcal{S}')}{|\mathcal{S}|} = \frac{f_{\mathcal{V}}(\mathcal{S}') + f_{\mathcal{E}}(\mathcal{S}')}{|\mathcal{S}'|} = g(\mathcal{S}').$$

**Axiom 4.2 (Edge Suspiciousness)** Let $e = (u, v)$.

$$g(\mathcal{S}(\mathcal{V}, \mathcal{E} \cup \{e\})) = \frac{f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S}) + c_{uv}}{|\mathcal{S}|}$$
$$> \frac{f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S})}{|\mathcal{S}|} = g(\mathcal{S}(\mathcal{V}, \mathcal{E})).$$

**Axiom 4.3 (Size)** Let $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$, and $\rho$ be the edge density.

$$g(\mathcal{S}) = \frac{f_{\mathcal{V}}(\mathcal{S}) + f_{\mathcal{E}}(\mathcal{S})}{|\mathcal{S}|} = a + b\left(\frac{\rho|\mathcal{A}||\mathcal{B}|}{|\mathcal{A}| + |\mathcal{B}|}\right)$$
$$= a + b\rho\left(\frac{1}{|\mathcal{A}|} + \frac{1}{|\mathcal{B}|}\right)^{-1}$$

which is increasing in both $|\mathcal{A}|$ and $|\mathcal{B}|$.

**Axiom 4.4 (Concentration)**

$$g(S) = \frac{f(\mathcal{S})}{|\mathcal{S}|} > \frac{f(\mathcal{S})}{|\mathcal{S}'|} = \frac{f(\mathcal{S}')}{|\mathcal{S}'|} = g(\mathcal{S}').$$

■

Note some simple metrics that violate these axioms: the edge density $\rho(\mathcal{S})$ itself, as a metric, does not increase with the size of $\mathcal{S}$ and hence violates axiom 4.3. On the opposite end, the total edge weight function $\sum_{i,j\in\mathcal{S}\wedge(i,j)\in\mathcal{E}} c_{ij}$ as a metric violates axiom 4.4 as it does not consider how concentrated the edge weight is. In contrast, $g$ scales in a reasonable manner as its size or concentration changes.

A simple example of a metric $g$ as defined in (4.1) and (4.2) is the bipartite graph average degree:

**Example 1.** *(Bipartite Graph Average Degree) Let $a_i = 0$, and let $c_{ij} = 1$ if $(i, j) \in \mathcal{E}$ and $0$ otherwise. In the expression (4.2) for $f(\mathcal{S})$, we add one term $c_{ij}$ for each edge $(i, j)$ for which $i, j$ are both in the subset $\mathcal{S}$. Thus, $f(\mathcal{S})$ is equal to the number of edges in the subgraph spanned by $\mathcal{S}$, or half the total degree in the subgraph spanned by $\mathcal{S}$. As a result, $g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|}$ is half the average degree of the subgraph spanned by $\mathcal{S}$.*

### 4.4.2 Algorithm

Let $f$ and $g$ be as given in (4.1) and (4.2). In this section, we give an algorithm for optimizing the density metric $g$ in near-linear time.

Algorithm 4.1 describes our proposed FRAUDAR algorithm, a greedy approach inspired by that of [Cha00] but which covers our broader objective class. We start with the entire set of nodes $\mathcal{U} \cup \mathcal{W}$, then repeatedly remove the node which results in the highest value of $g$ evaluated on the remaining set of nodes. Formally, denote by $\mathcal{X}$ the current set we are optimizing over; initially we set $\mathcal{X} = \mathcal{U} \cup \mathcal{W}$. Let $\Delta_i = f(\mathcal{X} \setminus \{i\}) - f(\mathcal{X})$ be the change in $f$ when we remove $i$ from the current set. At each step, we will select $i$ to maximize $\Delta_i$, i.e. to leave behind the set with highest value of $f$. We then remove $i$ from $\mathcal{X}$. We then repeat this process: we recompute the values of $\Delta_j$, then choose the next node to delete, and so on. This leads to a shrinking series of sets $\mathcal{X}$ over time, denoted $\mathcal{X}_0, \ldots, \mathcal{X}_{m+n}$ of sizes $m+n, \ldots, 0$. At the end, we return the one of these that maximizes the density metric $g$.

The key fact that allows the algorithm to be efficient is the forms for $f$ and $g$ in (4.1) and (4.2). When $i$ is removed, the only values of $\Delta_j$ which need to be updated are those where $j$ is a neighbor of $i$. This is because for all other $j$, the expressions (4.1) and (4.2) ensure that $\Delta_j$ does not change. Hence, the updates are fast: for each $(i, j) \in \mathcal{E}$, over the lifetime of the algorithm we will perform at most one such update over this edge, for a total of $O(|\mathcal{E}|)$ updates. Using appropriate data structures, as we next describe, each update can be performed in $O(\log |\mathcal{V}|)$ time, totalling $O(|\mathcal{E}| \log |\mathcal{V}|)$ time.

**Algorithm 4.1:** FRAUDAR, which greedily removes nodes to maximize a metric $g$. Line 5 and 6 run in $O(\log |\mathcal{V}|)$ time, using a data structure described in Section 4.4.2.1.

**Input** : Bipartite $G = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$; density metric $g$ of the form in (4.1)

1 Construct priority tree $T$ from $\mathcal{U} \cup \mathcal{W}$ ▷See Section 4.4.2.1
2 $\mathcal{X}_0 \leftarrow \mathcal{U} \cup \mathcal{W}$ ▷Suspicious set is initially the entire set of nodes $\mathcal{U} \cup \mathcal{W}$
3 **for** $t = 1$ **to** $(m + n)$ **do**
4  ▷Exonerate least suspicious node
5  $i^* \leftarrow \arg\max_{i \in \mathcal{X}_i} g(\mathcal{X}_i \setminus \{i\})$
6  Update priorities in $T$ for all neighbors of $i^*$
7  $\mathcal{X}_t \leftarrow \mathcal{X}_{t-1} \setminus \{i^*\}$
8 **end**
9 ▷Return most suspicious set $\mathcal{X}_i$
10 **Return** $\arg\max_{\mathcal{X}_i \in \{\mathcal{X}_0, \ldots, \mathcal{X}_{m+n}\}} g(\mathcal{X}_i)$

#### 4.4.2.1 Priority Tree

Each element $i \in \mathcal{X}$ has a priority that will change as the algorithm progresses: the priority of element $i$ at the $t$th iteration is $\Delta_i = f(\mathcal{X}_t \setminus \{i\}) - f(\mathcal{X}_t)$. This ensures that in Line 5, the element $i^* = \arg\max_{i \in \mathcal{X}_t} g(\mathcal{X}_t \setminus \{i\})$ we wish to find is exactly the element of highest priority, allowing us to retrieve it quickly (in $O(\log |\mathcal{V}|)$ time, as we explain below). Note that it does not matter if we use $f$ or $g$ in the $\arg\max$ since the denominator of $g$ in (4.1), $|\mathcal{X}_t \setminus \{i\}|$, is the same for all possible deletions $i$.

These priorities are stored in the priority tree $T$ constructed in line 1 of Algorithm 4.1. This data structure is a binary tree with all $|\mathcal{V}|$ elements as leaves, all at the bottom level of the tree. Each internal node keeps track of the maximum priority of its two children.

The priority tree supports fast retrieval of the maximum priority element (used in Line 5 of Algorithm 4.1); it does this by starting at the root and repeatedly moving to the child with higher priority. It also supports quickly updating priorities: since all the leaves can be stored in fixed locations, we can easily retrieve the leaf at any index to update its priority. Then, after updating that node's priority, we travel up the tree to update each parent up to the root (used in Line 6). Each of these operations on $T$ takes $O(\log |\mathcal{V}|)$ time.

#### 4.4.2.2 Scalability

The bottleneck is the loop in Lines 5 to 7 which runs $m+n$ times. Lines 5 and 6 take $O(\log |\mathcal{V}|)$ as discussed, while Line 7 is constant time. Finally, we need $|\mathcal{E}|$ updates to node priorities, one for each edge. Thus the algorithm takes $O(|\mathcal{E}| \log |\mathcal{V}|)$ time.

### 4.4.3 Theoretical Bounds

So far, we have shown that $g$ can be optimized in near-linear time. In this section, we will show that when $f$ and $g$ are of the form (4.1) and (4.2), FRAUDAR is guaranteed to return a solution of at least half of the optimum value.

**Theorem 4.2**

Let $\mathcal{A}, \mathcal{B}$ be the set of users and objects returned by FRAUDAR. Then:

$$g(\mathcal{A} \cup \mathcal{B}) \geq \frac{1}{2} g_{\text{OPT}}$$

where $g_{\text{OPT}}$ is the maximum value of $g$, i.e.

$$g_{\text{OPT}} = \max_{\mathcal{A}', \mathcal{B}'} g(\mathcal{A}' \cup \mathcal{B}')$$

*Proof.* Let the elements of $\mathcal{V}$ be labeled $v_1, v_2, \ldots, v_{m+n}$. We define 'weight assigned to node $v_i$ in $\mathcal{S}$' as

$$w_i(\mathcal{S}) = a_i + \sum_{(v_j \in \mathcal{S}) \wedge ((v_i, v_j) \in \mathcal{E})} c_{ij} + \sum_{(v_j \in \mathcal{S}) \wedge ((v_j, v_i) \in \mathcal{E})} c_{ji}$$

where $a_i (\geq 0)$ indicates the weight of node $v_i$ and $c_{ij} (> 0)$ indicates that of edge $(v_i, v_j)$ as in (4.2). Note that when node $v_i$ is removed from the current set $\mathcal{S}$ at some point in the algorithm, $w_i(\mathcal{S})$ is the decrease in the value of $f$, since it is the sum of all terms excluded in (4.2) when node $v_i$ is removed.

Now consider the optimal set $\mathcal{S}^*$. For each node $v_i \in \mathcal{S}^*$, we claim that $w_i(\mathcal{S}^*) \geq g(\mathcal{S}^*)$. Otherwise, removing a node with $w_i(\mathcal{S}^*) < g(\mathcal{S}^*)$ results in

$$g' = \frac{f(\mathcal{S}^*) - w_i(\mathcal{S}^*)}{|S^*| - 1} > \frac{f(\mathcal{S}^*) - g(\mathcal{S}^*)}{|S^*| - 1}$$
$$= \frac{f(\mathcal{S}^*) - f(\mathcal{S}^*)/|\mathcal{S}^*|}{|S^*| - 1} = g(\mathcal{S}^*),$$

which is a contradiction.

Let $v_i$ be the node that FRAUDAR removes first among those in $\mathcal{S}^*$, and let $\mathcal{S}'$ be the set before FRAUDAR removes $v_i$. Then, since $\mathcal{S}' \supset \mathcal{S}^*$, $w_i(\mathcal{S}') \geq w_i(\mathcal{S}^*)$. Moreover, since FRAUDAR chooses to remove node $v_i$, for each of the other remaining nodes $v_j \in \mathcal{S}'$, $w_j(\mathcal{S}') \geq w_i(\mathcal{S}')$. Since each term in $f(\mathcal{S}')$ can be assigned to at most two nodes, summing over $j$ gives $f(\mathcal{S}') \geq \frac{|\mathcal{S}'|w_i(\mathcal{S}')}{2}$. Also note that $g(\mathcal{A} \cup \mathcal{B}) \geq g(\mathcal{S}')$ since FRAUDAR returns the best solution that it encounters. We conclude that

$$g(\mathcal{A} \cup \mathcal{B}) \geq g(\mathcal{S}') = \frac{f(\mathcal{S}')}{|\mathcal{S}'|} \geq \frac{w_i(\mathcal{S}')}{2} \geq \frac{w_i(\mathcal{S}^*)}{2} \geq \frac{g(\mathcal{S}^*)}{2}.$$

■

### 4.4.4 Edge Weights and Camouflage Resistance

So far, we have seen that metrics of the form: $g(\mathcal{S}) = \frac{f(\mathcal{S})}{|\mathcal{S}|}$, where $f(\mathcal{S}) = \sum_{i \in \mathcal{S}} a_i + \sum_{i,j \in \mathcal{S} \wedge (i,j) \in \mathcal{E}} c_{ij}$ can be optimized efficiently and with approximation guarantees. In this section, we show how we can select metrics within this class that are resistant to camouflage, i.e. they do not allow fraudulent users to make themselves less suspicious by adding *camouflage edges*, i.e. edges toward honest objects.

Recall that $a_i$ and $c_{ij}$ are the weights of node $i$ and edge $ij$, while $f(\mathcal{S})$ is the total node and edge weight in $\mathcal{S}$. A key idea of our approach is that instead of treating every edge equally, we assign a lower weight $c_{ij}$ when the target object $j$ has high degree. This is because objects of very high degree are not necessarily suspicious (since highly popular objects commonly exist). Thus, this weighting allows us to put greater emphasis on objects within unexpectedly dense subgraphs, rather than just high degree objects.

If we consider the adjacency matrix with rows representing users and columns representing objects, we would like to downweight columns with high column sum (column-weighting). A simple result we show in this section is that column-weightings are camouflage resistant. Recall that a density metric $g$ is camouflage-resistant if $g(\mathcal{A} \cup \mathcal{B})$ does not decrease when any amount of camouflage is added by an adversary with fraudulent users $\mathcal{A}$ and customers $\mathcal{B}$. Let $d_i$ be the the $i$th column sum, i.e. the degree of object $i$.

Formally, define a **column-weighting** as a choice of weighting in which each $c_{ij}$ is a function of the respective column sum, i.e. $c_{ij} = h(d_j)$ for some function $h$.

**Theorem 4.3**

Let $c_{ij}$ be a column-weighting. Then $g$ (as defined in (4.1) and (4.2)) is camouflage resistant.

*Proof.* Adding camouflage only adds edges in the region between $\mathcal{A}$ (fraudulent users) and $\mathcal{B}^C$ (honest objects). It does not add or remove edges within the fraudulent block; moreover, the weights of these edges do not change either as their weights only depend on the column degrees of $\mathcal{B}$, which do not change when camouflage is added. Thus the value of $g$ does not change. ■

A natural follow-up question is whether camouflage-resistance also holds for row-weightings (i.e. selecting $c_{ij}$ to be a function of the corresponding row sum). It turns out that row-weightings are in general not camouflage resistant. This is because a fraudulent user account can add a large number of camouflage edges, thereby increasing their row sum, decreasing the weight of each of their edges. Thus $g(\mathcal{A} \cup \mathcal{B})$ decreases, meaning that $g$ is not camouflage resistant.

Hence we may choose any column-weighting while ensuring camouflage resistance. The remaining question is what function to choose for the column-weighting, i.e. the function $h$ where $c_{ij} = h(d_j)$. It should be decreasing (so as to downweight columns with high sum). It should shrink more slowly than $h(x) = 1/x$, since $h(x) = 1/x$ allows a single edge to contribute as much as the total contribution of a column with any number of edges, causing us to catch columns with single ones rather than dense blocks.

Within the remaining space of choices, we note that a very similar problem of downweighting based on column frequency appears in deciding the form of the 'inverse document frequency' term of the popular heuristic *tf-idf* weighting scheme [RUUU12], in which logarithmic weighting of frequency has been empirically found to perform well. We also show empirical results (in Section 4.5.1) that logarithmic weighting leads to strong theoretical bounds. For these reasons, we recommend using $h(x) = 1/\log(x + c)$, where $c$ is a small constant (set to 5 in our experiments) to prevent the denominator from becoming zero, or excessive variability for small values of $x$. We use the resulting density metric (denoted $g_{\log}$) in our experiments.

### 4.4.5 Implications: Bounding Fraud

Figure 4.1(a) shows curves representing our theoretical bounds on the maximum amount of fraud that can be present for each possible size of the fraudulent block, based on Theorem 4.2. We now explain how such bounds can be computed from Theorem 4.2. Assume that the fraudulent block contains $m_0$ user accounts and $n_0$ customers.

In this section, we assume that no side information is present, so we set the $a_i$, the prior suspiciousness of each node, to 0. Thus here $g_{\log}(\mathcal{S}) = \frac{1}{|\mathcal{S}|}\sum_{i,j\in\mathcal{S}}\frac{1}{\log(d_j+c)}$, where $d_j$ is the degree of the $j$th object. Consider a fraudulent subgraph with $m_0$ user nodes and $n_0$ object nodes. Assume that each fraudulent customer has at least a certain fraction $0 < \lambda < 1$ of fraudulent edges: each fraudulent customer should be receiving at least a comparable fraction of fraudulent reviews to its actual honest reviews, otherwise it would not be profiting appreciably from the fraud.

**Theorem 4.4**

Let $(\hat{\mathcal{A}}, \hat{\mathcal{B}})$ be the block detected by FRAUDAR. Then the number of edges that a fraudulent block of size $(m_0, n_0)$ can have without being detected is at most $2(m_0 + n_0)g_{\log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}}) \log(m_0/\lambda + c)$. In other words, our algorithm will detect a fraudulent block without fail if it contains more edges than this threshold.

*Proof.* By Theorem 4.2, $2g_{\log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}})$ is an upper bound on the value of $g_{\log}$ on any subgraph of users and objects. Since the fraudulent block has $m_0+n_0$ nodes in total, thus $2(m_0+n_0)g_{\log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}})$ is an upper bound on the value of total suspiciousness $f_{\log}$.

Moreover, each fraudulent customer has at most $m_0$ fraudulent edges joined to it, and since at least $\lambda$ fraction of its edges must be fraudulent, it can have at most $m_0/\lambda$ degree in total. Hence the weight of each fraudulent edge is at least $\frac{1}{\log(m_0/\lambda+c)}$. But since the total weighted degree is at most $2(m_0 + n_0)g_{\log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}})$, it follows that the number of fraudulent edges is at most $2(m_0 + n_0)g_{\log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}}) \log(m_0/\lambda + c)$. ■

We apply this bound to real data in Section 4.5.1.

## 4.5 Experiments

We design experiments to answer the following questions:
**Q1. Illustration of our theorem:** How strong are the bounds that FRAUDAR provides in terms of bounding undetectable fraud in the graph? Does column weighting improve those bounds?
**Q2. Evaluation on synthetic data:** How accurately does FRAUDAR detect injected fraud under different types of camouflage attacks? Does FRAUDAR outperform state-of-the-art competitors?
**Q3. Effectiveness in real-world data:** Does FRAUDAR detect true fraud in real-world graphs? Have the fraudulent accounts already been detected by previous methods?
**Q4. Scalability:** Is FRAUDAR scalable with regard to the data size?

We implemented FRAUDAR in Python; all experiments were carried out on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM, running OS X 10.9.5. The code is available for download at `www.andrew.cmu.edu/user/bhooi/code`. We test FRAUDAR on a variety of real world datasets. Table 8.3 offers details on the datasets we used.

To test the accuracy of our method, we use synthetic attacks injected into our Amazon dataset. We structure our "attacks" as shown in Figure 4.2. We injected a fraudulent block of users and customers with varying densities. We assume $\lambda = 0.5$ for our theoretical bounds.

| | Nodes | Edges | Density | Content |
|---|---|---|---|---|
| *Amazon* [ML13] | 28K (24K,4K) | 28K | 2.7e-4 | Review |
| *Trip Advisor* [WLZ11] | 84K (82K,2K) | 90K | 5.9e-4 | Review |
| *Epinion* [LHK10] | 264K (132K,132K) | 841K | 4.8e-5 | Who-trust-whom |
| *Wiki-vote* [LHK10] | 16K (8K,8K) | 103K | 1.5e-3 | Vote |

Table 4.3: Bipartite graph datasets used in our experiments.

### 4.5.1 Q1. Illustration of our Theorem

In Figure 4.1 (a), we showed our theoretical bounds (Theorem 4.4) applied to compute the maximum number of edges an adversary with $m_0 = 50$ user nodes can have for various values of $n_0$. These are computed by running FRAUDAR under two weighting schemes. First, we use our $g_{\log}$ scheme exactly as in Theorem 4.4 to get an upper bound $2(m_0 + n_0)g_{log}(\hat{\mathcal{A}} \cup \hat{\mathcal{B}}) \log(m_0/\lambda + c)$ on the number of fraudulent edges; plotting this against $n_0$ gives the green region ('improved') in Figure 4.1 (a). The blue region ('original') comes from using the analogous procedure without the log-weighting, i.e. where $g(\mathcal{S})$ is half the average degree, as in Example 1.

In this case, we see that the log-weighted scheme provides stronger bounds, since the bound is lower, i.e. an adversary should have fewer edges in order not to be detected. Intuitively, this happens because down-weighting high degree columns decreases the weight of many of the honest high degree objects in the dataset, so groups of adversaries stand out more, resulting in stronger bounds on how many edges an adversary can have.

Next, we apply our FRAUDAR in the same way over various real-world graphs to analyze the theoretical upper bounds computed by FRAUDAR on the density that fraudulent blocks can have. We run FRAUDAR on four real-world graphs: *Amazon* [ML13], *Trip Advisor* [WLZ11], *Epinions* [LHK10], and *Wiki-vote* [LHK10]. The detailed description of each graph is in Table 8.3. For all datasets, Figure 4.3 shows the maximum number of fraudulent edges that an adversary can have without being detected, assuming $50$ fraudulent users and varying the number of fraudulent customers. We see that we can detect fraud most easily in *Trip Advisor*, followed by *Epinion, Wiki-vote, Amazon*; even a fairly sparse block of density around $0.05$ would stand out strongly in the *Trip Advisor* graph. While density is important in determining how easy it is to detect fraud in each graph (fraudulent blocks stand out more strongly in a sparse graph), it is not the only factor. Indeed, *Wiki-vote* is actually denser than *Amazon*. In fact, the difficulty of detecting fraud in each graph is mainly determined by its densest blocks, since an adversarial block that is significantly less dense than the densest normal blocks in the graph is unlikely to be detected.

### 4.5.2 Q2. Evaluation on Synthetic Data

In Figure 4.1 (b), we demonstrated that FRAUDAR can effectively detect fraud under four types of camouflage attacks: 1) Injection of fraud with no camouflage, 2) random camouflage, 3) biased camouflage and 4) hijacked accounts, more accurately than competitors.

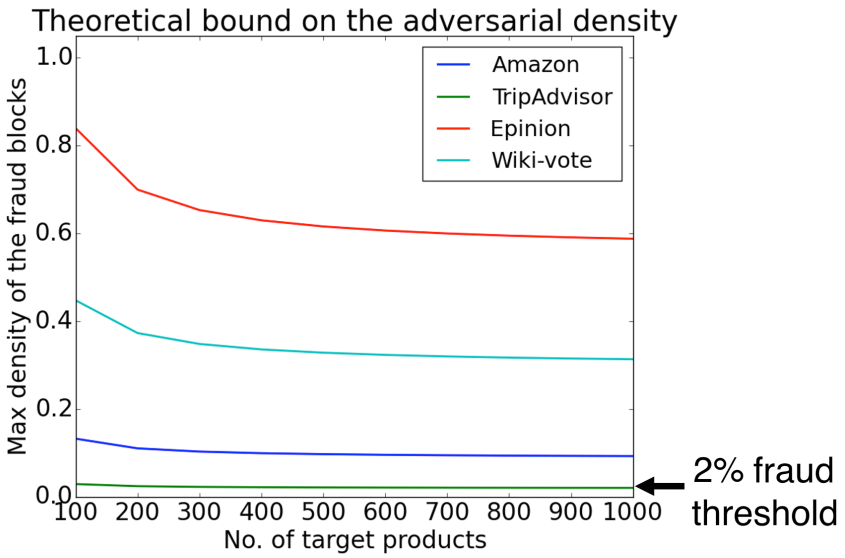



Figure 4.3: **Fraudar's bounds on fraud are stringent, on real graphs:** E.g., on TripAdvisor, the bound says that a fraudulent block containing 50 user accounts and anywhere between 100 and 1000 products must have density of $< 2\%$ to avoid detection.

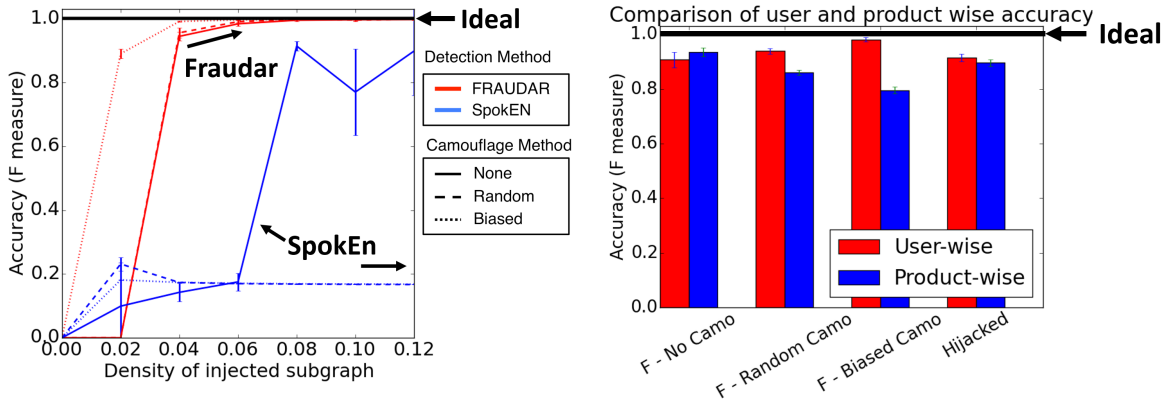



Figure 4.4: **(a) Fraudar outperforms competitors in multiple settings**. Accuracy of fraud detection on Amazon data in the experiment with "reverse camouflage" (edges from honest users to fraudulent products). **(b) Fraudar has similar and high accuracy both in detecting fraudulent users and fraudulent customers**. Comparison of accuracy on fake users and targets under four different camouflage attacks.

We conduct experiments based on the settings at the beginning of this section, averaged over 5 trials. For the camouflage scenarios 2) and 3), the amount of camouflage added per fraudulent user account was (on average) equal to the amount of actual fraudulent edges for that user. For the 'Random Camo' case, for each fake user node, camouflage edges were chosen at random, with on average the same number of camouflage edges as fraudulent edges, as shown in Figure 4.2 (a). For the 'Biased Camo' case, for each fake user node, camouflage edges were directed toward each object with probability proportional to the degree of the object as shown in Figure 4.2 (b). For the 'Hijacked' case, we used a random subset of existing users to form the fraudulent block.

In each case, we injected $200$ fraudulent users and $200$ fraudulent products with various edge densities to the subsetted Amazon review graph of $2000$ users and $2000$ products, with a density of $0.0006$. We compare FRAUDAR to SPOKEN in their F measure ($= \frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$) in detecting the fake users. In the first set of experiments, we assume that no honest user added an edge to the fraudulent target (i.e. object) nodes.

As seen in Figure 4.1(b), the results demonstrate that FRAUDAR works robustly and efficiently against all four attacks, achieving F-measures of over $0.95$ on all four scenarios for densities of at least $0.04$. On the other hand, SPOKEN was able to reach its maximum performance of $0.9$ only when fraud blocks had densities of higher than $0.06$ and under the 'no camouflage' scenario.

The experimental results in Figure 4.1(b) were based on the assumption that no honest user added an edge to the fraudulent target nodes. However, in a real-world environment, some honest users may add edges to the fraudulent target nodes (which we refer to as "reverse camouflage"). To incorporate this, we conducted another experiment using an attack model where we add edges between honest users and the fraudulent target nodes, but with sparser density compared to the fraud blocks. We added random edges to this region, with half the density of the fraud blocks. All other experimental settings were unchanged. The experimental results are shown in Figure 4.4 (a). For FRAUDAR, the results are generally similar. In contrast, SPOKEN shows slightly worse performance under this additional camouflage.

To show that FRAUDAR is effective both at catching fraudulent users accounts as well as fraudulent objects, we next separately evaluate the fraud detection of both fake users and fake targets using F measure. The basic experimental setup is same as before, with the density of the fraudulent blocks now fixed to $0.03$. In Figure 4.4 (b), the bar plots are shown for the comparison. 'User-wise' (red) denotes the F measure of the detecting fake users, and 'target-wise' denotes the F measure of detecting fake target nodes. We see that in general, accuracy is high and fairly similar, but the performance in detecting fake users is slightly higher than that of detecting products.

### 4.5.3 Q3. Effectiveness on Real Data

In this section, we verify that FRAUDAR accurately detects a large block of fraudulent accounts in the Twitter follower-followee graph, as verified by hand labelling and by clear signs of fraud exhibited by a majority of the detected users. Indeed, a majority of the detected accounts had tweets advertising follower-buying services, and the tweets had not been removed or the accounts suspended for the 7 years since the data was collected. Figure 4.1(d) shows a sample fraudster caught by FRAUDAR.

The Twitter graph we use contains 41.7 million users and 1.47 billion follows; it was extracted in July 2009 and first used in [KLPM10]. On this graph, FRAUDAR detected a dense subgraph of size 4031 followers by 4313 followees. This subgraph is extremely dense, with 68% density, which is highly suspicious in itself.

To further investigate this block, we randomly sampled 125 followers and 125 followees in the block detected by FRAUDAR for hand labeling to determine how many of them appear fraudulent. To do this, we labeled which users were fraudulent based on the following characteristics of their profile data, chosen based on established criteria in the literature [SBGF14] summarized below.

- links on profile associated with malware or scams
- clear bot-like behavior (e.g. replying to large numbers of tweets with identical messages)
- account deleted
- account suspended

For comparison, we also construct two control groups of size 100 containing users that were not detected by the algorithm. The first control group contains randomly selected non-detected users. For the second (degree-matched) control group, we constructed it to match the follower count of users in the detected group; we do this by repeatedly selecting a random detected user, then finding another non-detected user who has at most 10% bigger or smaller follower count. During the labelling process, we shuffled the detected users with the control groups randomly and hid group memberships from labellers, labeling users in a "blind" manner.

Additionally, we also check and report how many of these users have Tweets containing the URLs of two known follower-buying services, *TweepMe* and *TweeterGetter*, showing that they had advertised these follower-buying services through tweets.

Note that this entire labelling process used only profile and tweet data and not follower-followee data, whereas our algorithm uses only follower-followee data, so the labelling is a fair estimate of the algorithm's accuracy. We present two pieces of evidence which strongly indicates fraud in the detected group. Firstly, the percentage of users with tweets advertising *TweepMe* or *Tweetergetter* is much higher among the detected users than among both control groups (Figure 4.5): 41% of the detected followers, and 26% for the detected followees. These rise to 62% and 42% respectively as shown in Figure 4.1(c) if we ignore deleted, protected and suspended accounts (for which profile information was unavailable). In the control groups, there were no mentions of *TweepMe* and very few mentions of *TweeterGetter*, as shown in Figure 4.5. Figure 4.5 shows the breakdown of our groups in terms of deleted and suspended users. Given the sparsity of *TweepMe* and *TweeterGetter* in the control groups, we see that the detected users are likely characterized by a large block of users using these and possibly other follower-buying services, resulting in a dense block.

Secondly, we used our hand-labelling using the above criteria to determine how many of each group appear fraudulent. 57% of the detected followers and 40% of the followees were labelled as fraudulent, deleted or suspended accounts, but much fewer in the control groups, with 25% for the degree-matched control group, and 12% for control group with no condition. Thus both these results support the effectiveness of FRAUDAR in detecting fraudulent users in the real-world graphs.

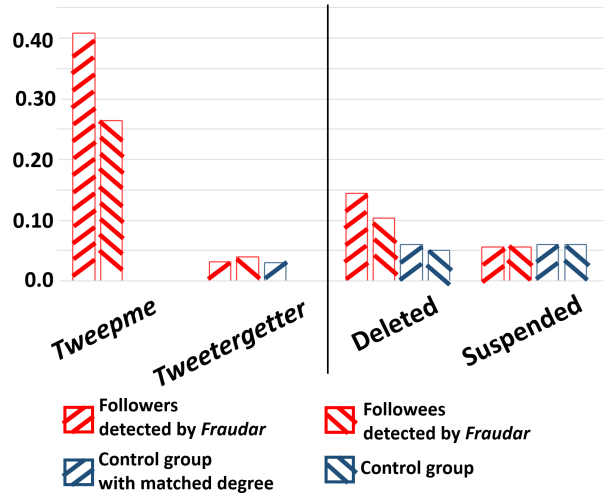



Figure 4.5: **FRAUDAR detects a large, clearly fraudulent block in Twitter.** A majority of the detected accounts were either deleted, suspended, or contained known follower-buying services, *TweepMe* and *TweeterGetter*. In comparison, the control groups had much less detected fraud.

### 4.5.4 Q4. Scalability

Figure 9.6 shows the near-linear scaling of FRAUDAR's running time in the number of edges. Here we used the Trip Advisor dataset, and subsampled user nodes in proportions of $0.7^0, \ldots, 0.7^{12}$. Slopes parallel to the main diagonal indicate linear growth.

## 4.6 Conclusion

In this chapter, we propose FRAUDAR, a fraud detection algorithm which provably bounds the amount of fraud adversaries can have, even in face of camouflage. Our main contributions are as follows.

- **Metric** we propose a novel family of metrics which satisfies intuitive "axioms" and has several advantages as a suspiciousness metric.
- **Theoretical Guarantees** we provide theorems (See Theorem 4.2 in Section 4.4.3 and Theorem 4.4 in Section 4.4.5) on how FRAUDAR gives a provable upper bound on undetectable fraud. We also prove that our proposed metric is camouflage-resistant.
- **Effectiveness** FRAUDAR was successfully applied on real-world graphs on fraud attacks with various types of camouflage, and outperformed the competitor. It also detected a large block of fraudulent activity in the Twitter follower-followee graph.
- **Scalability** FRAUDAR runs near-linearly in the input size. (See Figure 9.6).


Figure 4.6: **Follower-buying services:** a large fraction of detected accounts use *TweepMe* (bottom) or *TweeterGetter* (middle, top).

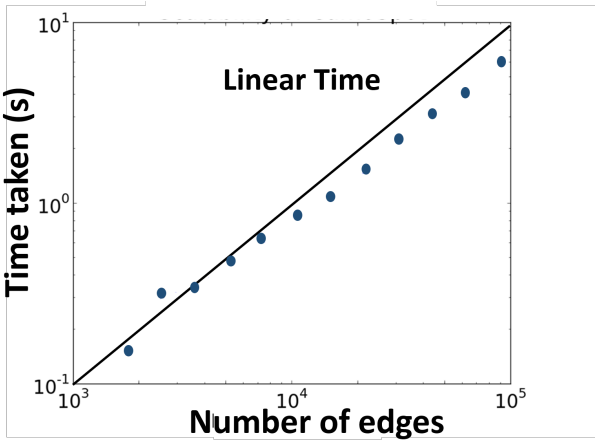



Figure 4.7: **FRAUDAR runs in near-linear time**: the curve *(blue)* shows the running time of FRAUDAR, compared to a linear function *(black)*.

# Part II

# Time Series

# Overview: Time Series

*Given temporal activity, how can we detect unusual time periods?*

In this part, we propose several approaches for anomaly detection and forecasting in time series data of several types. For applications arising from online systems (e.g. online commerce), time series data often involves **discrete events** with associated timestamps, such as ratings, page visits, clicks, likes and so on. Hence, BirdNest focuses on this discrete case, e.g. by detecting bursts of ratings or unusual temporal patterns via a Bayesian approach. More classical time series data is **real-valued** (e.g. sensor data): StreamCast proposes a forecasting and anomaly detection approach for power systems data, that exploits domain knowledge via a physics-based model of power systems. **Mixed data** combines real-valued, categorical, and ordinal data into a multivariate time series: our BNB proposes a nonparametric change detection approach for this case. Finally, interaction data (e.g. user-product) is often modelled using matrices, resulting in a **time-series of matrices**: our SMF algorithm constructs a matrix factorization model capturing drift and seasonality, and proposes an efficient anomaly detection approach.

# Chapter 5

# BirdNest: Fraud Detection in Timestamped Ratings

Chapter based on work that appeared at SDM16 [HSB+16a] [PDF].

In this chapter we consider anomaly detection in discrete time-series data: in particular, detecting fraud in timestamped ratings data. We define a Bayesian model: Bayesian Inference for Rating Data (BIRD). Based on our model we then formulate a likelihood-based anomalousness metric, Normalized Expected Surprise Total (NEST). Experiments on real data show that BIRDNEST successfully spots review fraud in large, real-world graphs: the 50 most suspicious users of the Flipkart platform flagged by our algorithm were investigated and all identified as fraudulent by domain experts at Flipkart.

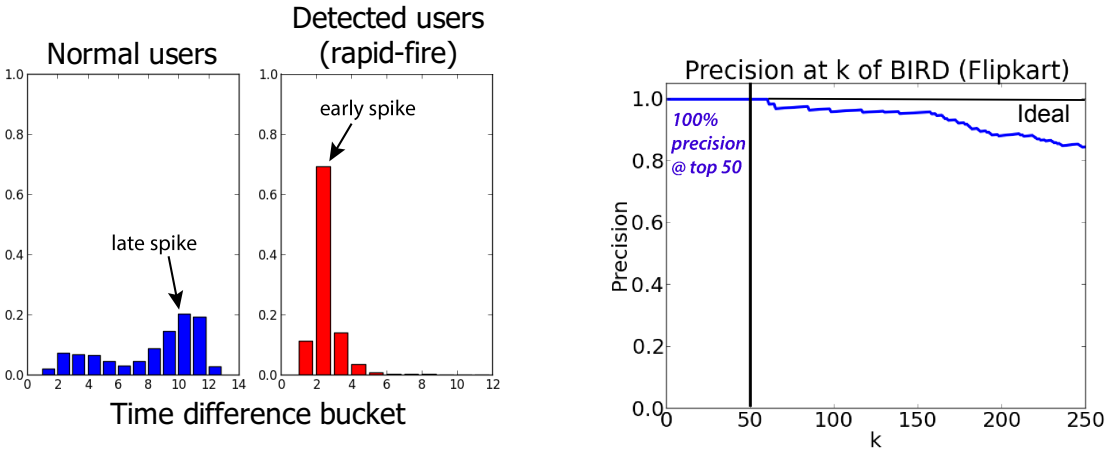
## 5.1 Introduction

Online reviews play an important role in informing customers' purchasing decisions. This has led to the problem of fake reviews, in which businesses write or purchase fake reviews in order to raise the popularity of their products or services. Hence, it is crucial for online commercial platforms to identify and remove these reviews, in order to maintain customers' trust in the accuracy of their reviews.

Various inputs such as rating, review text, timestamp etc. may be available for detection systems; in this work we focus on ratings and timestamps as they are commonly available and informative features. Informally, our problem is:

**Problem 5.1: Suspiciousness Score**

**Given** a set of users and products, and timestamped ratings (e.g. 1 to 5 stars) by users for products, **compute** a suspiciousness score for each user.




(a) **Common pattern observed that detected users' ratings are more 'bursty' than normal users.**. Times between a user's ratings were bucketed logarithmically; detected users have shorter times between ratings.

(b) **BIRDNEST is effective in practice**, with 211 users of the top 250 flagged by BIRDNEST involved in fraud.

Figure 5.1: **BIRDNEST combines temporal and rating information in a principled manner to detect fraud with high precision.** Inspecting the most suspicious 100 users shows their strongly anomalous patterns.

Currently, a number of algorithms use a temporal approach to detect ratings fraud [FCYJMT+15, GGF14, XWLY12]. These focus on catching products that receive a large number of positive or negative reviews in a short time, motivated by the 'bursty' nature of fraudulent reviews when a store wishes to rapidly increase their popularity or defame their competitors. An alternative approach based on rating distributions is to focus on finding users who rate products very differently from other users [LNJ+10, JLL10]. These focus on detection of suspicious behavior by users or products in terms of their deviation from normal practice.

In this chapter, we aim to combine both approaches in a principled way by constructing a Bayesian model for rating behavior, then formulating a likelihood-based metric which measures how much a user deviates from the rest of the users.

The Bayesian approach also provides a principled solution to the conceptually difficult problem of finding a good tradeoff between users with extreme rating distributions vs. users with larger number of ratings. Is a user with 50 ratings (average rating 5.0) more suspicious than a user with 500 ratings (average rating 4.95)? Bayesian methods allow us to quantitatively answer this question. Namely, our Bayesian method combines the rating distribution and number of ratings to estimate our *beliefs* about the rating characteristics of a user in a way that captures our uncertainty, which then determines how suspicious the user is.

Our contributions are:

- **Theoretically sound user behavior model**: we define a Bayesian model for the data based on a mixture model which captures different types of user behavior. This model then allows us to determine how much an anomalous user deviates from normal behavior.

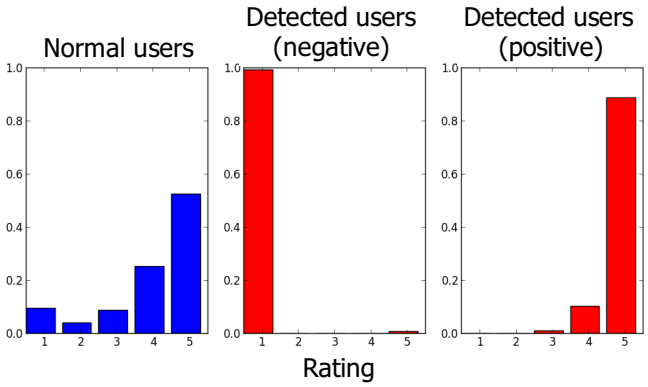



Figure 5.2: **Common pattern observed that detected users' ratings deviate strongly from normal users**: inspecting the detected users shows that they consists of two groups: highly negative users (middle) and highly positive users (right).

- **Suspiciousness metric**: we define a likelihood-based metric which measures how much a user deviates from normal behavior.
- **Algorithm**: we propose a scalable and effective algorithm for learning the Bayesian model and evaluating suspiciousness.
- **Effectiveness**: we show that our method successfully spots review fraud in large, real-world graphs, with precision of over $84\%$ on the top $250$ Flipkart users flagged by our algorithm.

**Reproducibility:** our code is open-sourced at `www.andrew.cmu.edu/user/bhooi/ratings.tar`.

## 5.2 Background and Related Work

### Content-based approaches

A significant portion of opinion fraud comes from customer reviews online. Customer reviews have been long studied [HL04], and many methods for review fraud focus on review text, such as [OCCH11, JL08, FBC12]. While these methods are illuminating, many sites only have ratings without text, or text is easily manipulated. Therefore, in our setting, we focus on ratings and their temporal characteristics, as review text is not always available.

### Graph-based approaches

Much of the existing work in fraud or anomaly detection on graphs has focused on detecting fraud in pure graphs; that is, graphs with no node or edge labels. This includes spectral methods which use eigen-decomposition or singular value decomposition (SVD) to group similar nodes in the graph [PSS+10, JCB+14b, SBGF14]. [WXLY11] uses an iterative approach to label as honest and dishonest. Approaches based on Markov Random Fields and belief propaga-

tion have also been used to identify dense or suspicious subgraphs [PCWF07, ACF13]. [YA15] detects spammers through graph-based measures measuring self-similarity and neighborhood diversity. However, these methods do not make use of key temporal and rating data.

### Temporal methods for fraud detection

There are a number of works on anomaly detection in multivariate time series [LH07, CTPK09, VS10, RRS00]. [BXG+13] focuses on fraudulent temporal patterns in graphs, and [FCYJMT+15] found suspicious inter-arrival times between events in social media. A couple of works address temporal patterns of reviews, e.g. [XWLY12] detects spam singleton reviews and [GGF14] detects time periods of unusual activity. However, our goal is to compute a general, principled, likelihood-based measurement of how suspicious each user is. In this regard, [JBC+15] offers a general suspiciousness metric for count data but is not suitable for ratings data.

### Behavior modeling and fraud detection

A wide body of research has focused on understanding user behavior and especially rating behavior. In particular ratings have been studied by the recommendation systems community, with both frequentist [Kor08] and Bayesian models [SM08] demonstrating great success. Additionally some models have worked to take into account temporal features [Kor10], and others have captured the bimodal patterns in ratings data [BMFS14].

Other behavior models have been proposed to detect users who deviate from normal practice in a meaningful way [LNJ+10, JLL10]. In [SBH+15] a similar problem of finding anomalies in temporal rating data was treated with information theoretic arguments. By taking a Bayesian approach, we develop a significantly different perspective on the problem and our resulting metric of suspiciousness is more flexible, allowing for explicit priors, unique posteriors for each user, and easy extensions to other distributions.

## 5.3 Bayesian Model

### 5.3.1 Motivating Example

We start by illustrating why a Bayesian approach is helpful. Consider users Alice, Bob and Carol whose rating distributions are as given in Figure 5.3. For example, Alice rated $4$ products, all with $5$ stars. Bob did the same, $50$ times. Carol gave about $300$ ratings, and exhibits a 'hockey-stick' distribution, which is close to the average over all users. Which user is the most suspicious (i.e. likely fraudulent)? Our goal is to come up with a principled and intuitive measure of how suspicious each user is.

Why does Alice's low rating count makes her less suspicious than Bob? Our answer is: since we only have $4$ products rated by Alice, we have little information about her true (i.e. long-term) rating behavior. She may simply be a normal user who appears unusual as her first few ratings were high, but given more ratings, she would converge to a more typical distribution. Bob, however, is much less likely to be a normal user: we can say with greater certainty that his true rating behavior is anomalous.

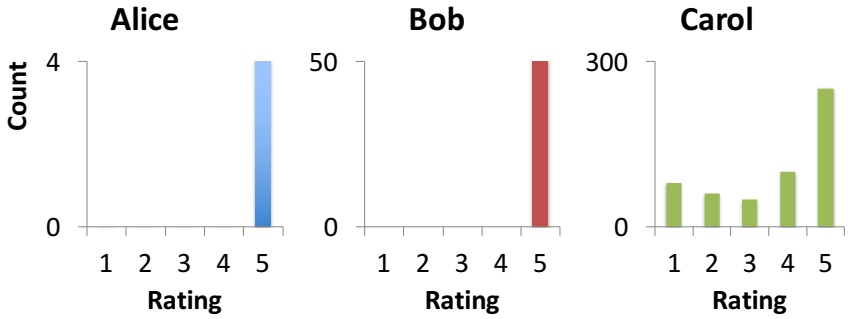



Figure 5.3: Rating distributions of example users. The histograms show how many times each user gave each star rating.

Intuitively, deciding how suspicious each user is involves a two-step process: first, we estimate our beliefs for what that user's true rating distribution is. Second, we estimate how suspicious we believe they are, given our beliefs. For Alice, our beliefs are highly uncertain: we cannot be confident that her rating distribution is unusual. For Bob, we are confident that his rating distribution is fairly skewed toward 5s. For Carol, we know her rating distribution with high confidence, but it is not suspicious.

The Bayesian approach applies this intuition in a principled manner. It first sets a prior, estimated from data, representing our 'default' beliefs about users's rating behavior. It then estimates our beliefs (in the form of a posterior distribution) about their rating distribution. Finally, we compute how suspicious we believe them to be, averaging over their posterior distribution. Figure 5.4 illustrates how posterior distributions capture the information we need to identify a user as suspicious. The posterior distributions in Figure 5.4 refer to our beliefs about each user's true long-term average rating, expressed as a probability distribution. The point estimates refer to each user's observed average rating, which do not capture how much more certain we are in Bob's case than Alice, and hence how much more suspicious Bob is.

**Alternatives that don't work ($z$ or $t$ tests)** What about instead performing a standard hypothesis test (such as a $z$ or $t$-test) for each user's average rating (or any other quantity associated with their rating distribution), to see whether their average rating differs significantly from the population? The problem with this approach lies with users like Carol, who differ slightly from the population but have a large number of ratings. Even as normal (non-fraudulent) users, we expect their true average rating to differ slightly from that of the population (say, by $0.1$) just due to inter-person variation.

Given enough ratings, however, even such a small difference could produce arbitrarily small p-values under such a hypothesis test, since the test correctly concludes that there is an extremely small probability of drawing Carol's observed average rating if her true average rating were equal to that of the population. However, such small differences are not suspicious. The Bayesian approach would instead estimate Carol's posterior distribution as in Figure 5.4 and conclude that it is both narrow and entirely non-suspicious, which is a more sensible result.

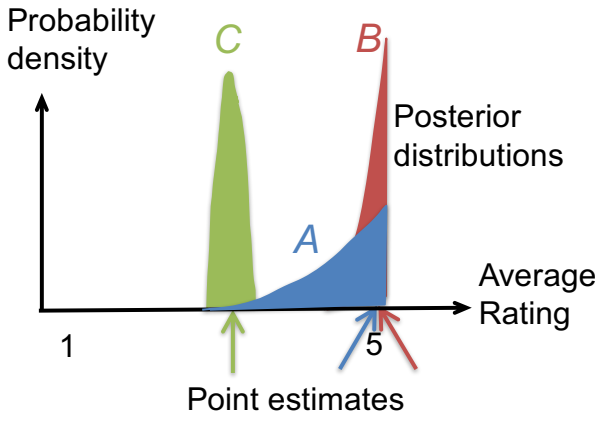



Figure 5.4: **Posterior distributions, not point estimates, mark a user as suspicious.** Bob is suspicious because our beliefs about his true average rating are both narrow and close to 5, while Alice is less suspicious because our beliefs about her true average rating are more spread out.

### 5.3.2 Proposed Model

Table 8.2 summarizes the notation used in this chapter.

In our problem setting, users are indexed $i = 1, \ldots, m$. User $i$ has $n_i$ ratings, indexed by $j = 1, \ldots, n_i$. The ratings in stars given by user $i$ are denoted by the variables $x_{ij} \in \{1, 2, \ldots, s\}$ (e.g. for star ratings from 1 to 5 we have $s = 5$). Similarly to [FCYJMT+15], we preprocess the rating timestamps by computing its time difference from the previous rating, i.e. the difference between its timestamp and the timestamp of the last rating given by the same user. We then bucket the time differences according to the integer part of the log base $b$, where $b$ is chosen to result in close to 20 buckets. The temporal bucket of the $j$th rating of user $i$ is denoted $\Delta_{ij} \in \{1, 2, \ldots, \Delta_{max}\}$ for $j = 1, \ldots, n_i$, analogous to $x_{ij}$.

Using time differences instead of raw timestamps makes it possible to detect either unusually rapid rating of products by a user (due to having a concentration of small time differences), or unusually regular patterns, such as rating products once every hour. Both of these patterns suggest bot-like or spammy behavior, which we would like to detect. Moreover, the discretized i.e. multinomial approach allows us to flexibly detect a wide range of possible deviations from normal behavior without assuming a more restrictive parametric form, such as a Gaussian distribution.

We will consider the ratings $X$ and time differences $\Delta$ to be generated based on a model. From a high level, our generative model for user behavior is a mixture model in which each user belongs to one of $K$ clusters: in general, there is no single type of user behavior, so we use clusters to capture different types of user behavior. Each cluster represents a certain type of rating distribution and temporal distribution for the users in that cluster.

Let $k = 1, \ldots, K$ index into the $K$ clusters. For each user $i$, we first generate which cluster they belong to, $z_i \in \{1, 2, \ldots, K\}$, from a Multinomial($\boldsymbol{\pi}$) distribution, where $\pi_k$, the $k$th entry of $\boldsymbol{\pi}$, is the probability that a random user is generated in cluster $k$.

Table 5.1: Commonly used notation in this chapter. Vectors are in **bold**.

| Parameter | Interpretation |
|---|---|
| $m$ | No. of users |
| $n_i$ | No. of ratings given by user $i$ |
| $s$ | No. of star levels (e.g. $s = 5$ for 1 to 5 stars) |
| $x_{ij}$ | Rating of the $j$th rating given by user $i$ |
| $b$ | base of logarithm for temporal bucketing |
| $\Delta_{ij}$ | temporal bucket of the $j$th rating by user $i$ |
| $\Delta_{max}$ | temporal bucket with highest index |
| $\mathbf{x_i}, \boldsymbol{\Delta_i}$ | Vector $(x_{ij})_{j=1}^{n_i}$ (resp. $\Delta_{ij})_{j=1}^{n_i}$) |
| $X, \Delta$ | Matrix containing all the $(x_{ij})$ (resp. $(\Delta_{ij})$) |
| $n_{il}^x, n_{il}^\Delta$ | No. of times user $i$ gave rating (resp. time) $l$ |
| $\mathbf{n}_i^x, \mathbf{n}_i^\Delta$ | Vector $((n_{i1}^x), \ldots, (n_{is}^x))$ (resp. $((n_{i1}^\Delta), \ldots)$) |
| $K$ | No. of clusters |
| $\pi_k$ | Probability of a random user being in cluster $k$ |
| $z_i$ | Cluster (or mixture component) of user $i$ |
| $\mathbf{p_i}, \mathbf{q_i}$ | User $i$'s rating (resp. temporal) distribution |
| $\boldsymbol{\alpha_k}, \boldsymbol{\beta_k}$ | Dirichlet parameters for cluster $k$ |
| $F_x, F_\Delta$ | Global distributions; refer to (5.5) |

Even within a single cluster, it would not be reasonable to expect all users to behave exactly the same way. Thus, instead of using a single rating/temporal distribution per cluster, we allow small deviations per user. We do this by associating a common Dirichlet prior with each cluster: each user has their individual rating distribution drawn from this prior. We denote user $i$'s rating distribution by $\mathbf{p_i}$, a vector of length $s$ of nonnegative entries which sums to 1, where the $j$th entry of this vector gives their probability of giving the $j$th rating. Thus, we draw user $i$'s rating distribution $\mathbf{p_i}$ from a Dirichlet($\boldsymbol{\alpha}_{z_i}$). Similarly, $\mathbf{q_i}$ represents user $i$'s temporal distribution, and we draw $\mathbf{q_i} \sim$ Dirichlet($\boldsymbol{\beta}_{z_i}$).

Finally, to generate user $i$'s ratings, we draw each rating $x_{ij}$ based on user $i$'s rating distribution: $x_{ij} \sim$ Multinomial($\mathbf{p_i}$). Similarly, for the temporal buckets, we draw each $\Delta_{ij}$ from a Multinomial($\mathbf{q_i}$) distribution.

The generative model we have described is summarized in (5.1).

$$\begin{aligned} z_i &\sim \text{Discrete}(\boldsymbol{\pi}) \\ \mathbf{p_i}|z_i = k &\sim \text{Dirichlet}(\boldsymbol{\alpha}_k) \\ x_{ij} &\sim \text{Multinomial}(n_i, \mathbf{p_i}) \\ \mathbf{q_i}|z_i = k &\sim \text{Dirichlet}(\boldsymbol{\beta}_k) \\ \Delta_{ij} &\sim \text{Multinomial}(n_i, \mathbf{q_i}) \end{aligned} \tag{5.1}$$

The corresponding graphical model is given in Figure 5.5.

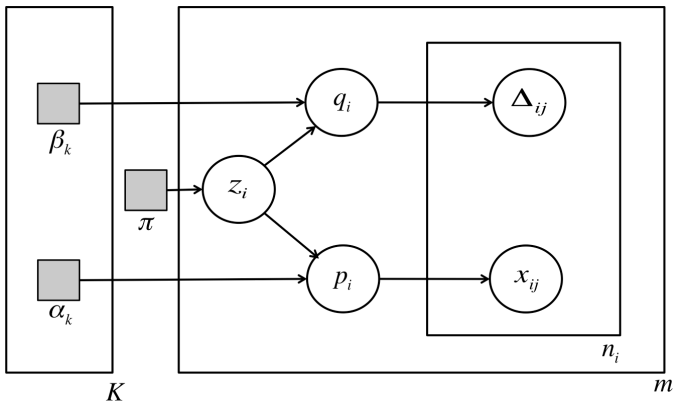



Figure 5.5: Graphical model describing users, ratings and rating times. User $i$'s mixture component $z_i$ determines how we generate their individual multinomial parameter vectors $p_i, q_i$, which then generate $x_{ij}$ and $\Delta_{ij}$ as samples from these multinomial distributions.

## 5.4 Proposed Algorithms

### 5.4.1 Fitting our Bayesian Model (BIRD)

Algorithm 5.1 fits data to the model of Fig.5.5, by using a greedy hill climbing approach to maximize the overall likelihood function. In this algorithm, we iteratively adjust each parameter and the cluster assignments $z$ until convergence. Each of the $\arg\max$ lines in the algorithm can be solved efficiently, which we next describe how to do.

#### Cluster parameters

Here we fix $z$ and compute $\arg\max_{\boldsymbol{\alpha}_k} P(X, \Delta|\boldsymbol{\alpha}_k, z)$ in Line 6; adjusting with respect to $\boldsymbol{\beta}$ will be similar. Note that adjusting $\boldsymbol{\alpha}_k$ only affects the likelihood with respect to $x_i$, for $i$ in cluster $k$. Thus we are equivalently maximizing $\prod_{i:z_i=k} P(x_i|\boldsymbol{\alpha}_k, z)$.

To be clear, here $P(x_i|\boldsymbol{\alpha}_k, z)$ refers to the marginal likelihood, i.e. the probability of generating $x_i$, after marginalizing out $\mathbf{p_i}$. Thus we need to find the maximum likelihood update for $\boldsymbol{\alpha}_k$ given the $x_i$ for $i$ in cluster $k$, which were sampled from the two-step process of first generating $\mathbf{p_i} \sim \text{Dirichlet}(\boldsymbol{\alpha}_k)$ and then generating $x_i \sim \text{Multinomial}(\mathbf{p_i})$. This two-step process is also known as the Dirichlet-multinomial distribution; [Min00] provide fixed-point iteration methods for maximum likelihood estimation of $\boldsymbol{\alpha}_k$ in this setting. Specifically, we repeat until convergence, for each $k = 1, \ldots, K$ and $l = 1, \ldots, s$:

$$\alpha_{kl}^{new} = \alpha_{kl} \frac{\sum_{i=1}^{m} \frac{n_{il}^x}{n_{il}^x - 1 + \alpha_{kl}}}{\sum_{i=1}^{m} \frac{n_i^x}{n_i^x - 1 + \sum_{l'} \alpha_{kl'}}} \tag{5.2}$$

Similarly, the update for $\beta$ is:

$$\beta_{kl}^{new} = \beta_{kl} \frac{\sum_{i=1}^{m} \frac{n_{il}^{\Delta}}{n_{il}^{\Delta} - 1 + \beta_{kl}}}{\sum_{i=1}^{m} \frac{n_{i}^{\Delta}}{n_{i}^{\Delta} - 1 + \sum_{l'} \beta_{kl'}}} \tag{5.3}$$

### Cluster assignments

In Line 11, we fix the cluster parameters and fit the maximum likelihood cluster assignment $z_i$. Note that changing $z_i$ only affects the likelihood with respect to user $i$. Referring to our graphical model in Figure 5.5, maximizing $P(X, \Delta | z_i = k)$ is equivalent to finding:

$$z_i = \arg\max_k \pi_k P(\mathbf{x_i} | z_i = k) P(\boldsymbol{\Delta_i} | z_i = k) \tag{5.4}$$

To compute $P(\mathbf{x_i} | z_i = k)$, note that this is the probability of drawing $\mathbf{x_i}$ from a Dirichlet-multinomial distribution with known parameter $\boldsymbol{\alpha}_k$.

Let $n_{il}^{x} = \sum_{j=1}^{n_i} 1\{x_{ij} = l\}$ be the number of user $i$'s ratings that equal $l$, and similarly $n_{il}^{\Delta} = \sum_{j=1}^{n_i} 1\{\Delta_{ij} = l\}$. The marginal distribution of a Dirichlet-multinomial distribution (after marginalizing out $\mathbf{p_i}$) is known to be

$$P(x_i | z_i = k) = \frac{\Gamma(A_k)}{\Gamma(n_i + A_k)} \prod_{l=1}^{s} \frac{\Gamma(n_{il}^{x} + \alpha_{kl})}{\Gamma(\alpha_{kl})}$$

where $\Gamma$ is the gamma function, and $A_k = \sum_l \alpha_{kl}$. The term $P(\boldsymbol{\Delta_i} | z_i = k)$ can be computed in the same manner. Since $z_i$ is discrete, we can thus maximize (5.4) by computing $\pi_k P(\mathbf{x_i} | z_i = k) P(\boldsymbol{\Delta_i} | z_i = k)$ for each value of $k$ and choosing the maximizing value of $k$.

### Posterior distributions of $p$ and $q$

Here we explain how to compute the posterior distributions in Line 14 of Algorithm 5.1. Let $\mathbf{n}_i^{x} = ((n_{i1}^{x}), \ldots, (n_{is}^{x}))$ and $\mathbf{n}_i^{\Delta} = ((n_{i1}^{\Delta}), \ldots, (n_{is}^{\Delta}))$. At this point the entire iterative process of estimating the hyperparameters and cluster assignments is complete, and we have to compute the posterior distributions of $\mathbf{p_i}$ and $\mathbf{q_i}$ given the data $X$ and $\Delta$. $\mathbf{p_i}$ has a Dirichlet($\boldsymbol{\alpha}_{z_i}$) prior, so by the conjugate prior property of Dirichlet distributions, its posterior distribution is Dirichlet($\alpha_{z_i} + \mathbf{n}_i^{x}$). Similarly, the posterior distribution of $\beta$ is Dirichlet($\beta_{z_i} + \mathbf{n}_i^{\Delta}$).

### Number of clusters

We select the number of clusters $K$ using the Bayesian Information Criterion (BIC).

### Convergence

As we can see from Algorithm 5.1, each adjustment to $\boldsymbol{\pi}, \alpha, \beta$ or $z$ is an $\arg\max$ step and increases the overall likelihood $P(X, \Delta | z; \boldsymbol{\pi}, \alpha, \beta)$. Because the overall likelihood is bounded, this must converge.

**Algorithm 5.1:** Fitting parameters for the model in Figure 5.5. $X$ is a matrix containing all the $x_{ij}$ and $\Delta$ is a matrix containing all the $\Delta_{ij}$.

**Input** : Rating matrix $X$, time-difference matrix $\Delta$
**Output**: Cluster hyperparameters $\boldsymbol{\pi}$, $(\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k)_{k=1}^K$
**Output**: Posterior distributions for each user's rating and temporal distribution $P(\mathbf{p_i}), P(\mathbf{q_i})$

**while** *not converged* **do**
  ▷Adjust cluster proportions $\boldsymbol{\pi}$
  $\pi_k = \sum_{i=1}^m 1\{z_i = k\}/m$
  **for** $k = 1$ **to** $K$ **do**
    ▷Adjust cluster hyperparameters $\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k$
    $\boldsymbol{\alpha}_k^{new} = \arg\max_{\boldsymbol{\alpha}_k} P(X, \Delta | \boldsymbol{\alpha}_k, z)$ (5.2)
    $\boldsymbol{\beta}_k^{new} = \arg\max_{\boldsymbol{\beta}_k} P(X, \Delta | \boldsymbol{\beta}_k, z)$ (5.3)
  **end**
  **for** $i = 1$ **to** $m$ **do**
    ▷Adjust users' assignments to clusters:
    $z_i^{new} = \arg\max_k P(X, \Delta | z_i = k)$ (5.4)
  **end**
**end**
▷Compute user posterior distributions:
$P(\mathbf{p_i}|X, \Delta) = \text{Dirichlet}(\boldsymbol{\alpha}_{z_i} + \mathbf{n}_i^x)$
$P(\mathbf{q_i}|X, \Delta) = \text{Dirichlet}(\boldsymbol{\beta}_{z_i} + \mathbf{n}_i^\Delta)$

### 5.4.2 NEST: Proposed Metric for Detecting Suspicious Users

Algorithm 5.1 gives us the posterior distributions $P(\mathbf{p_i}|x_i, \Delta_i)$ and $P(\mathbf{q_i}|x_i, \Delta_i)$ for the user parameters. In this section, we propose a suspiciousness metric, NEST (Normalized Expected Surprise Total). Recalling Figure 5.4, the overall idea is to compute user $i$'s suspiciousness, averaged over their posterior distribution.

We will compute suspiciousness with respect to rating and temporally, then normalize and combine them to ensure that each has equal influence. This is a practically motivated decision that ensures that even in settings where one of the variables has a much finer resolution than the other (i.e. it is bucketized into more buckets), neither variable will dominate the other in determining suspiciousness.

We now explain how to compute user $i$'s suspiciousness is in terms of their ratings distribution; the same formulas directly apply to the temporal distribution, and we explain how to combine the scores in (5.7).

#### Global Distribution

Recall that our Bayesian model BIRD gives us an estimate for the distribution underlying the rating behavior of all users, in the form of a mixture of Dirichlet($\boldsymbol{\alpha}_k$) distributions with mixture coefficients $\boldsymbol{\pi}_k$ (and similarly, mixture of Dirichlet($\boldsymbol{\beta}_k$) distributions for temporal distributions). Denote this global distribution by $F_x$ (resp. $F_\Delta$):

$F_x$ can be thought of as our estimate for the distribution of $\mathbf{p_i}$ in general over all users ($\mathbf{p_i}$ is the true rating distribution of user $i$).

**Definition 5.1: Global Distribution**

The global distribution $F_x(\mathbf{p})$ is the distribution over users' rating distributions:

$$F_x(\mathbf{p}) = \sum_{k=1}^{K} \boldsymbol{\pi}_k \,\text{Dirichlet}(\mathbf{p}; \boldsymbol{\alpha}_k) \tag{5.5}$$

where Dirichlet($\mathbf{p}; \boldsymbol{\alpha}_k$) refers to the probability of generating $\mathbf{p}$ under a Dirichlet($\boldsymbol{\alpha}_k$) distribution.

#### Surprise

Denote $\tilde{\mathbf{p}}_\mathbf{i} := P(\mathbf{p_i}|x_i, \Delta_i)$, the posterior distribution of $\mathbf{p_i}$ given the data. To be clear, observe that $\tilde{\mathbf{p}}_\mathbf{i}$ is a distribution over multinomial vectors. Recall that we estimate $\tilde{\mathbf{p}}_\mathbf{i}$ as part of BIRD: $\tilde{\mathbf{p}}_\mathbf{i}$ is a Dirichlet($\boldsymbol{\alpha}_{z_i} + \mathbf{n}_i^x$) distribution. $\tilde{\mathbf{p}}_\mathbf{i}$ represents our beliefs about user $i$'s rating distribution.

For the sake of intuition, imagine $\tilde{\mathbf{p}}_\mathbf{i}$ was a point mass, i.e. we had perfect knowledge of user $i$'s rating distribution: assume that it consists of a point mass at $\mathbf{p}$. Recall that the posterior distribution $\tilde{\mathbf{p}}_\mathbf{i}$ is a distribution over multinomial vectors, so $\mathbf{p}$ here is a multinomial vector. Then user $i$'s suspiciousness could be calculated as *surprise* or negative log likelihood under the global distribution $F_x$ evaluated at the rating distribution $\mathbf{p}$:

**Definition 5.2: Surprise**

The surprise of a given rating distribution $\mathbf{p}$ is its negative log likelihood under the global distribution:

$$\text{surprise}(\mathbf{p}) = -\log F_x(\mathbf{p})$$

The less likely $\mathbf{p}$ is, the more suspicious the user is. This makes sense because $F_x$ is our estimate for the global distribution from which all the users are drawn from; the lower the log-likelihood of $\mathbf{p}$, the more anomalous user $i$ is when compared to this distribution. Thus, we use *surprise* to estimate the suspiciousness of a rating distribution $\mathbf{p}$.

### Expected Surprise

Now returning to the general case, when $\tilde{\mathbf{p}}_\mathbf{i}$ is a posterior distribution. In this case, we compute the average over $\tilde{\mathbf{p}}_\mathbf{i}$ of the *surprise* $-\log F_x(\mathbf{p})$: that is, now $\mathbf{p}$ is drawn at random from this posterior distribution $\tilde{\mathbf{p}}_\mathbf{i}$. Averaging the *surprise* gives us the posterior mean (or 'Bayes estimate') of user $i$'s suspiciousness, which can be regarded as our 'best estimate' of user $i$'s suspiciousness given our knowledge of them.[1] Thus, we use *expected surprise* to estimate the suspiciousness of a user based on their posterior distribution $\tilde{\mathbf{p}}_\mathbf{i}$.

**Definition 5.3: Expected Surprise**

The expected surprise for user $i$ measures how surprising user $i$ rating distribution is averaged over its posterior distribution, and is given by:

$$s_x(i) = -\mathbb{E}_{\mathbf{p}\sim\tilde{\mathbf{p}}_\mathbf{i}} \log F_x(\mathbf{p}) \tag{5.6}$$

The expected surprise $s_\Delta(i)$ with respect to the temporal distribution is computed similarly.

As discussed, $-\mathbb{E}_{\mathbf{p}\sim\tilde{\mathbf{p}}_\mathbf{i}} \log F_x(\mathbf{p})$ measures the expected suspiciousness of a sample rating distribution drawn at random from the posterior distribution $\tilde{\mathbf{p}}_\mathbf{i}$, where suspiciousness of a single rating distribution is given by its *surprise* or negative log likelihood under $F_x$.

[1]The posterior mean of a parameter (i.e. the mean of its posterior distribution) is also known as the minimum mean square error estimator, as it minimizes expected least squares loss. It has desirable properties such as consistency under fairly general conditions, and is widely used in practice.[LC98]

### Normalized Expected Surprise Total (NEST)

In our dataset, we use both ratings and temporal data. Using ratings data, we compute posterior distribution $\tilde{\mathbf{p}}_\mathbf{i}$ and the resulting *expected surprise* $s_x(i)$; using temporal data similarly gives us posterior distribution $\tilde{\mathbf{q}}_\mathbf{i}$ and *expected surprise* $s_\Delta(i)$. To combine these, we could simply add them; however, if one had a larger range of possible values than the other, the one with the largest range could end up dominating the sum. To give both terms comparable influence, we normalize them by their respective standard deviations. Let $\sigma_x = \text{std.dev}(s_x(1), \ldots, s_x(m))$ and $\sigma_\Delta$ be defined analogously. Then NEST is defined as:

**Definition 5.4: NEST**

NEST measures how jointly suspicious user $i$ is based on his or her ratings and temporally, and is given by:

$$\text{NEST}(i) = \frac{s_x(i)}{\sigma_x} + \frac{s_\Delta(i)}{\sigma_\Delta} \tag{5.7}$$

Note that there is no need to normalize $s_x$ and $s_\Delta$ additively (e.g. by subtracting the mean scores) since that would simply shift all the scores by the same amount.

### Computing NEST

Fitting BIRD gives us the posterior distribution for user $i$'s rating distribution $\tilde{\mathbf{p}}_\mathbf{i} = P(\mathbf{p_i}|X, \Delta)$ (Line 15 in Algorithm 5.1); we get $\tilde{\mathbf{q}}_\mathbf{i} = P(\mathbf{q_i}|X, \Delta)$ similarly. We also know the full global distribution $F_x(\mathbf{p}) = \sum_{k=1}^{K} \boldsymbol{\pi}_k \, \text{Dirichlet}(\mathbf{p}; \boldsymbol{\alpha}_k)$. Hence, we can compute the expected surprise $s_x(i) = -\mathbb{E}_{\mathbf{p}\sim\tilde{\mathbf{p}}_\mathbf{i}} \log F_x(\mathbf{p})$ by taking a fixed number of samples from $\tilde{\mathbf{p}}_\mathbf{i}$ and repeatedly evaluating their log-likelihood under $F_x$. We can then compute the expected surprise values $s_\Delta(\cdot)$ and combining the two as shown in (5.7).

## 5.5 Experiments

We conducted experiments to answer the following questions: **Q1. Effectiveness on real data**: does BIRDNEST catch fraud on real data? **Q2. Scalability**: does it scale to large datasets? **Q3: Interpretability**: can the results of the Bayesian model BIRD and the scores given by NEST be interpreted in a real-life setting?

We implemented BIRDNEST in Python; all experiments were carried out on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM, running OS X 10.9.5. The code is available for download at `www.andrew.cmu.edu/user/bhooi/code`. We test BIRDNEST on a variety of real world datasets: table 8.3 offers details on the datasets we used.

Table 5.2: Datasets used.

| Dataset | # of users | # of products | # of ratings |
|---|---|---|---|
| Flipkart | 1.1M | 550K | 3.3M |
| SWM[ACF13] | 0.97M | 15K | 1.1M |

### 5.5.1 Q1: Effectiveness

#### Evaluation on Flipkart data

Flipkart is an online e-commerce platform on which merchants sell products to customers, on which customers review products from 1 to 5 stars. We applied BIRDNEST to detect the 250 most suspicious users and provided them to Flipkart; these accounts were investigated and hand-labelled by Flipkart, finding that 211 users of the top 250 flagged by BIRDNEST were involved in fraud. Figure 5.1b shows the algorithm's precision at $k$: for various values of $k$ up to 250: note that precision for the most suspicious users is very high: e.g. precision of 1.0 for the first 50 users. These are substantial findings for Flipkart. One common pattern that the domain-experts found was that most of the users labeled as fraudulent are either spamming 4/5 star ratings to multiple products from a single seller (boosting seller's ratings), or spamming 1/2 star ratings to multiple products from another seller (defaming the competition).

Figure 5.2 plots the averaged rating distributions of users within each group: that is, for each user we computed their frequency of giving each rating from 1 to 5; Figures 5.2 and 5.1a takes the average of the rating distributions for users in the corresponding group. Examining the detected users in Figure 5.2, a common pattern we find is that they consist of extreme polarized rating distributions as well as temporal distributions. The detected users consist of highly negative users (who give only 1 ratings) and highly positive users (who give mostly 5 ratings, with a relatively small fraction of 4s). Similarly, Figure 5.1a shows the common pattern that detected users contain much shorter temporal differences than normal users.

#### Evaluation on SWM data

The SWM datasets consists of software product (app) reviews. The dataset was collected by [ACF13] by crawling all the app reviews in the entertainment category from an anonymous online app store. Each review consists of review text as well as a rating from 1 star to 5 stars.

We find clear evidence of fake reviews in the dataset: for example, the most suspicious user posted a block of 27 reviews for the same app all within the span of less than a week, all five-star ratings with near-identical review title and text, as shown in Table 5.3. Moreover, the review text shows clear signs of being a fake review: they advertise a code associated with an app: typically, users advertise such codes because they give some benefit to the owner when new users download the app via one of these codes.

Aside from this block of reviews, almost all of this user's reviews also consist of similar blocks of repeated text advertising the code. In fact, all 10 of the top suspicious user accounts flagged by BIRDNEST contain advertisements for codes in similar contexts, often accompanied with promises of free cash, points and gift cards.

Table 5.3: **BIRDNEST detects fake reviews in the SWM data:** Example of a 5-star review by the user flagged as most suspicious by BIRDNEST. 27 such near-identical reviews were present for the same app (only trivial differences between them were present, such as the number of dollars signs). All 10 of the top 10 user accounts flagged by BIRDNEST contain similar advertisements for codes.

| ***AWESOMEApp4FreeMoney!!! $$$$$$*** |
| --- |
| *All first time users will need a CODE after downloading this app. So download it now and use my CODE for bonus points. CODE: ...* |

### 5.5.2 Q2: Scalability

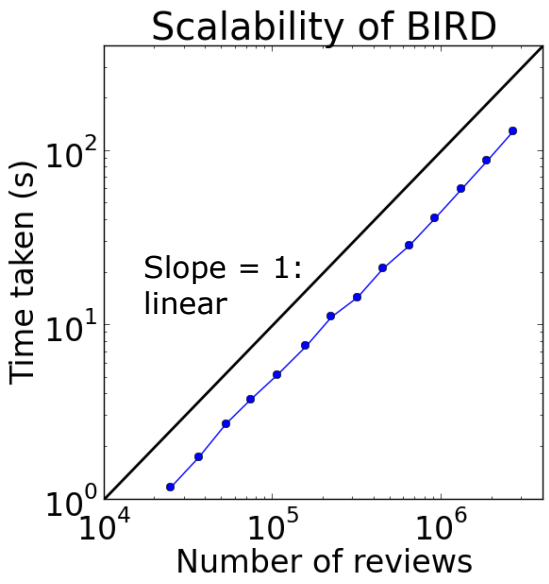



Figure 5.6: **BIRDNEST is fast and scalable:** running on our 1.1M user, 3.3M ratings dataset in around 2 minutes. BIRDNEST shows linear growth in computation time.

Assume that we run $(\#it)$ iterations of the outer loop of Algorithm 5.1, let $m$ be the number of users, and $K$ the number of clusters. In each iteration, adjusting $\boldsymbol{\pi}$ takes $O(m)$ time; adjusting each $\boldsymbol{\alpha}_i$, $\boldsymbol{\beta}_i$ and $z_i$ all take $O(K)$ time. Hence, the algorithm takes $O((\#it)mK)$ time, which is linear. Figure 5.6 shows that the algorithm is fast and its computation time grows linearly in practice.

### 5.5.3 Q3: Interpretability

In Section 5.3.1, we motivated the Bayesian approach by giving three users, Alice, Bob and Carol. We explained how the Bayesian approach captures our intuitions about these users

through posterior distributions. We now use real data from Flipkart to verify that BIRDNEST indeed conforms to the intuitions that motivated this approach, and that the posterior distributions from BIRDNEST are interpretable and useful in real-life settings.

We selected 3 real Flipkart users: `alice`, `bob` and `carol` (names changed to maintain anonymity). They were chosen to match the rating frequencies of Alice, Bob and Carol in 5.3.1. We computed each user's posterior distribution of their true rating distribution using BIRDNEST. From this we computed the posterior distribution of their true, long-term average rating, by simulating $10,000$ draws of $\mathbf{p_i}$ from their Dirichlet posterior distribution. We display these with their NEST values in Figure 5.7.

Agreeing with intuition, both `alice` and `carol` are nonsuspicious, while `bob` is very suspicious, as indicated by the NEST scores: `alice` is ranked around $189,000$th most suspicious, `bob` is ranked around $800$th, and `carol` is ranked around $10,000$th, out of the $1.1$ million users. The NEST scores given by our algorithm are interpretable as *expected surprise* values, i.e. they are in units of log-likelihood, so a unit difference (after normalization) represents an exponential increase in likelihood. As such, we see from this example that BIRDNEST conforms to intuition in the way it uses posterior distributions to measure uncertainty. Moreover, the posterior distributions of average rating or other quantities can be plotted via simulation and used for further understanding and investigation.

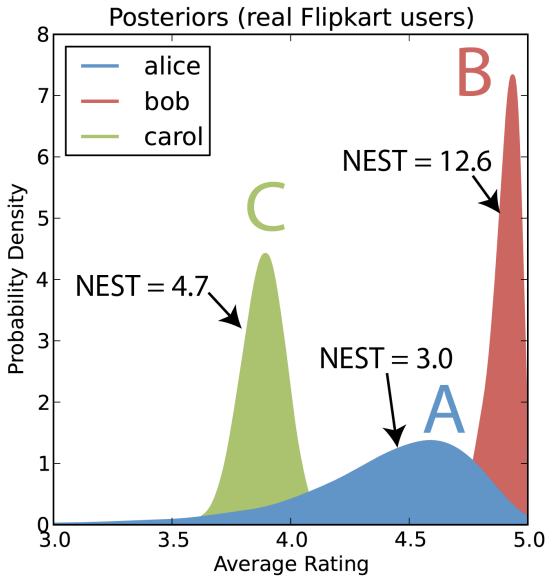



Figure 5.7: **BIRDNEST is interpretable and agrees with intuition:** the posterior distributions for users capture both the location and certainty about a user's true rating distribution. Note that `bob`'s NEST is highest as his entire posterior distribution is extreme (far from other users).

## 5.6 Conclusion

In this chapter, we developed BIRD, a Bayesian inference approach for ratings data, and NEST, a principled likelihood-based suspiciousness metric for fraud detection. Our method provides a principled way to combine rating and temporal information to detect rating fraud, and to find a tradeoff between users with extreme rating distributions vs. users with larger number of ratings. Our contributions are:

- **Theoretically sound user behavior model**: we define a Bayesian model for the data based on a mixture model which captures different types of user behavior. This model then allows us to determine how much an anomalous user deviates from normal behavior.
- **Suspiciousness metric**: we define a likelihood-based metric which measures how much a user deviates from normal behavior.
- **Algorithm**: we propose a scalable and effective algorithm for learning the Bayesian model and evaluating suspiciousness.
- **Effectiveness**: we show that our method successfully spots review fraud in large, real-world graphs, with precision of over $84\%$ for the top $250$ Flipkart users flagged by our algorithm.

# Chapter 6

# StreamCast: Forecasting and Anomaly Detection in Power Grid Time Series

Chapter based on work that appeared at SDM16 [HSB+16a] [PDF].

In this chapter we consider real-valued time series data, in particular from power grid sensors. How can we efficiently forecast the power consumption of a location for the next few days? More challengingly, how can we forecast the power consumption if the temperature increases by $10°C$, the number of appliances in the grid increase by $20\%$, and voltage levels increase by $5\%$? Such 'what-if scenarios' are crucial for future planning, to ensure that the grid remains reliable even under extreme conditions. We propose StreamCast, an online forecasting and anomaly detection approach which scales linearly, provides confidence intervals, and has $27\%$ lower forecasting error than baselines in experiments.

## 6.1 Introduction

The smart electrical grid is a set of technologies designed to improve the efficiency and security of power delivery. Estimates [Ami11] suggest that reducing outages in the U.S. grid could save $\$49$ billion per year, reduce emissions by 12 to $18\%$, while improving efficiency could save an additional $\$20.4$ billion per year. A key part of achieving this goal is to use monitoring data to accurately model the behavior of the grid and forecast future load requirements, especially under extreme conditions such as changes in temperature, number of appliances in the grid, or voltage patterns. This allows the grid to remain reliable even under adverse conditions.

A major challenge is scalability - power systems data can be both high-volume and received in real time. This motivates us to develop fast methods that work in this online (or streaming) setting. Rather than operating on an entire dataset at once, online algorithms allow input that arrives over time as a continuous stream of data points. When each new data point is re-

ceived, the algorithm updates itself - for our algorithm, each update requires constant time, and bounded memory (i.e. it does not need to remember the entire history of the time series).

Hence, our goal is an online algorithm for modelling and forecasting power consumption of a location. The input is a stream over time of real and imaginary voltage and current values:

**Informal Problem 6.1: Online Model Estimation**

- **Given:** A continuous stream of values of real and imaginary current $(I_r(t), I_i(t))$, voltage $(V_r(t), V_i(t))$, and temperature $T(t)$, for $t = 1, 2, \cdots$
- **Estimate:** Time-varying parameters of a physics-based model of electrical load behavior that accurately explains the observed values.

Our model is based on the circuit theoretic BIG model [JPS+17], which characterizes load in the electric grid by modeling its voltage sensitivities. Importantly, the use of physics-based models together with current and voltage state variables improves interpretability. For instance, real power consuming loads such as light bulbs can be interpreted as the contribution of the conductance $(G)$ parameter, while susceptance $(B)$, the reactive power component, represents the contributions of motors or capacitors in the grid.

The fitted model is used to forecast future values:

**Informal Problem 6.2: Multi-step Forecasting**

- **Given:** values of current $(I_r(t), I_i(t))$, voltage $(V_r(t), V_i(t))$, and temperature $T(t)$ for $t = 1, \cdots, N$, and given temperature forecasts for the next $N_f$ time steps (i.e. for $t = N + 1, \cdots, N + N_f$),
- **Forecast:** voltage and current for $N_f$ time steps in the future; i.e. $(V_r(t), V_i(t))$ and $(I_r(t), I_i(t))$ for $t = N + 1, \cdots, N + N_f$.

Due to our physics-based model, our algorithm can handle **what-if scenarios** in which the temperatures or voltages change, which is useful for future planning.

**Informal Problem 6.3: What-if Scenarios**

- **Given:** current, voltage and temperature data, as above,
- **Forecast:** future values of voltage and current, under the condition that, e.g., temperature increases by $10°C$, and voltage levels increase by $5\%$.

Our contributions are as follows:

1. **Domain knowledge infusion:** we propose a novel, *Temporal BIG* model that extends the physics-based BIG model, allowing it to capture trends, seasonality, and temperature effects.
2. **Forecasting:** our StreamCast algorithm forecasts multiple steps ahead and outperforms baselines in accuracy by 27% or more. StreamCast is online, requiring linear time and bounded memory.
3. **What-if scenarios and anomaly detection:** our approach accurately handles scenarios in which the voltage levels, temperature, or number of appliances change. We also use it to detect anomalies in a real dataset.

**Reproducibility:** our code is publicly available at `www.andrew.cmu.edu/user/bhooi/power.tar`.

## 6.2 Background and Related Work

### 6.2.1 Related Work

#### 6.2.1.1 The BIG model for electrical load

The constant power *PQ* model [PJL+16] is a common approach for power grid modelling. However, industry experience has shown that it incorrectly characterizes load behavior [MAB13]. Recent advances [BJL+15] have shown that load behavior at a given time can be accurately described by a linear relationship between current and voltage. From circuit theory, this can be represented by a parallel or series combination of susceptance ($B$) and conductance ($G$). This captures both magnitude and angle, in contrast to existing traditional load models [PJL+16].

#### Time series forecasting

Classical time-series forecasting methods include autoregression (AR)-based methods, including ARMA, ARIMA [BJRL15], seasonal ARIMA [BJRL15], and vector autoregression (VAR) [Ham94]. Exponential smoothing (ETS) models [Win60], including Holt-Winters [Win60] capture trends and seasonal patterns. Other methods include Kalman filtering [K+60], Hidden Markov Models (HMMs) [LRBP09], and non-linear dynamical systems [MS16].

For power grid load modelling, common approaches include AR models [PPL91, HS03], ETS [JXWC12], and neural networks [HPS01]. [SCM+14, ZWWB15] use weather data as inputs. PowerCast [SHJ+17] uses tensor decomposition to forecast power grid time sequences.

#### Contrast with existing literature

Other than [SHJ+17], all methods above do not use physics-based electrical models, and do not consider what-if scenarios, while our approach does both. Compared to [SHJ+17], our approach (which is completely different from their tensor-based approach) additionally allows for weather data, is an online algorithm, and produces confidence intervals.

Table 6.1: **StreamCast captures the listed properties**. AR++ refers to ARIMA, seasonal ARIMA etc.

| *Property* | AR++[BJRL15] | Kalman/LDS[K+60] | Weather-based [SCM+14] | HMM++[LRBP09] | PowerCast [SHJ+17] | StreamCast |
|---|---|---|---|---|---|---|
| **Forecasting** | ✓ | ✓ | ✓ | ✓ | ✓ | ✔ |
| **Seasonal patterns** | ✓ | ? | ✓ | | ✓ | ✔ |
| **Physics-based model** | | | | | ✓ | ✔ |
| **Weather-based** | | | ✓ | | | ✔ |
| **Online algorithm** | | ✓ | | | | ✔ |
| **Confidence intervals** | ✓ | ✓ | | ✓ | | ✔ |
| **What-if scenarios** | | | | | ✓ | ✔ |

### 6.2.2 Background

#### BIG model

The BIG model [JPS+17] models the current as a linear function of the voltage, parameterized by the BIG parameters: susceptance ($B$), conductance ($G$), and a current offset ($\alpha_r, \alpha_i$). $G$ can be interpreted as the component contributing to real power consuming loads (e.g. due to light-bulbs), while $B$ can be interpreted as the contribution of the reactive power component (e.g. due to motors or capacitors):

$$\begin{aligned} I_r(t) &= G \cdot V_r(t) - B \cdot V_i(t) + \alpha_r + \text{noise} \\ I_i(t) &= B \cdot V_r(t) + G \cdot V_i(t) + \alpha_i + \text{noise} \end{aligned} \tag{6.1}$$

#### 6.2.2.1 Holt-Winters model

The Holt-Winters model [Win60] models a univariate time series $x(t)$ with seasonal structure. The key idea is to model $x(t)$ as the sum of a nonseasonal or *level* component $l(t)$ and a *seasonal* component $s(t)$. The level component changes according to smooth trends, while the seasonal component is approximately periodic with period $m$ (e.g. $m = 24$ for hourly data with daily seasonality). Smooth trends over time are modelled by a linear trend $b(t)$, representing the rate of change of $l(t)$. Full details can be found in [Win60].

## 6.3 Proposed Model

Table 10.2 shows the symbols used in this chapter.

Table 6.2: Symbols and definitions

| Symbol | Definition |
|---|---|
| $N$ | Number of time ticks in time sequences |
| $I_r, I_i, V_r, V_i$ | Real and imaginary current and voltage |
| $B, G, \alpha_r, \alpha_i$ | BIG parameters (susceptance, conductance, offset to $I_r$ and $I_i$) |
| $\theta(t)$ | Parameter vector $(B, G, \alpha_r, \alpha_i)$ at time $t$ |
| $y(t), X(t)$ | BIG in linear model form; see Eq. (6.3) |
| $\theta_L(t)$ | Nonseasonal part of $\theta(t)$ |
| $\theta_S(t)$ | Seasonal part of $\theta(t)$ |
| $\theta_T(t)$ | Trend in $\theta_L(t)$ |
| $\theta_W(t)$ | Weather part of $\theta(t)$ |
| $w$ | Weather coefficients |
| $T_0$ | Temperature threshold |
| $N_{\text{init}}$ | Initialization period |

### 6.3.1 Proposed Dynamic BIG Model

Consider the static BIG model in Eq. (6.1). Its parameters $B, G, \alpha_r, \alpha_r$ can be interpreted as types of load; but in practice, these should change over time as appliances are switched on and off, or usage levels change. How do we add temporal structure to this model? A natural step is to replace $B, G, \alpha_r, \alpha_r$ by time series $B(t), G(t), \alpha_r(t), \alpha_r(t)$. Eq. (6.1) becomes:

$$\begin{aligned} I_r(t) &= G(t) \cdot V_r(t) - B(t) \cdot V_i(t) + \alpha_r(t) + \text{noise} \\ I_i(t) &= B(t) \cdot V_r(t) + G(t) \cdot V_i(t) + \alpha_i(t) + \text{noise} \end{aligned} \tag{6.2}$$

For notational simplicity, rewrite this equivalently as:

$$\underbrace{\begin{pmatrix} I_r(t) \\ I_i(t) \end{pmatrix}}_{y(t)} = \underbrace{\begin{pmatrix} V_r(t) & -V_i(t) & 1 & 0 \\ V_i(t) & V_r(t) & 0 & 1 \end{pmatrix}}_{X(t)} \underbrace{\begin{pmatrix} G(t) \\ B(t) \\ \alpha_r(t) \\ \alpha_i(t) \end{pmatrix}}_{\theta(t)} + \text{noise} \tag{6.3}$$

Now, changes in usage (e.g. appliances switching on and off) correspond to changes in $\theta(t)$. What temporal patterns do we need to capture? Intuitively, some appliances follow daily seasonality (e.g. lights used during working hours), while others follow slow-moving trends, e.g. a gradual increase in load due to population growth. Hence, like in Holt-Winters, we decompose

$\theta(t)$ into a nonseasonal **level** part $\theta_L(t)$, and a **seasonal** part $\theta_S(t)$; hence, $\theta(t) = \theta_L(t) + \theta_S(t)$. Here $\theta_L(t)$ changes according to smooth trends, while $\theta_S(t)$ has seasonal patterns, with period $m$.

To model smooth trends (e.g. population growth), we additionally define a **trend** term $\theta_T(t)$, which approximates the change in $\theta_L(t)$ from one time tick to the next; i.e. $\theta_L(t) \approx \theta_L(t-1) + \theta_T(t)$. Then, our assumptions under this model are that:

A1: $y(t) \approx X(t)(\theta_L(t) + \theta_S(t))$ (Low noise; see (6.3))

A2: $\theta_L(t) \approx \theta_L(t-1) + \theta_T(t)$ (Level moves wrt. trend)

A3: $\theta_T(t) \approx \theta_T(t-1)$ (Trends change smoothly)

A4: $\theta_S(t) \approx \theta_S(t-m)$ (Seasonality)

We will formalize these as soft constraints in our optimization objective in Section 6.4.

### 6.3.2 Dynamic BIG with Temperature Model

Let $T(t)$ denote the temperature at time $t$. When temperature increases above a threshold, electricity demand tends to increase due to the use of air conditioning. To capture this, we introduce **weather coefficients** $w$ and a **temperature threshold** $T_0$: temperature over the threshold linearly adds to a weather component $\theta_W(t)$. Hence, we replace assumption A1 with:

A1': $y(t) \approx X(t)(\theta_L(t) + \theta_S(t) + \theta_W(t))$, where
$\theta_W(t) = w \cdot \max(0, T(t) - T_0)$

The $\max$ function ensures that only temperatures above the threshold contribute to the weather component.

## 6.4 Proposed Optimization Objective

We now define our optimization objective based on our assumptions A1' to A4. All norms are L2 norms, for later computational simplicity.

$$
\begin{aligned}
\mathcal{L} &= \mathcal{L}_1 + \lambda L_2 + \mu \mathcal{L}_3 + \nu \mathcal{L}_4, \text{ where:} \\
\mathcal{L}_1 &= \sum_{t=1}^{N} \|y(t) - X(t)(\theta_L(t) + \theta_S(t) + \theta_W(t))\|^2 \\
\mathcal{L}_2 &= \sum_{t=2}^{N} \|\theta_L(t) - \theta_L(t-1) - \theta_T(t)\|^2 \\
\mathcal{L}_3 &= \sum_{t=1}^{N} \|\theta_T(t) - \theta_T(t-1)\|^2 \\
\mathcal{L}_4 &= \sum_{t=m+1}^{N} \|\theta_S(t) - \theta_S(t-m)\|^2
\end{aligned}
\tag{6.4}
$$

Here $\lambda, \mu, \nu > 0$ are hyperparameters (which we end up tuning indirectly rather than directly; see Section 7.4). The problem to solve is then:

$$\underset{\theta_L, \theta_T, \theta_S, w, T_0}{\text{minimize}} \ \mathcal{L} \tag{6.5}$$

## 6.5 Proposed StreamCast Algorithm

How do we approximately minimize $\mathcal{L}$ in an efficient and online manner?

### 6.5.1 Overview

In this section, we outline our proposed StreamCast. StreamCast has two steps: 1) an offline step TempFit, which takes a subset of $N_{\text{init}}$ data points, and fits $w$ and $T_0$; 2) an online 'extension' stage StreamFit: upon receiving each new data point at time $t$, this updates $\theta_L, \theta_T, \theta_S$ according to Eq. (6.7).

**Algorithm 6.1:** StreamCast

**Input** : Streams $V_r, V_i, I_r, I_i$
**Output**: Parameter streams $\theta_L, \theta_T, \theta_S$

$w, T_0 \leftarrow$ TempFit$(V_r(1\colon N_{\text{init}}), V_i(1\colon N_{\text{init}}), I_r(1\colon N_{\text{init}}), I_i(1\colon N_{\text{init}}))$
**while** *input at time $t$ is received:* **do**
  Update $\theta_L(t), \theta_T(t), \theta_S(t)$ using StreamFit
**end**

We will describe StreamFit in Section 6.5.2, and TempFit in Section 6.5.3.

### 6.5.2 Streaming Optimization (StreamFit)

StreamFit estimates $\theta_L, \theta_T$ and $\theta_S$ for fixed temperature parameters $w, T_0$, by minimizing our objective, Eq. (9.6). Note that Eq. (9.6) could in theory be minimized using least squares; however, this is not online, and is also much too slow as it contains $O(N)$ unknowns for $N$ time points, which would then take around $O(N^3)$ time. We would like a linear-time algorithm with constant update time per data point.

Our approach is to split up the objective over $t$, and take gradient update steps with respect to the term for $t = 1, 2, \cdots$ successively. At time $t$, we take a gradient step with respect to only the terms of $\mathcal{L}$ corresponding to time $t$. This allows the fitted parameters to 'track' the true values over time as we perform gradient updates. Meanwhile, each update is highly efficient as it only involves a single term of the objective function. Assume that we have fit $\theta_L, \theta_T, \theta_S$ up to time $t-1$ and are fitting them at time $t$. The component of $\mathcal{L}$ for time $t$ is:

$$\begin{aligned}\mathcal{L}(t) = &\|y(t) - X(t)(\theta_L(t) + \theta_S(t) + \theta_W(t))\|^2 \\ &+ \lambda\|\theta_L(t) - \theta_L(t-1) - \theta_T(t)\|^2 \\ &+ \mu\|\theta_T(t) - \theta_T(t-1)\|^2 + \nu\|\theta_S(t) - \theta_S(t-m)\|^2\end{aligned} \tag{6.6}$$

To do gradient descent at time $t$, we start from the previously learned values and take a gradient step. Hence, we start by assuming the previous trend ($\theta_T(t) = \theta_T(t-1)$) and seasonality ($\theta_S(t) = \theta_S(t-m)$), while for the level we follow a 'default extrapolation' of starting at the previous level and following the previous trend ($\theta_L(t) = \theta_L(t-1) + \theta_T(t-1)$).

Next, we want to move in the gradient direction with respect to minimizing $\mathcal{L}$. Computing the gradients of (6.6) is straightforward and results in the update equations: letting $\hat{y}(t) = X(t)(\theta_L(t-1) + \theta_T(t-1) + \theta_S(t-m) + \theta_W(t))$,

$$\begin{aligned}
\theta_L(t) &= \underbrace{\theta_L(t-1) + \theta_T(t-1)}_{\text{default extrapolation}} + \alpha \underbrace{X(t)^T(y(t) - \hat{y}(t))}_{\text{gradient update}} \\
\theta_T(t) &= \underbrace{\theta_T(t-1)}_{\text{previous trend}} + \beta \underbrace{(\theta_L(t) - \theta_L(t-1) - \theta_T(t-1))}_{\text{gradient update}} \\
\theta_S(t) &= \underbrace{\theta_S(t-m)}_{\text{previous seasonality}} + \gamma \underbrace{X(t)^T(y(t) - \hat{y}(t))}_{\text{gradient update}}
\end{aligned} \tag{6.7}$$

$\alpha, \beta, \gamma > 0$ are 'learning-rate' tuning parameters that replace $\lambda, \mu, \nu$. We will consider how to set these, and how to initialize $\theta_L, \theta_T$ and $\theta_S$, in Section 6.5.3.

### 6.5.3 Temperature Model Optimization (TempFit)

In TempFit, we optimize the temperature coefficient $w$ and threshold $T_0$ using an alternating approach. First, fixing $w$ and $T_0$, we solve for $\theta_L, \theta_T$ and $\theta_S$ to minimize $\mathcal{L}$. We then do the reverse (fixing $\theta_L, \theta_T$ and $\theta_S$ and solving for $w$ and $T_0$), until convergence, as shown in Algorithm 6.2. The former step of solving for $\theta_L, \theta_T$ and $\theta_S$ given $w$ and $T_0$ is done using StreamFit.

**Algorithm 6.2:** TempFit

**Input** : $V_r, V_i, I_r, I_i$
**Output**: BIG parameters $\theta_L, \theta_T, \theta_S$; weather parameters $w$, $T_0$

```
while not converged do
    Solve θ_L, θ_T, θ_S using Eq. (6.7) (StreamFit)
    Solve w, T_0 by minimizing Eq. (6.8)
end
```

It remains to solve for $w$ and $T_0$ for fixed $\theta_L, \theta_T, \theta_S$. Define $r(t) = y(t) - X(t)(\theta_L(t) + \theta_S(t))$. Then, since changing $w$ and $T_0$ only affects the $\mathcal{L}_1$ loss term, minimizing $\mathcal{L}$ is equivalent to minimizing:

$$\mathcal{L}' = \sum_{t=1}^{N} \|r(t) - X(t) \cdot w \cdot \max(0, T(t) - T_0)\|^2 \tag{6.8}$$

Minimizing over $w$ is a straightforward least squares problem. Minimizing over $T_0$ is less straightforward since $\max(0, T(t) - T_0)$ is nonlinear in $T_0$. However, temperature is typically given to 0 or 1 decimal of precision (roughly the precision of most thermometers), and varies

within a fairly narrow range, e.g. $0°C$ to $35°C$. Hence, it suffices to try all thresholds between $\min_t T(t)$ and $\max_t T(t)$ in intervals of $0.1°C$, and select among these to minimize $\mathcal{L}'$.

To complete our algorithm, we need to explain how to choose $\alpha, \beta, \gamma$, and initial values for $\theta_L(t), \theta_T(t), \theta_S(t)$. We select the former using nonlinear optimization (Levenberg-Marquadt algorithm [Mar63]) of $\mathcal{L}$. For the latter, it can be verified that the objective $\mathcal{L}$ is a quadratic in these initial values, so we solve for them using least squares. We do these in Line 2 of TempFit, then fix these values subsequently for the rest of the algorithm (Line 2-4 of StreamCast).

### 6.5.4 Forecasting Step (Forecast)

Given fitted model parameters up to time $N$, how do we forecast future currents $\hat{I}_r(t), \hat{I}_i(t)$, for $t = N+1, \cdots, N+N_f$?

We first forecast voltages $\hat{V}_r(t), \hat{V}_i(t)$, $t = N+1, \cdots, N+N_f$ using standard univariate Holt-Winters. Then, for $k = 1, 2, \cdots$ we forecast $\theta(N+k)$ as the sum of last estimated level $\theta_L(N)$, trend $\theta_T(N)$, last estimated seasonality at the corresponding position $\theta_S(N - m + 1 + ((k-1) \mod m))$, where $\mod$ is the modulo function, and temperature component $w \cdot \max(0, T(N+k) - T_0)$. Finally, we forecast $\hat{I}_r(t), \hat{I}_i(t)$ using the BIG model, Eq. (6.2).

### 6.5.5 Extensions

#### Anomaly Detection

To detect anomalies, we compute an anomaly score at each time. Intuitively, the larger the error between the fitted and actual values, the more anomalous a point is. The fitted values $\hat{I}_r(t), \hat{I}_i(t)$ are obtained by plugging the learned parameters into the BIG equations, Eq. (6.2). The real and imaginary errors are then $E_r(t) = |I_r(t) - \hat{I}_r(t)|$, and $E_i(t) = |I_i(t) - \hat{I}_i(t)|$. The anomalousness at time $t$ is then the sum of real and imaginary errors at time $t$, each in units of inter-quartile ranges (IQRs[1]):

**Definition 6.1**

The anomalousness at time $t$ is the sum of real and imaginary residuals at time $t$, each in units of IQRs:

$$\text{anomalousness}(t) = \frac{E_r(t)}{\text{IQR}(E_r(1:N))} + \frac{E_i(t)}{\text{IQR}(E_i(1:N))} \tag{6.9}$$

While any threshold may then be used, we can follow common practice of designating deviation of $\geq 2 \times IQR$ as outliers, resulting in a threshold of $4$ for our anomalousness score. In Section 8.5.4 we show a clear anomaly found in the LBNL dataset.

[1]The IQR is the difference between 75% and 25% quartiles, used as a more robust measure of spread compared to standard deviation.[UC96]

#### Confidence Intervals

Confidence intervals allow us to provide lower and upper bounds that contain future values, with e.g. $95\%$ confidence. We use the past distribution of residuals, $I_r(t) - \hat{I}_r(t)$ and $I_i(t) - \hat{I}_i(t)$, as an estimate of residuals in the future. Thus, we sample a 'possible future' by repeatedly sampling from this distribution of past residuals, and treating the sampled values as residuals at time $N+1$, repeating this process for $N+2$, and so on. We sample $1000$ possible futures in this way. To generate $(1-\alpha)$ confidence intervals, we use the empirical $\alpha/2$ and $(1-\alpha/2)$-quantiles of these possible futures. The results are shown in Figure 6.6.

## 6.6 Experiments

We design experiments to answer the questions:

- **Q1. Forecasting accuracy:** how accurately does StreamCast forecast, based on real data?
- **Q2. Scalability:** how does the algorithm scale with the data size?
- **Q3. What-if scenarios:** does StreamCast give accurate results under what-if scenarios, and detect real anomalies in real data?

Our code and links to datasets are publicly available at `www.andrew.cmu.edu/user/bhooi/power.tar`. Experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM running OS X 10.11.2. We set $N_{\text{init}}$ to $10m$ (i.e. 10 days) in our experiments.

### 6.6.1 Data

We use the following two datasets:

- **CMU data:** ($N = 648$) hourly voltage and current for the Carnegie Mellon University (CMU) campus for 23 days, from July 29, 2016 to August 20, 2016. Voltage angle is unavailable for this data, so here $V_r$ is the voltage magnitude and $V_i = 0$.
- **LBNL data:** ($N = 3168$) from the Lawrence Berkeley National Laboratory (LBNL) Open $\mu$PMU project [SLR16], from October 1, 2015 to October 11, 2015. The data is originally at 120Hz, but we downsample it to one sample every 5 minutes (where each new data point is the mean of the raw data within those 5 minutes).

### 6.6.2 Q1. Forecasting accuracy

#### Baselines

our baselines are ARIMA [BJRL15], Holt-Winters (ETS) [Bro59], seasonal ARIMA [BJRL15], PowerCast (PCAST) [SHJ+17] (a recent tensor-based power grid forecasting approach), and vector autoregression (VAR), which uses temperature data and the voltage time sequences as input. Following standard practice, the ARIMA, SARIMA, and VAR orders are selected using AIC (Akaike information criterion) [HT93], and the Holt-Winters hyperparameters are selected

using nonlinear optimization. For PowerCast, we follow the original paper in setting $N_w = 5, \sigma = 0.5$.

**Experimental setup**

in each trial, an algorithm is given the data for the first $N$ days and forecasts $I_r$ and $I_i$ for each time point of the $(N+1)$th day, where we average each algorithm's accuracy over $N = 11, 12, \cdots, 20$ for CMU and $N = 4, 5, \cdots, 8$ for LBNL. The forecasts are compared with the true values using normalized RMSE: $\text{RMSE}(x, \hat{x}) = \sqrt{\|x - \hat{x}\|_2^2 / \|x\|_2^2}$, where $x = [I_r \;\; I_i]$ contains $I_r$ and $I_i$ values stacked into a single vector.

**Results:**

Figure 6.1 shows that StreamCast outperforms the baselines in both datasets, with at least $27\%$ lower RMSE. The tensor-decomposition based PCAST, which is also physics-based, is the second-best performer, suggesting that physics-based models may be beneficial for forecasting.

### 6.6.3 Q2. Scalability

We run StreamCast on a version of our CMU dataset, duplicated repeatedly so as to produce larger datasets. We run StreamCast on time series of sizes as plotted on the x-axis of Figure 9.6, from around 1 million to around 40 million. The plot is parallel to the diagonal, indicating linear growth.

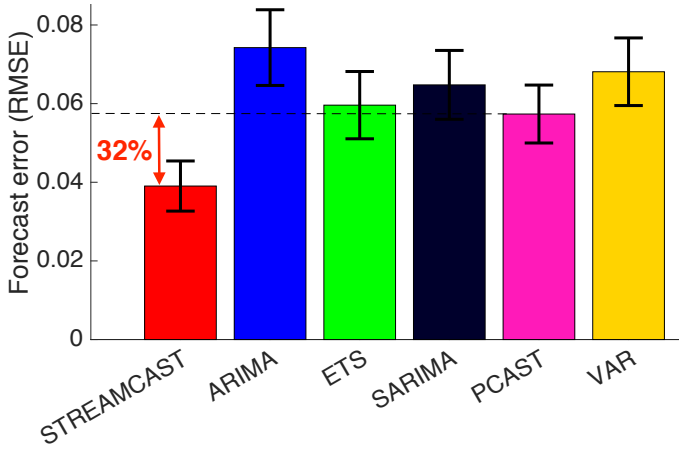
StreamCast takes less than 4 minutes for the trial of size 40 million, making it scalable to large datasets.

### 6.6.4 Q3. What-if scenarios

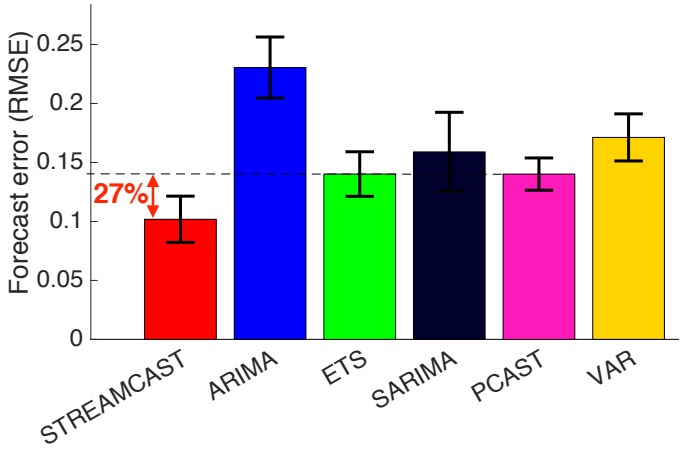
**Changing temperature and number of appliances:**

how can we forecast under the scenario that temperature increases by $10°C$, and number of appliances increases by $20\%$? Such scenarios are useful for future planning, but standard forecasting methods cannot handle them. The BIG parameter G represents the contribution of the conductive load component (e.g. light-bulbs) and B as the contribution of the reactive load component (e.g. motors), so we examine the results of increasing either G, B, or temperature, and plot how the forecasts change under each scenario.

The results are shown in Figure 6.3. The left plots are for increasing G; the center plots for changing B, and the right plots for increasing temperature. Upper plots are for $I_r$ while lower plots are for $I_i$. In each plot, the colored lines correspond to different amounts of increase: e.g. $1.1 \times G$ means that the amount of reactive load on campus increased by $10\%$. The results show that: 1) when G is increased, only $I_r$ increases, but not $I_i$; 2) when B is increased, only $I_i$ increases, but not $I_r$; 3) when temperature increases, both $I_r$ and $I_i$ increase. All three results are intuitive, given that in the CMU dataset we have $V_i = 0$; it can be verified from (6.2) that in this case $I_r$ should be influenced by changes in $G$, but not by changes in $B$.




(a) CMU data




(b) LBNL data

Figure 6.1: **StreamCast forecasts accurately:** it has at least 27% lower forecasting error than baselines. Error bars show 1 standard deviation.

### Changing voltage

An important goal in practice is to ensure that the system can make accurate predictions under changes to voltage levels. To test our model, we use an electrical system simulator, SUGAR [PJL+16]. The simulator uses physics-based models for different electrical equipment: we specify 10 motors and additional load with maximum resistance $50\Omega$, varying sinusoidally with daily periodicity. SUGAR generates realistic time series currents for an electrical system given specified input voltages $V_r, V_i$. In order to have a realistic voltage series, our input voltages are the voltage series from our CMU dataset. We obtain currents $(I_r, I_i)$ as outputs from this simulation.

To test StreamCast, we fit it on $(V_r, V_i, I_r, I_i)$, but then evaluate its RMSE on a different $(V_r', V_i', I_r', I_i')$ in which $V_r$ and $V_i$ are defined in one of two ways: 1) **Increase:** $V_r' = 1.05 \times V_r$,

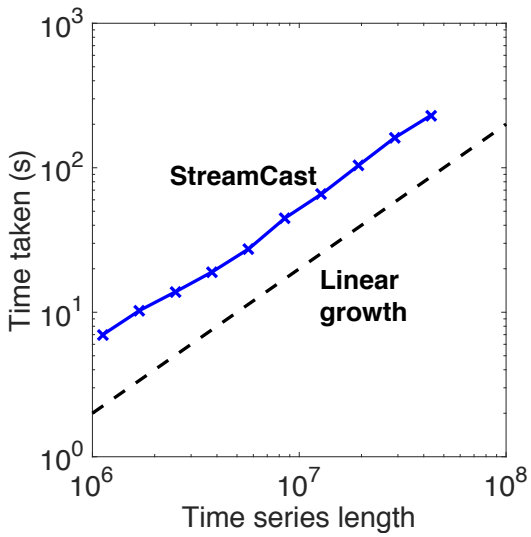



Figure 6.2: **StreamCast is fast and scales linearly:** growth parallel to the diagonal indicates linear growth.

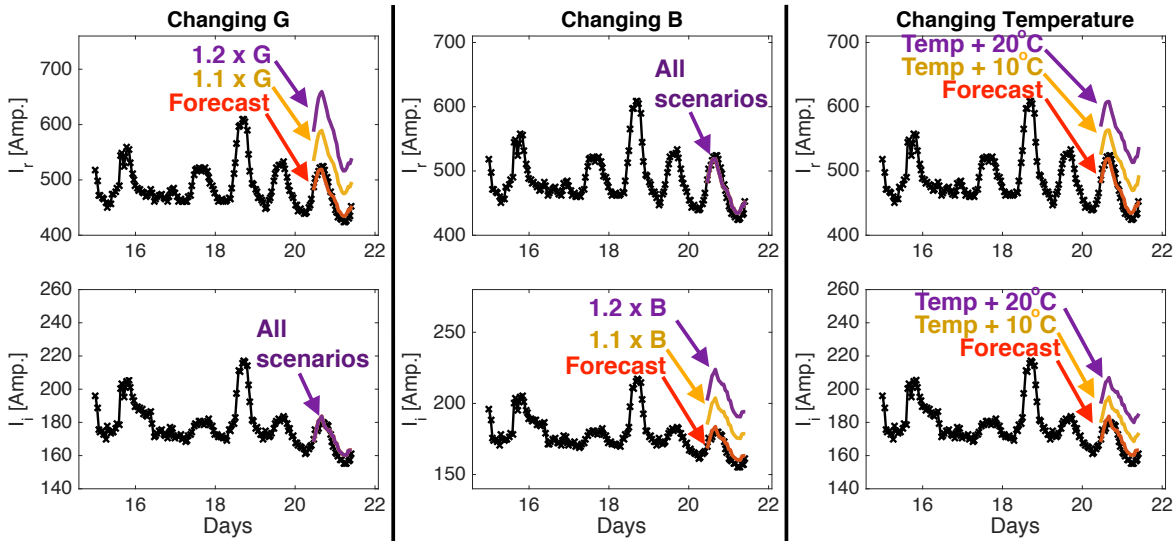



Figure 6.3: **StreamCast can handle forecasting under what-if scenarios:** the plots show the results of 1) increasing G; 2) increasing B; and 3) increasing temperature.

$V_i' = 1.05 \times V_i$. 2) **Decrease:** $V_r' = 0.95 \times V_r$, $V_i' = 0.95 \times V_i$. We then obtain $I_r'$ and $I_i'$ from the same electrical simulation under voltages $V_r'$ and $V_i'$.

Standard forecasting methods would not work as baselines, since an electrical model is needed to accurately predict what happens to $I$ when $V$ changes. Hence, we use the following baselines: 1) PQ: this is the same as our StreamCast approach, but substituting the BIG model with the more common PQ electrical model [PJL+16]. 2) PowerCast [SHJ+17] as in the previous

| Test case | StreamCast | PQ | PowerCast | Window2 | Window4 | Window8 | Window16 |
|---|---|---|---|---|---|---|---|
| Increase | **0.19** | 7.13 | 5.26 | 11.18 | 4.94 | 4.95 | 4.72 |
| Decrease | **0.27** | 7.69 | 5.66 | 9.46 | 5.78 | 5.56 | 4.58 |

Table 6.3: **StreamCast is accurate even under different voltage levels:** here the methods are tested on what-if scenarios with different voltage levels. Bold underline shows the best performer. Error values are given as **percentage** RMSE (i.e. RMSE $\times 100$).

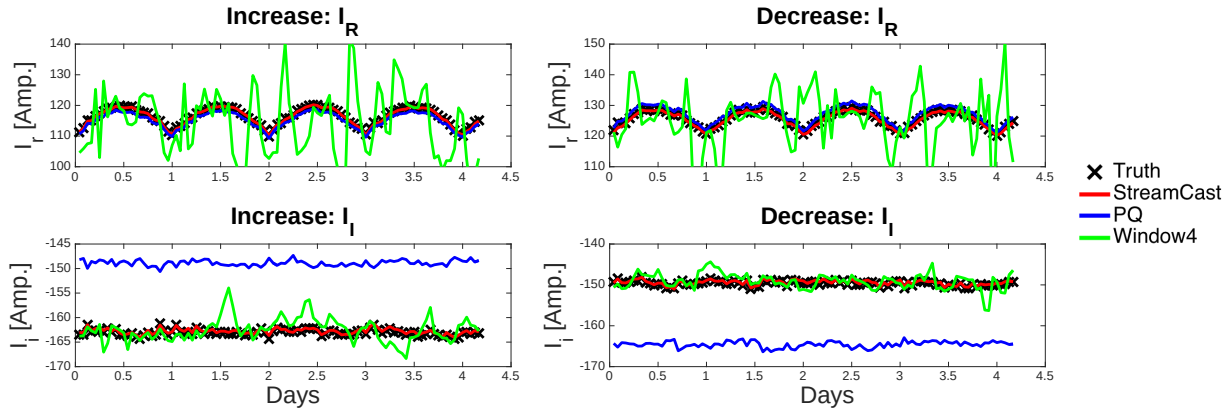



Figure 6.4: **StreamCast accurately responds to changes in voltage:** forecasts of $I_r$ and $I_i$ by each method vs. true values, when voltage was increased or decreased by $5\%$. StreamCast fits the data more accurately then baselines. RMSE results (with additional baseline methods) are in Table 6.3.

section; 3) Window$k$: this fits the *static* BIG model (Section 6.2.2) to short time windows of size $k$, where $k = 2, 4, 8, 16$.

Figure 6.4 shows the fit of StreamCast, PQ and Window4 against the true values (the remaining methods are not plotted for visibility, but their RMSE is in Table 6.3), and Table 6.3 shows the RMSE of each method against the true values. StreamCast outperforms the baselines by a clear margin. Because PQ is a constant-power model, increases in voltage magnitude necessarily lead to the model predicting a decrease in current of a similar magnitude. However, in practice, both voltage and current can change in the same direction, in which case the PQ model makes the wrong qualitative predictions. The Window$k$ baselines tend to overfit to near-term behavior due to their use of windows; this explains the high variance over time in Figure 6.4.

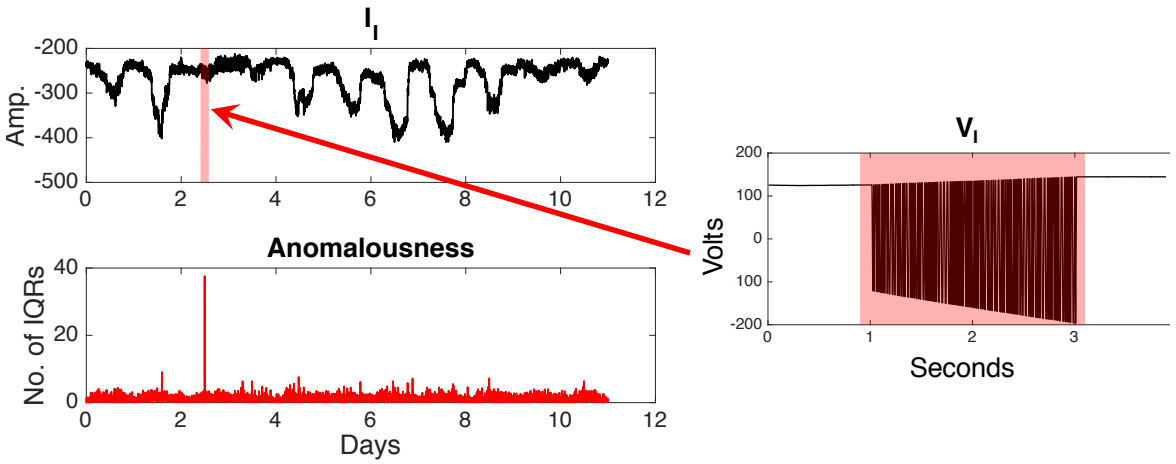



Figure 6.5: **StreamCast detects an anomaly in the LBNL dataset**. It corresponds to a 2-second period where voltage rapidly oscillates between negative and positive values.

### Anomaly Detection

Section 10.3 explains how to detect anomalies under our method; Figure 6.5 shows the results on the LBNL dataset, where our method outputs a plot ('anomalousness') of the anomaly score over time. There is a large spike in anomalousness around day 2; note that the anomaly is not at all visible from only the current time series. In the voltage time series, however, we find a 2-second period of quick oscillations between negative and positive values exactly at the time of the anomaly. Follow-up analysis reveals that the oscillation is likely an error in the measuring device: the complex angle of voltage is synchronized with the rest of the system via a GPS signal, and in case of loss of this GPS signal, we may observe oscillations such as those in Figure 6.5, which explains why the voltage magnitude remains stable while the angle oscillates.

### Confidence Intervals

Confidence intervals are useful for obtaining ranges of predictions: e.g. when monitoring the grid. Section 6.5.5 explains how we compute them. 95% and 99% confidence intervals on our CMU dataset are shown in Figure 6.6.

## 6.7 Conclusion

Our contributions are as follows:

1. **Domain knowledge infusion:** we propose a novel, *Temporal BIG* model that extends the physics-based BIG model, allowing it to capture changes over time, trends, seasonality, and temperature effects.
2. **Forecasting:** our StreamCast algorithm forecasts multiple steps ahead and outperforms baselines in accuracy. StreamCast is online, requiring linear time and bounded memory.
3. **What-if scenarios and anomaly detection:** our approach accurately handles scenarios in which the voltage levels, temperature, or number of appliances change. We also use it

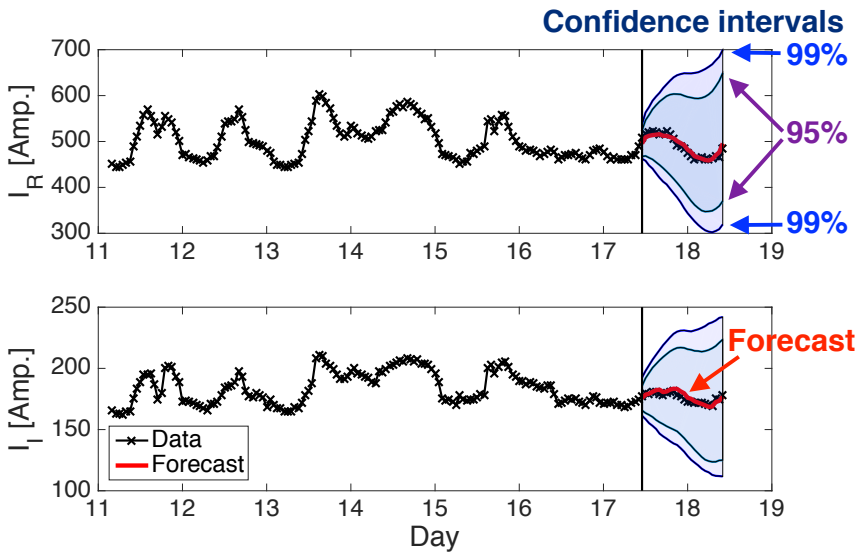



Figure 6.6: **StreamCast provides confidence intervals**.

to detect anomalies in a real dataset. Finally, StreamCast provides confidence intervals for its forecasts, to assist in planning for various scenarios.

**Reproducibility:** our code is publicly available at www.andrew.cmu.edu/user/bhooi/power.tar.

# Chapter 7

# BNB: Nonparametric Anomaly Detection in Mixed Time Series

Chapter based on work to appear at SDM 2019 [PDF].

Given mixed time series data, e.g. numerical, categorical, and ordinal data, how do we detect changes in the behavior of the time series: for example, the onset of illnesses or complications in patients? We propose BNB (Branch and Border), an online, nonparametric change detection method. Unlike existing methods, BNB approaches change detection by separating points before and after the change using an ensemble of random partitions. Our method (a) scales linearly; (b) provides theoretical guarantees on the false positive rate; and (c) is nonparametric and works on mixed data.

How do we detect change points in multivariate time series data? This problem has wide-ranging applications: in the medical domain, changes can indicate the onset of illnesses or complications, such as seizures, panic, and heart attacks [RGBK15]. Change detection on sensor data has been used to detect power plant failures and blackouts [BCDGV14]. Other applications include intrusion detection in computer networks [YTWM04], disease outbreak detection [KHH+05], and location recognition for robots [LYCS13].

In many cases, we cannot assume that the data follows any standard distribution (e.g. Gaussian, Poisson). In realistic settings, data can be multi-modal, skewed, and nonlinear; moreover, different columns of the data may have different types: e.g. binary, numerical, ordinal, or count data. We also often want to detect change points in an **online** manner: many types of data (e.g. medical data) are received in real time. As each data point is received, the algorithm should update efficiently. Thus, our goal is an online, nonparametric change detection algorithm:

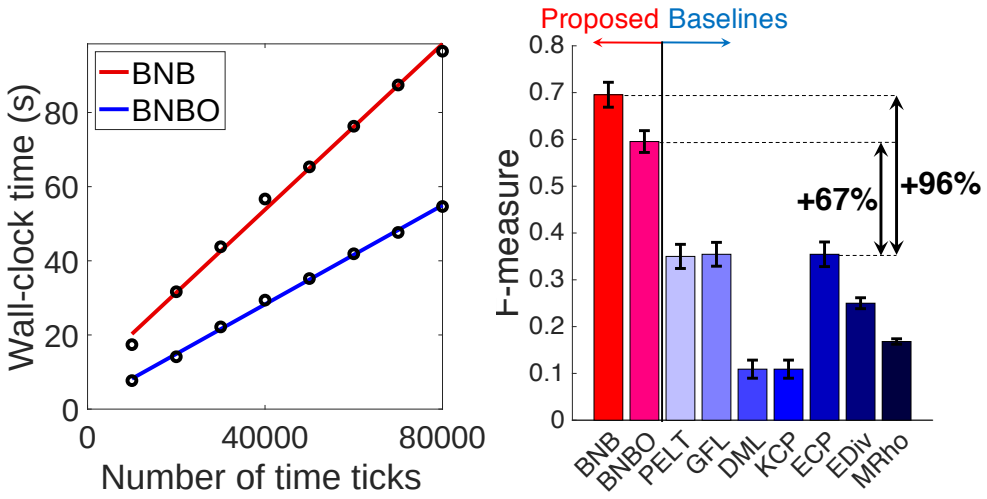
**Informal Problem 7.1: Online Change Detection**

- **Given** a multivariate stream $X_1, X_2, \cdots$ (possibly containing numerical, ordinal or categorical data);
- **Find** a set of time ticks $t$ where changes occurred, reported in a streaming manner.

Change detection in time series [SK74, CF15] has been studied extensively (expanded in Section 9.2). Our work differs in two key aspects. Firstly, most work focuses on purely numerical data, whereas we want to allow for e.g. count, ordinal and categorical data. Secondly, we use a nonparametric, random partition based approach. Algorithms which make distributional assumptions can be negatively affected when these do not hold: e.g. Gaussian or Poisson-based approaches will be overly influenced by outliers given heavy-tailed data, and report wrong change points. Moreover, many real world datasets may be hard to fit using **any** standard distribution: e.g. nonlinear or multimodal data.

Our contributions are as follows:

1. **Algorithm:** We propose an online nonparametric change detection algorithm based on random partitions, a novel approach for change detection.
2. **Scalability:** BNBO is linear (Figure 7.1a) and online, using bounded memory and time per iteration.
3. **Effectiveness:** Our algorithms outperform baselines in accuracy by $70\%$ or more (Figure 7.1b), in experiments on real and synthetic data. Theorem 7.3 provides a theoretical guarantee on the false positive rate.




(a) Scalability (b) Accuracy

Figure 7.1: **BNB is scalable and accurate:** (a) linear scalability of BNB and its online variant, BNBO (for a time series with $100$ dimensions) (b) F-measure of detecting change points, averaged over $11$ datasets.

**Reproducibility:** our code and data are publicly available at http://www.andrew.cmu.edu/user/bhooi/bnb/bnb.zip.

## 7.1 Related Work

[AC17] reviews time series change detection methods.

**Supervised Change Detection** Supervised methods treat change detection as a binary classification task, or directly output class labels [RMB+10]. However, labeled data is often unavailable.

**Parametric Change Detection** Many approaches rely on a parametric statistical model: e.g. Gaussians [KFE12], and more general exponential family distributions [FMS14]. Accompanying these are search algorithms: greedy binary segmentation [SK74], bottom-up segmentation [KCHP01], dynamic programming (DP) [JSB+05]. Wild binary segmentation [Fry14] uses binary segmentation on random intervals. PELT [KFE12] applies pruning to DP. Other approaches include Hidden Markov Models [KCG+15], Group Fused Lasso (GFL) [BV11], and Distance Metric Learning (DML) [XJRN03], which uses a Mahalanobis metric.

**Online Change Detection** FLOSS [GDY+17] is an online segmentation approach for multivariate data. GGS [HNB16] is an online, multivariate, Gaussian approach. CUSUM [ZWZJ15] is an online approach for several parametric families; other control-chart approaches include e.g. MEWMA [QLW16]. CD [QAWZ15] uses a PCA-based, online multivariate approach.

**Other Methods** Nonparametric change detection methods include Multivariate Rho (MRHO) [KQR16], and E-statistic based EDIV [MJ14] and ECP [MJ14], which use nonparametric goodness-of-fit measures. Other methods include Kernel Change Point Detection (KCP) [DDD05], and neural networks [LLJ02]. Other methods exist for discrete change detection: e.g. see [CBK12] for a survey.

**Forest-Based Outlier Detection** [LTZ08] proposed Isolation Forests, which detect outliers based on the intuition that outliers have low depth in randomly constructed trees. [TTL11] proposes a similar but streaming approach. However, outlier detection and change detection require different intuitions: outlier detection is non-temporal, and finds deviations from the data, while change detection requires a time-series, and finds transitions from one regime to another.

Table 10.1 summarizes existing change point detection methods. BNB differs from existing methods in that it is online, nonparametric, and allows for multiple data types. Our approach is also very different, relying on an ensemble of random partitions.

## 7.2 Problem Definition

### 7.2.1 Problem Setting

Table 10.2 shows the symbols used in this chapter.

We are given $X$, an $n \times d$ dataset, i.e. it has $n$ time ticks with $d$ dimensions. Denote by $X_t$ the data point at time $t$, and denote by $X_{tj}$ the data value in dimension $j$ at time $t$. Our

Table 7.1: Comparison of relevant change detection approaches. 'Mixed Data Types' refers to allowing data types beyond just numerical data: including categorical, count, and ordinal data.

| *Property* | PELT [KFE12] | GFL [BV11] | GGS [HNB16] | DML [XJRN03] | FLOSS [GDY+17] | ZWZJ [ZWZJ15] | KCP [DDD05] | MRHO [KQR16] | EDIV [MJ14] | ECP [MJ14] | CD [QAWZ15] | **BNBO** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Multivariate Data** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✔ |
| **Nonparametric** | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✔ |
| **Scales Linearly** | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | | | ✔ |
| **Online Algorithm** | | | ✓ | | ✓ | ✓ | | | | | ✓ | ✔ |
| **Mixed Data Types** | | | | | | | | | | | | ✔ |

Table 7.2: Symbols and definitions

| **Symbol** | **Interpretation** |
|---|---|
| $n$ | Number of time ticks in data |
| $d$ | Number of dimensions in data |
| $X_t$ | Data point at time $t$ |
| $X_{tj}$ | Data value at time $t$ in dimension $j$ |
| $w$ | Window size |
| $d_{lim}$ | Depth limit for trees |
| $N$ | Number of trees |
| $s_{k,t}$ | Separation depth for tree $k$ at time $t$ (Definition 7.2) |
| $c_t$ | Change score for time $t$ (Definition 7.3) |
| $z_t$ | Indicator vector for ancestors of $X_t$ (Section 7.4.3) |
| $s_L, s_R$ | Left and right ancestor counts (Section 7.4.3) |
| $s_I$ | Intersection set (Section 7.4.3) |

goal is to detect **change points**: i.e. time ticks where the behavior of the time series changes significantly. Formally:

**Problem 7.1: Multivariate Change Detection**

- **Given** a multivariate dataset of $t$ time ticks with $d$-dimensions ($X_{tj}$ for $j = 1, \cdots, d$ and $t = 1, \cdots, n$) containing numeric, ordinal or categorical data;
- **Output** a set of time ticks where changes occurred.

A major challenge is the **online** setting, in which we receive the data incrementally, one time tick at a time, i.e. $X_1, X_2, \cdots$. In some cases the stream may be infinite, in which case $n = \infty$. As we receive the data, we should output which time ticks are change points.

However, it would be unrealistic to expect to determine whether time point $t$ is a change point without any future information after time $t$: this is because the time ticks $t+1, t+2, \cdots$ are important in deciding whether an apparent change at time $t$ was a true change point, or simply noise. Hence, we introduce a **window** parameter $w$: upon receiving time tick $t+w$, the algorithm must output whether time $t$ is a change point.

**Problem 7.2: Online Change Detection**

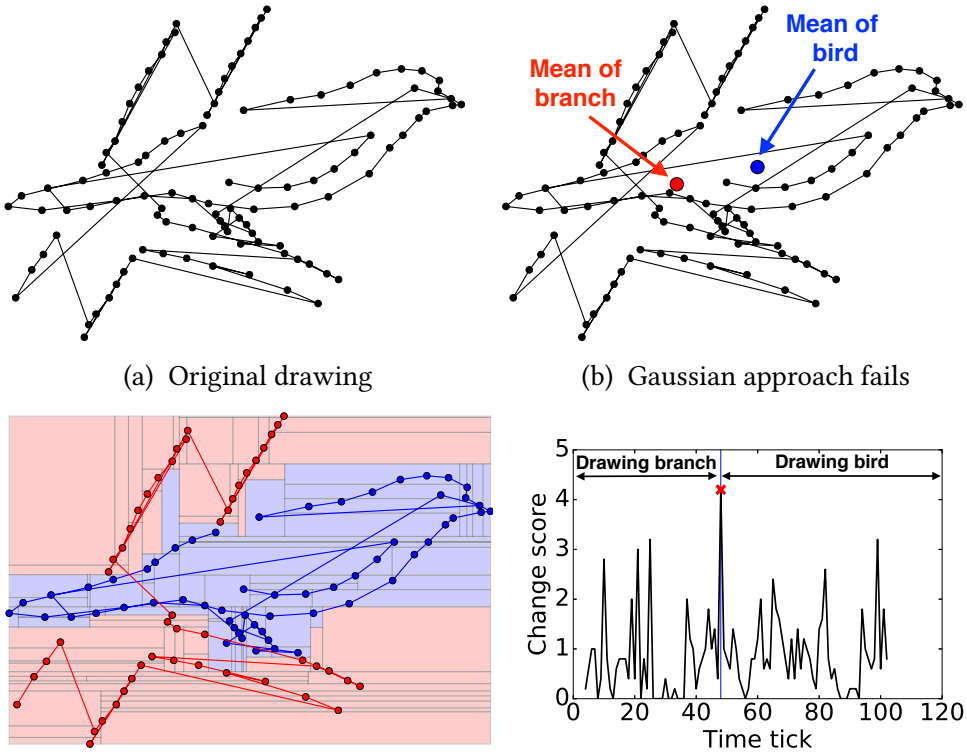
- **Given** a multivariate stream of $d$-dimensional data ($X_{tj}$ for $j = 1, \cdots, d$ and $t = 1, 2, \cdots$), possibly containing numeric, ordinal or categorical data;
- **Output** at time $t + w$ whether a change point occurred at time tick $t$.

## 7.3 Illustrative Example

The main idea of our approach is its use of random partitions to measure how good a change is. Consider a 1-dimensional toy dataset: $X = [0, 1, 0, 0, 10, 10, 9, 10]$, which clearly has a change point at $t = 5$. If we sample a random cut point uniformly between 0 and 10 (the min and max of the data), then with $80\%$ probability, the cut point lies in $(1, 9)$. In this case, the cut point perfectly separates the points before and after $t = 5$. Intuitively then, $t = 5$ is a good change point because the points before and after (or equal to) it are easily separable by random partitions.

In real data, single cuts do not always suffice, and we may need multiple cuts, but the intuition is similar. Consider the drawing of a bird on a branch in Figure 7.2a, containing an unknown change point, from drawing the branch to drawing the bird. The connecting line segments (in black) indicate that the drawing is a time series, not just a set of points. Finding the change point is challenging: the bird's feet and tail are hard to separate from the branch. Figure 7.2b shows that the Gaussian approach has difficulty: these approaches rely on a large difference in means before and after the change, but the bird and branch have fairly close means.

Figure 7.2c illustrates our random partition approach. We conduct random cuts (gray lines) that partition the space recursively. At the time of the correct change, the points before the change (in red) and after the change (in blue) can be cleanly separated by the random partition:




(a) Original drawing

(b) Gaussian approach fails

(c) Random partition approach finds change point

(d) Peak in change score coincides with ground truth

Figure 7.2: **BNB is adaptive and nonparametric:** (a) A drawing of a bird on a branch, with an unknown change point (when moving from the branch to the bird) (b) The bird and the branch have similar means, making them difficult to separate using standard approaches. (c) At the time of the true change, the red points (before the change) and blue points (after the change) can be fully separated using a relatively simple random partition, suggesting that this is a good change. The blue region (boxes only containing points after the change) conforms to the bird's outline well despite the bird's unique shape which is hard to separate from the branch. (d) Averaging over many such partitions, the change score is highest at the true change time.

i.e. all boxes contain either only red or only blue points. The stronger the change point, the easier the red and blue points are to cleanly separate, and the simpler the random partition we would need to cleanly separate them. Hence, in Figure 7.2d we compute a score for each possible change point based on how easily we can separate the points before and after the change using random partitions (we formalize this in Section 7.4). Averaging over many such partitions, Figure 7.2d shows that the highest change score indeed coincides with the true change point.

## 7.4 Proposed BNB Algorithm

### 7.4.1 Random Partition Tree

We first explain how our random trees are built. Given a dataset $X$ of size $n$ with $d$ dimensions, we first uniformly sample one of the $d$ dimensions to split. Then, we sample the split point: we sample two independent uniform values between the minimum and maximum of the data values, and use their mean as the split point. This causes split points to tend toward the middle of the range, making the children more balanced in their node counts. We then recurse onto each child, stopping when a node contains only one data point, or reaches a depth limit $d_{lim}$.

Recall that we want to allow for ordinal and categorical data. These can be easily handled by splitting ordinal data at a randomly chosen split point, and splitting categorical data by randomly dividing the categories into two sets.

**Definition 7.1: Random Partition Tree**

A random partition tree is built by recursively sampling a feature, and then sampling a split point to partition the data space into two, continuing recursively until a depth of $d_{lim}$ is reached.

### 7.4.2 Change Score

Our main intuition is that if time tick $t$ is a good change point, then the data before and after the change point should be cleanly separable using relatively simple partitions, i.e. shallow trees. For a given candidate change point $t$, define the '*left set*' $L$ as the $w$ points before time $t$ (i.e. $\{X_{t-w}, \cdots, X_{t-1}\}$). Similarly, define the '*right set*' $R$ as the $w$ points at or after time $t$ (i.e. $\{X_t, \cdots, X_{t+w-1}\}$). Given a single random partition tree $\mathcal{T}$, define the '*separation depth*' as the minimum depth at which $\mathcal{T}$ fully separates the left set from the right set. Here 'fully separated' means that like in Figure 7.2c, each node of $\mathcal{T}$ at that depth either contains only points from $L$ or $R$, but not both.

**Definition 7.2: Separation Depth**

The separation depth $s_{k,t}$ for the $k$th random partition tree at time tick $t$ is the minimum depth at which $\mathcal{T}_k$ fully separates the left set from the right set.

Intuitively, low separation depth means that $L$ and $R$ can be easily separated, which means $t$ is a good change point. If $L$ and $R$ are not fully separated when reaching the depth limit $d_{lim}$, we set the separation depth to $d_{lim} + 1$, which functions as an upper limit to prevent any single tree from being too influential.

Finally, we average the separation depth values over $N$ random partition trees. Since $d_{lim}+1$ is an upper bound, we then use $d_{lim}+1$ minus this average separation depth as our change score: thus, higher values indicate a better change, with $0$ being the minimum value indicating a poor change.

**Definition 7.3: Change Score**

The change score $c_t$ at time tick $t$ is $d_{lim}+1$ minus the average separation depth computed using $N$ random partition trees:

$$c_t = d_{lim} + 1 - \frac{1}{N}\sum_{j=1}^{N} s_{j,t} \tag{7.1}$$

### 7.4.3 Efficient Implementation

The step of computing change scores can be sped up significantly over the naive approach. Implemented naively, at each of $n$ time ticks, we would need to compute $N$ separation depths, each taking $O(w)$ time, for a total of $O(n \cdot N \cdot w)$. However, we can save a factor of $O(w)$, by computing the change scores in a single pass over the dataset.

Consider a random partition tree $\mathcal{T}$. Every data point $X_t$ for each $t$ has been assigned to a leaf of $\mathcal{T}$ during the tree-building step. For convenience, let the nodes in $\mathcal{T}$ be assigned integer ids $i = 1, 2, \cdots$ arbitrarily. Define $s_L$ ('*left ancestor count*'), a sparse vector: for each node $i$ of $\mathcal{T}$, $s_L[i]$ is the number of points in the left set that node $i$ contains. For example, in Figure 7.2c, each node corresponds to a box; then $s_L[i]$ is the number of points from the left set in that box. Define $s_R$ ('*right ancestor count*') similarly using the right set. Define $s_I$ ('*intersection set*') as the set of nodes which have nonzero counts in both $s_L$ and $s_R$.

**Lemma 7.1**

At time $t$, the separation depth is $1$ more than the maximum depth among nodes in $s_I$.

*Proof.* All nodes with nonzero counts in $s_L$ are those with descendants in the left set, and similarly for $s_R$, so $s_I$ contains nodes with descendants from both sets. If $D$ is the maximum depth among nodes in $s_I$, the tree at depth $D$ is not enough to perfectly separate the left and right sets, since a node at depth $D$ has descendants from both sets (so the node contains points from the left and the right sets). However, the tree at depth $D+1$ does perfectly separate the left and right sets, otherwise there would exist a node in $s_I$ with depth $D+1$. ■

Algorithm 9.1 shows our change scoring algorithm. We initialize $s_L, s_R, s_I$ and separation depths $s_{j,t}$, then maintain them efficiently as we move forward in time.

Let $z_t$ be a 0-1 vector with 1 in entry $i$ iff node $i$ is an ancestor of the leaf containing the data point $X_t$. Note then that $s_L$ is just the sum of $z_t$ over the left set, and similarly for $s_R$. Initially, the left set contains the first $w$ time ticks, so $s_L$ is initialized as $s_L = \sum_{t=1}^{w} z_t$ (Line 4). Similarly $s_R = \sum_{t=w+1}^{2w} z_t$ (Line 5). Next we initialize $s_I$ as the set of nodes with nonzero counts in both $s_L$ and $s_R$ (Line 7), and compute the separation depth $s_{j,w+1}$ following Lemma 7.1 (Line 8).

**Algorithm 7.1:** CHANGESCORE

**Input** : Dataset $X$, $N$ random partition trees, window $w$
**Output**: Change scores $c_t$ for $w \leq t \leq n - w + 1$

1 ▷For each tree:
2 **for** $j$ *in* $1$ *to* $N$ **do**
3  ▷Initialize left and right ancestor counts
4  $s_L = \sum_{t=1}^{w} z_t$
5  $s_R = \sum_{t=w+1}^{2w} z_t$
6  ▷Intersection set and separation depth
7  $s_I = \{k : s_L[i] > 0 \text{ and } s_R[i] > 0\}$
8  $s_{j,w+1} = 1 + \max_{i \in s_I} \text{depth}(i)$
9 **end**
10 **for** $t$ *in* $w$ *to* $n - w + 1$ **do**
11  **for** $j$ *in* $1$ *to* $N$ **do**
12  ▷Update due to change in left and right sets
13  $s_L \leftarrow s_L + z_t - z_{t-w}$
14  $s_R \leftarrow s_R - z_{t+w} - z_t$
15  $s_I = \{x : s_L[x] > 0 \text{ and } s_R[x] > 0\}$
16  $s_{j,t} = 1 + \max_{x \in s_I} \text{depth}(x)$
17  **end**
18 **end**
19 **return** $c_t = d_{lim} + 1 - \frac{1}{N} \sum_{j=1}^{N} s_{j,t}, \ \forall\, t$

To maintain these values, for $s_L$ and $s_R$ we add and subtract the relevant vectors $z_t$ (Lines 13 and 14). For each such change, we incrementally update the intersection set $s_I$ (Line 15) and thus the separation depth (Line 16). Finally, we compute the overall change score by averaging over all the trees (Line 19).

Algorithm 7.2 summarizes our full offline algorithm BNB. It estimates $N$ random partition trees, then uses CHANGESCORE to compute scores.

### 7.4.4 Online Algorithm (BNBO)

We now explain how BNB can be modified to be run online. The change score computation CHANGESCORE is already online: Lines 2 to 8 are based on only the first $2w$ time ticks, while Lines 13 to 16 are based only on the current window. Hence, it adapts to streams of arbitrary length. At time $t$, to compute $z_t$, for each tree, we 'classify' $X_t$ by recursively following the split points, from the root node down to whichever leaf node $X_t$ belongs to.

**Algorithm 7.2:** Offline Change Detection BNB

**Input** : Dataset $X$, window $w$, number of trees $N$, depth limit $d_{lim}$
**Output**: Change scores $c_t$ for $w \leq t \leq n - w + 1$

```
for j in 1 to N do
    ▷Random Partition Tree creation (Section 7.4.1)
    T_j = MakeTree(X, d_lim)
end
return ChangeScore(X, w, T)
```

Only the tree generation algorithm MakeTree is not yet online, as it uses the full dataset to choose each split point. BNBO uses a small 'initialization' dataset of 250 points, which we use for MakeTree to decide the split points for all trees. The resulting trees are then used in the online version of ChangeScore for future data points in the stream.

### 7.4.5 Time Complexity

For BNB's tree-building step, we have $N$ trees, each requiring at most $d_{lim}$ splits, where all splits combined in each level involve $n$ data points. Hence, tree-building takes $O(N \cdot n \cdot d_{lim})$. For change scoring, we have $N$ trees, each of which has to be updated $O(n)$ times, and each update is $O(d_{lim})$ since each $z_t$ has at most $d_{lim}$ nonzeroes, and Lines 13 to 16 are sparse vector operations with $O(d_{lim})$ nonzeroes. Hence overall runtime is $O(N \cdot n \cdot d_{lim})$.

For the same reasons, BNBO's tree building step requires $O(N \cdot d_{lim})$. For each online update, computing $z_t$ is still proportional to tree height, since we travel the height of the tree, so each online update is $O(N \cdot d_{lim})$.

## 7.5 Theoretical Analysis

### 7.5.1 Interpretation of Separation Depth

Can we better understand what BNB does, by re-interpreting it in a more theoretical way: e.g. is there a metric space (i.e. a space where we can measure distances), for which our approach is related to distances in this space? We show that the answer is 'yes'.

A **metric space** is a set $\mathcal{X}$ and a distance function $d : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ such that:

1. $d(x, y) = 0$ iff $x = y$ (Identity)
2. $d(x, y) = d(y, x)$ (Symmetry)
3. $d(x, z) \leq d(x, y) + d(y, z)$ (Triangle Inequality)

Given a tree $\mathcal{T}_k$, the tree partitions the data space, mapping each data point $X_t$ into a leaf of $\mathcal{T}_k$: let leaf$(X_t)$ map $X_t$ to its leaf. Define $\mathcal{X}$ as the set of leaves, and $d(x, y)$ as the shortest path distance (in terms of number of graph hops) between leaves $x$ and $y$.

**Lemma 7.2**

$(\mathcal{X}, d)$ is a metric space.

*Proof.* This is a special case of the fact that geodesic distance in a graph forms a metric space, if and only if the graph is connected [BDFG03]. In our case, the graph is a tree, hence the leaves indeed form a metric space. ■

**Theorem 7.1**

The separation depth $s_{k,t}$ can be rewritten in terms of distances in this metric space:

$$s_{k,t} = d_{lim} - \frac{1}{2} \min_{x \in L, y \in R} d(\text{leaf}(x), \text{leaf}(y)) + 1 \tag{7.2}$$

*Proof.* Consider the pair of points $x \in L$ and $y \in R$ minimizing (7.2). Their shortest path distance is $d(\text{leaf}(x), \text{leaf}(y))$, so their deepest common ancestor (call it $z$) must be $d(\text{leaf}(x), \text{leaf}(y))/2$ steps above them. Since $\text{leaf}(x)$ and $\text{leaf}(y)$ are at depth $d_{lim}$, thus $z$ is at depth $d_{lim} - d(\text{leaf}(x), \text{leaf}(y))/2$. Moreover, since $x$ and $y$ were the closest pair of points, $z$ must be the deepest node with descendants in both $L$ and $R$. Thus, $L$ and $R$ are fully separated exactly at depth $d_{lim} - d(\text{leaf}(x), \text{leaf}(y))/2 + 1$, but not at any lower depth (due to node $z$). ■

This implies that separation depth $s_{k,t}$, which measures how good a change is, is effectively a nearest-neighbor like statistic (though not in a Euclidean space).

### 7.5.2 Bounds on False Positive Rate

Our main theorems bound our false positive rate. Assume that no change is present; as is standard in change detection literature, we use the null hypothesis $H_0 : X_{t-w}, \cdots, X_{t+w-1}$ are i.i.d., representing no change, and bound the probability of high values of change score. First consider a single tree, which outputs the separation depth $s_{k,t}$ as change score. Then:

**Theorem 7.2**

For any $\lambda > 0$, letting $p_\lambda = 2^{\lambda - 1}$, we have:

$$P(s_{k,t} \leq \lambda) \leq (1/2)^{w - p_\lambda}. \tag{7.3}$$

*Proof.* Fix an arbitrary ordering $<$ on the data space (e.g. lexicographic ordering), and let $X^1, \cdots, X^{2w}$ be the original data points $(X_1, \cdots, X_{2w})$ sorted according to the order $<$. Since

the data points $X_1, \cdots, X_{2w}$ are i.i.d., if we condition on $X^1, \cdots, X^{2w}$, by symmetry, the conditional probability that the original data is mapped to $X^1, \cdots, X^{2w}$ by any particular permutation is equal, which is $1/(2w)!$.

Define $Z_i$ as a 0-1 random variable, taking value 0 if $X^i$ is one of $X_1, \cdots, X_w$, and 1 otherwise. Then exactly $w$ of the random variables $Z_1, \cdots, Z_{2w}$ are 1, and since we earlier showed that every permutation of the $X^1, \cdots, X^{2w}$ is equally likely, thus now each assignments of 0s and 1s to $Z_1, \cdots, Z_{2w}$ with exactly $w$ 1s is also equally likely. Thus, by symmetry, each such assignment has probability $1/\binom{2w}{w}$.

For any $\lambda$, $P(s_{k,t} \leq \lambda)$ is the probability that the sets $L$ and $R$ were fully separated by the time the tree reached level $\lambda$. Note that at level $\lambda$, our tree partitions the data space into $p_\lambda = 2^{\lambda-1}$ parts. The number of assignments to the $Z$ variables for which $L$ and $R$ are fully separated is then at most $2^{p_\lambda}$, since each of the $p_\lambda$ parts we have to choose to assign either 0 or 1 to all variables in that part. Moreover, we earlier showed that each of the $\binom{2w}{w}$ assignments has the same probability. Thus, the probability that $L$ and $R$ are fully separated at level $\lambda$ is at most

$$P(s_{k,t} \leq \lambda) \leq \frac{2^{p_\lambda}}{\binom{2w}{w}} \tag{7.4}$$

$$= \frac{2^{p_\lambda}(1)\dots(w)}{(w+1)\dots(2w)} \tag{7.5}$$

$$\leq 2^{p_\lambda}(1/2)^w \tag{7.6}$$

$$\leq (1/2)^{w-p_\lambda}. \tag{7.7}$$

In fact, this can be tightened significantly by using stronger bounds for central binomial coefficients: it is known that $\binom{2w}{w} \geq 4^w/\sqrt{4w}$ for any positive integer $w$ [Kaz14]. Substituting this instead gives

$$P(s_{k,t} \leq \lambda) \leq \frac{2^{p_\lambda}}{\binom{2w}{w}} \tag{7.8}$$

$$\leq \frac{2^{p_\lambda}\sqrt{4w}}{4^w} \tag{7.9}$$

$$= 2^{p_\lambda - 1 - 2w}\sqrt{w} \tag{7.10}$$

$$= (1/2)^{2w-1-p_\lambda}\sqrt{w}. \tag{7.11}$$

■

The next theorem concerns our final change score $c_t$. For any error threshold $\varepsilon > 0$, we have:

**Theorem 7.3**

$$P(c_t \geq d_{lim} + 1 - \log(1 + \log\frac{\varepsilon}{N} + w)) \leq N(1/2)^{w-p_\lambda} \tag{7.12}$$

*Proof.* Applying union bound on the (weaker) result of the previous theorem gives

$$P(s_{k,t} \leq \lambda) \leq N(1/2)^{w-p_\lambda} \; \forall \; k \in \{1, \cdots, N\} \tag{7.13}$$

Recalling that $c_t = d_{lim} + 1 - \frac{1}{N}\sum_{j=1}^{N} s_{j,t}$, substituting this gives:

$$P(c_t \geq d_{lim} + 1 - \lambda) \leq N(1/2)^{w-p_\lambda} \tag{7.14}$$

Finally, if $p_\lambda = \log \frac{\varepsilon}{N} + w$, then $N(1/2)^{w-p_\lambda} \leq \varepsilon$. Substituting this into (7.14) (and using the fact that $\lambda = \log p_\lambda + 1$) gives the result.

■

This implies that any change scores above this value can be used to trigger an alarm with $1 - \varepsilon$ confidence, i.e. at most $\varepsilon$ probability of a false positive.

## 7.6 Experiments

We design experiments to answer the questions:

- **Q1. Change Detection Accuracy:** how accurate are BNB and BNBO compared to baselines?
- **Q2. Scalability:** do they scale linearly?
- **Q3. Real-World Discoveries:** how do they perform on an indoor occupancy detection task?

Experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM running OS X 10.11.2. We implement our method in Python. For our methods we set $w = 15, d_{lim} = 15$ and $N = 50$, but show experiments on parameter sensitivity in this section.

We evaluate our algorithms on multivariate time series with various sizes and domains, described in Table 10.3. The `Occupancy` dataset contains ground truth change labels, which are the time points when the room changed from occupied to unoccupied, or vice versa.

### 7.6.1 Q1. Detection Accuracy

We now evaluate our algorithms' accuracy in detecting change points.

#### Single Change Points

For each dataset, we sample a change point as a random time tick between $n/4$ to $3n/4$ (rounded). For data points on or after the change point, we add a Gaussian with mean $0$ and standard deviation for each dimension equal to the standard deviation of the data for that dimension. Then the goal of each algorithm is to detect the change point (exactly). For each algorithm we pass input parameters for detecting one change point, and evaluate the results according to F-measure against the true change point. As several of the baselines scale quadratically or slower, we subset all datasets to a size of $500$. However, in Section 8.5.2 we show that our algorithms easily scale to $10K - 80K$ time ticks and $100 - 800$ dimensions. The results are averaged over $20$ repetitions.

Table 7.3: Datasets used

| Dataset Name | Time Ticks | Dimensions | Content |
|---|---|---|---|
| Chemical [DVMP+08] | 9357 | 13 | Chemical Sensor |
| Beijing [LZG+15] | 43824 | 6 | Air Quality |
| Exchange [LCYL17] | 7588 | 8 | Exchange Rate |
| Measles [VPGJ+13] | 2501 | 50 | Disease Counts |
| Influenza [VPGJ+13] | 2501 | 50 | Disease Counts |
| Scarlet [VPGJ+13] | 2501 | 50 | Disease Counts |
| Whooping [VPGJ+13] | 2501 | 50 | Disease Counts |
| Vehicle1 [HA18] | 264 | 2 | Unmanned Vehicle |
| Vehicle2 [HA18] | 342 | 2 | Unmanned Vehicle |
| Vehicle3 [HA18] | 578 | 2 | Unmanned Vehicle |
| Vehicle4 [HA18] | 488 | 2 | Unmanned Vehicle |
| Occupancy [CF16] | 9752 | 5 | In-room Sensors |
| Synthetic | 10K-80K | 100-800 | Synthetic |

### Baselines

We compare BNB to the following recently proposed multivariate change detection methods:

- *Parametric:* PELT (Killick, 2012) is a dynamic programming (DP)-based approach. GFL (Bleakley, 2011) uses group-fused lasso. DML (Xing, 2003) is a distance metric learning approach, using a Mahalanobis metric.
- *Nonparametric:* KCP (Desobry, 2005) is a kernel-based approach. EDiv (Matteson, 2014) and ECP (Matteson, 2014) use the E-statistic, a nonparametric goodness-of-fit statistic, with hierarchical division and DP. MRho (Kojadinovic, 2016) uses Spearman's rho.

We show results for unnormalized data in Table 7.4, and normalized data in Table 7.5. In both, BNB and BNBO perform the best on almost all the datasets. Normalization does not significantly affect our methods, as all their steps are invariant to scaling. The baselines mostly perform slightly better with normalization than without. The results averaged over all 11 datasets is in Figures 7.3a and 7.3b. BNB and BNBO significantly outperform the baselines: e.g. by $82\%$ or more for BNB. Between our two methods, BNB generally performs better, but BNBO still outperforms the baselines significantly despite all the baselines being offline.

### Multiple Change Detection

We use the same settings, except with 5 change points, sampled uniformly without replacement from $0.1n$ to $0.9n$ (rounded). We evaluate each algorithm based on its top $5$ changes, again using F-measure, compared to the true changes. Results are averaged over $20$ repetitions.

The results are shown in Table 7.6 for unnormalized data and Table 7.7 for normalized data. BNB and BNBO again outperform the baselines on almost all the datasets. The average F-

| | BNB | BNBO | PELT | GFL | DML | KCP | ECP | EDiv | MRho |
|---|---|---|---|---|---|---|---|---|---|
| Chemical | **0.70** | **0.70** | 0.50 | 0.50 | 0.15 | 0.15 | 0.45 | 0.28 | 0.00 |
| Beijing | **0.80** | 0.55 | 0.00 | 0.00 | 0.15 | 0.15 | 0.00 | 0.05 | 0.00 |
| Exchange | **1.00** | 0.90 | 0.00 | 0.10 | 0.00 | 0.00 | 0.15 | 0.00 | 0.00 |
| Measles | **0.95** | **0.95** | 0.40 | 0.45 | 0.20 | 0.20 | 0.15 | 0.33 | 0.33 |
| Influenza | **0.95** | 1.00 | 0.40 | 0.55 | 0.10 | 0.10 | 0.75 | 0.45 | 0.45 |
| Scarlet | 0.55 | **0.60** | 0.45 | 0.55 | 0.20 | 0.20 | 0.05 | 0.28 | 0.28 |
| Whooping | **1.00** | 0.70 | 0.00 | 0.00 | 0.20 | 0.20 | 1.00 | 0.40 | 0.40 |
| Vehicle1 | **0.65** | 0.60 | 0.35 | 0.40 | 0.10 | 0.10 | 0.10 | 0.17 | 0.00 |
| Vehicle2 | 0.40 | **0.50** | 0.10 | 0.10 | 0.05 | 0.05 | 0.15 | 0.05 | 0.00 |
| Vehicle3 | **0.25** | 0.15 | 0.15 | 0.15 | 0.05 | 0.05 | 0.10 | 0.07 | 0.00 |
| Vehicle4 | **0.95** | 0.70 | 0.25 | 0.25 | 0.05 | 0.05 | 0.70 | 0.05 | 0.00 |

Table 7.4: **BNB and BNBO win consistently:** F-measure for single change point detection (without normalization)

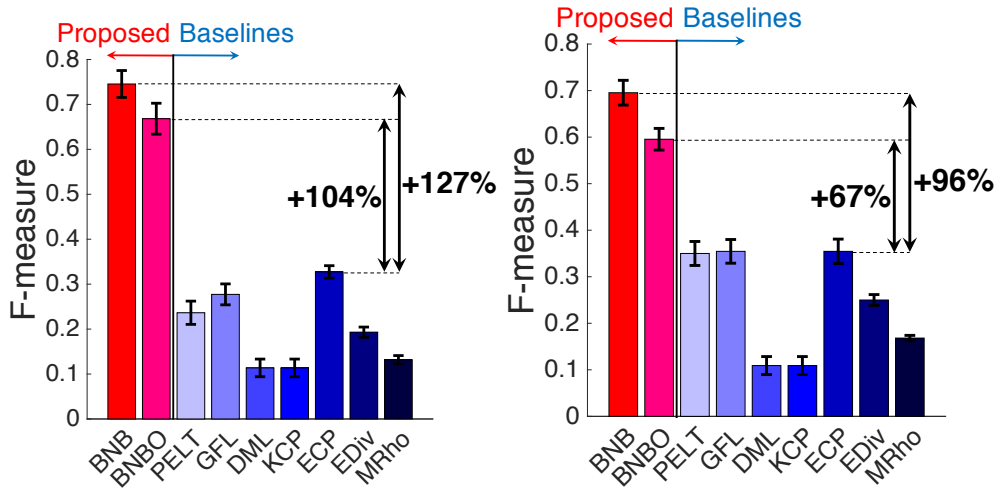
| | BNB | BNBO | PELT | GFL | DML | KCP | ECP | EDiv | MRho |
|---|---|---|---|---|---|---|---|---|---|
| Chemical | 0.70 | **0.75** | 0.65 | 0.65 | 0.30 | 0.30 | 0.50 | 0.28 | 0.00 |
| Beijing | **0.50** | 0.40 | 0.40 | 0.40 | 0.20 | 0.20 | 0.10 | 0.20 | 0.00 |
| Exchange | **0.95** | 0.85 | 0.25 | 0.25 | 0.10 | 0.10 | 0.50 | 0.12 | 0.00 |
| Measles | **0.95** | 0.90 | 0.80 | 0.90 | 0.20 | 0.20 | 0.00 | 0.42 | 0.42 |
| Influenza | **0.95** | **0.95** | 0.20 | 0.20 | 0.20 | 0.20 | 0.40 | 0.47 | 0.47 |
| Scarlet | 0.60 | 0.35 | **0.90** | **0.90** | 0.40 | 0.40 | 0.40 | 0.40 | 0.40 |
| Whooping | **1.00** | 0.70 | 0.00 | 0.00 | 0.15 | 0.15 | **1.00** | 0.47 | 0.47 |
| Vehicle1 | **0.50** | **0.50** | 0.30 | 0.30 | 0.10 | 0.10 | 0.25 | 0.20 | 0.00 |
| Vehicle2 | 0.15 | **0.30** | 0.00 | 0.00 | 0.00 | 0.00 | 0.10 | 0.03 | 0.00 |
| Vehicle3 | **0.40** | 0.30 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.03 | 0.00 |
| Vehicle4 | **0.95** | 0.55 | 0.45 | 0.45 | 0.05 | 0.05 | **0.95** | 0.20 | 0.00 |

Table 7.5: **BNB and BNBO win consistently** (with one exception): F-measure for single change point detection (with normalization)

measure over all datasets are shown in Figures 7.4a and 7.4b. BNB outperforms the baselines by 70% or more F-measure on average.

### Parameter Sensitivity

Figure 7.5 compares performance for different parameter values, on multiple change point detection (without normalization). Results are averaged over datasets and over 5 repetitions; the other settings follow the previous section. Our methods perform consistently well across values.




(a) Without normalization (b) With normalization

Figure 7.3: **BNB and BNBO outperform baselines for single changes:** (a) F-measure of detecting change points, averaged over 11 datasets. Error bars indicate one standard deviation. (b) Same results, with normalization for all methods.

| | BNB | BNBO | PELT | GFL | DML | KCP | ECP | EDiv | MRho |
|---|---|---|---|---|---|---|---|---|---|
| Chemical | **0.68** | 0.54 | 0.52 | 0.37 | 0.12 | 0.12 | 0.40 | 0.38 | 0.00 |
| Beijing | **0.62** | 0.61 | 0.21 | 0.23 | 0.03 | 0.03 | 0.19 | 0.23 | 0.00 |
| Exchange | **0.91** | 0.84 | 0.43 | 0.35 | 0.08 | 0.08 | 0.40 | 0.31 | 0.00 |
| Measles | **0.90** | 0.85 | 0.37 | 0.50 | 0.11 | 0.11 | 0.26 | 0.41 | 0.41 |
| Influenza | **0.99** | **0.99** | 0.29 | 0.59 | 0.14 | 0.14 | 0.61 | 0.50 | 0.50 |
| Scarlet | **0.60** | 0.56 | 0.31 | 0.40 | 0.10 | 0.10 | 0.10 | 0.29 | 0.29 |
| Whooping | **0.98** | 0.95 | 0.64 | 0.42 | 0.26 | 0.26 | 0.75 | 0.52 | 0.52 |
| Vehicle1 | 0.68 | **0.71** | 0.28 | 0.29 | 0.09 | 0.09 | 0.21 | 0.20 | 0.00 |
| Vehicle2 | **0.50** | **0.50** | 0.38 | 0.29 | 0.15 | 0.15 | 0.45 | 0.33 | 0.03 |
| Vehicle3 | **0.54** | 0.44 | 0.41 | 0.36 | 0.06 | 0.06 | 0.34 | 0.30 | 0.02 |
| Vehicle4 | **0.89** | 0.72 | 0.68 | 0.44 | 0.14 | 0.14 | 0.68 | 0.43 | 0.00 |

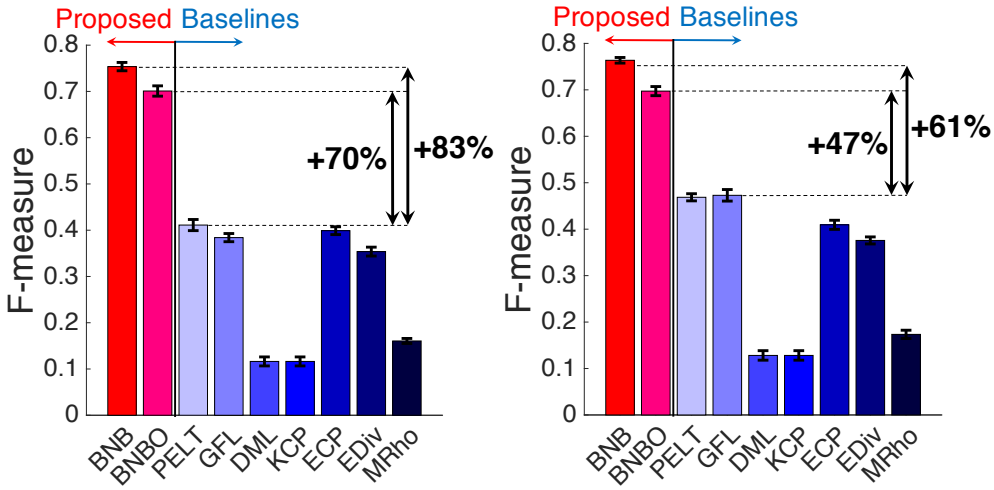
Table 7.6: **BNB and BNBO win consistently:** F-measure for multiple change point detection (without normalization)

### 7.6.2 Q2. Scalability

Next, we verify that our methods scale linearly. We repeatedly duplicate our Occupancy dataset in time ticks and dimensions, add Gaussian noise to each dimension with standard deviation equal to the standard deviation of that dimension, then subset the data to the required size. Figure 9.6a shows that BNB and BNBO scale linearly in time ticks; here dimensions is fixed at 100. Figure 7.6b shows that they scale linearly in dimensions; here time ticks is fixed

| | BNB | BNBO | PELT | GFL | DML | KCP | ECP | EDiv | MRho |
|---|---|---|---|---|---|---|---|---|---|
| Chemical | **0.68** | 0.55 | 0.58 | 0.63 | 0.14 | 0.14 | 0.45 | 0.41 | 0.02 |
| Beijing | **0.74** | 0.67 | 0.24 | 0.42 | 0.09 | 0.09 | 0.33 | 0.22 | 0.00 |
| Exchange | **0.93** | 0.87 | 0.57 | 0.51 | 0.11 | 0.11 | 0.47 | 0.37 | 0.02 |
| Measles | **0.91** | 0.87 | 0.36 | 0.69 | 0.14 | 0.14 | 0.26 | 0.45 | 0.45 |
| Influenza | **0.98** | 0.97 | 0.40 | 0.59 | 0.12 | 0.12 | 0.62 | 0.48 | 0.48 |
| Scarlet | **0.71** | 0.62 | 0.49 | **0.71** | 0.11 | 0.11 | 0.22 | 0.42 | 0.42 |
| Whooping | **0.99** | 0.95 | 0.64 | 0.48 | 0.17 | 0.17 | 0.72 | 0.55 | 0.55 |
| Vehicle1 | **0.61** | 0.61 | 0.26 | 0.24 | 0.05 | 0.05 | 0.17 | 0.15 | 0.00 |
| Vehicle2 | 0.49 | **0.50** | 0.41 | 0.28 | 0.08 | 0.08 | 0.41 | 0.32 | 0.00 |
| Vehicle3 | **0.56** | 0.50 | 0.43 | 0.30 | 0.06 | 0.06 | 0.29 | 0.35 | 0.02 |
| Vehicle4 | **0.91** | 0.63 | 0.62 | 0.47 | 0.21 | 0.21 | 0.62 | 0.47 | 0.00 |

Table 7.7: **BNB and BNBO win consistently:** F-measure for multiple change point detection (with normalization)




(a) Without normalization (b) With normalization

Figure 7.4: **BNB and BNBO outperform baselines for multiple changes:** (a) F-measure of detecting multiple change points, averaged over 11 datasets. Error bars indicate one standard deviation. (b) Same results, with normalization.

at $10,000$. Both methods are fast: for $100$ dimensional data, BNB takes $1.2$ms per time tick on average, while BNBO takes $0.7$ms, on a laptop computer. The online nature of BNBO means that in settings where we need incremental results, it provides further efficiency gains compared to repeatedly running an offline algorithm.

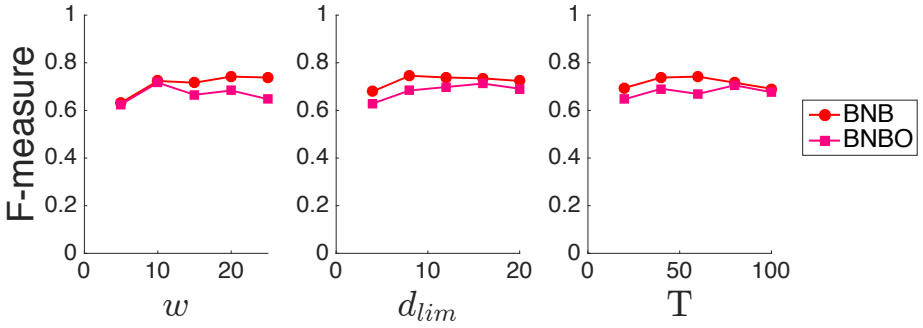
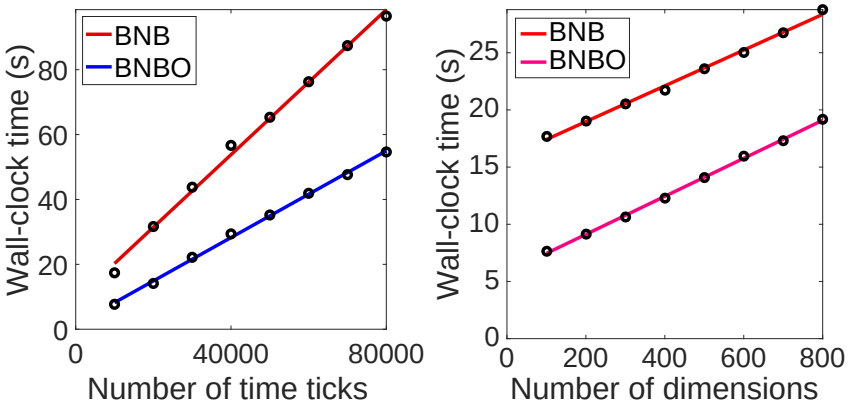



Figure 7.5: **Insensitive to parameters:** our methods perform consistently across parameter values.




(a) Time ticks (b) Dimensions

Figure 7.6: **Our methods scale linearly.**

### 7.6.3 Q3. Real-World Effectiveness

We now evaluate our methods against the same baselines on a real-world occupancy detection task [CF16], with ground truth change points when the room changed from being unoccupied to occupied, or vice versa, resulting in 13 change points. The sensors measure the room's temperature, humidity, light and carbon dioxide levels. Ground truth occupancy was obtained from time-stamped pictures taken every minute. We evaluate using F-measure, where reported change points are considered as correctly matched to a given ground truth change if the two are at most '*tolerance*' minutes apart, where we plot different values of '*tolerance*' in Figure 7.7. We repeat each method 5 times and average the results.

BNB and BNBO outperform the baselines, with BNB having F-measure of 115% higher accuracy. We omit GFL, DML and KCP as they did not terminate after 16 hours. (BNB and BNBO took 57s and 28s.)

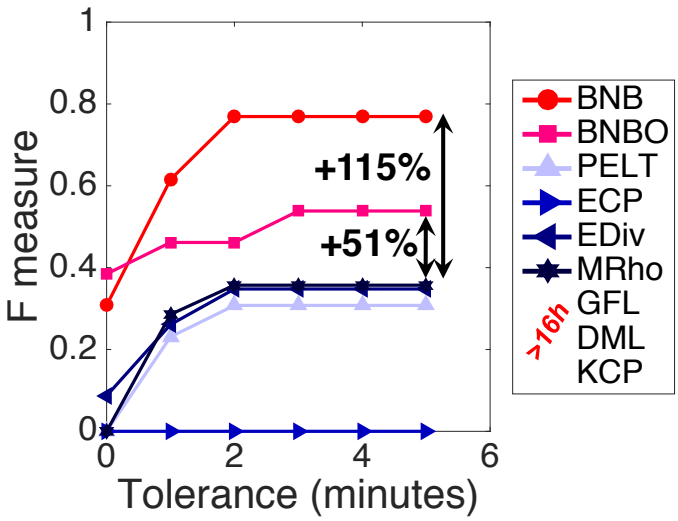



Figure 7.7: **Accuracy for occupancy detection:** our methods detect changes in occupancy in a room accurately, based on F-measure.

## 7.7 Conclusion

We propose BNB, which uses an ensemble of random partitions to measure change score. It generalizes to different data types, and BNBO provides an online approach. Our contributions are:

1. **Algorithm:** We propose a novel online nonparametric change detection approach.
2. **Scalability:** BNBO is linear (Figure 7.1a) and online, using bounded memory and time per iteration.
3. **Effectiveness:** Our algorithms outperform baselines in accuracy by 70% F-measure or more, on real and synthetic data. Theorem 7.3 provides a theoretical guarantee on the false positive rate.

# Chapter 8

# SMF: Drift-Aware Matrix Factorization with Seasonal Patterns

Chapter based on work to appear at SDM 2019 [PDF].

In this chapter we consider time-series matrix data: e.g. a stream of taxi rides, each recording the start and end locations of each ride. How do we learn a matrix factorization model which takes into account drift and seasonal patterns, and use it to forecast and perform anomaly detection? In this chapter, we propose SMF (Seasonal Matrix Factorization), a matrix factorization model for seasonal data, which is (a) accurate, outperforming baselines by $13\%$ to $60\%$ in RMSE; (b) online; and (c) provides interpretable results. We also propose SMF-A, which performs anomaly detection in a computationally feasible way, without forecasting every observation in the matrix.

## 8.1 Introduction

Consider a stream of events, represented as tuples of the form ($entity_1$, $entity_2$, $time$). Given such data, a natural goal is to model patterns, and to forecast future data: for example, the number of taxi rides from Brooklyn to Manhattan tomorrow. Other similar applications includes disease forecasting, movie recommendation, retweet prediction, etc. In all these cases, seasonal patterns are present, and relevant to making accurate forecasts. Hence, our problem is:

**Informal Problem 8.1: Forecasting**

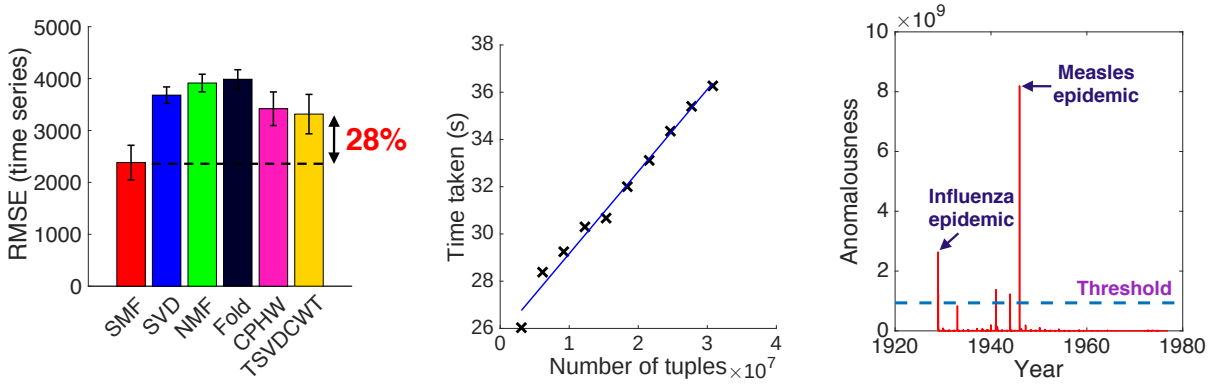
- **Given:** a stream of past events, containing seasonal patterns;
- **Forecast:** the number of events between each pair of entities at any future time tick.

Another goal is to detect when anomalies occur. These could involve a sudden increase in activity, but can also be any unusual shift in activity (e.g. a road accident redirecting traffic from one lane to another).

**Informal Problem 8.2: Anomaly Detection**

- **Given:** a stream of past events, containing seasonal patterns;
- **Find:** a measure of how much each entity deviates from normal behavior at each time tick.

Standard matrix factorization approaches model this data by ignoring the time dimension, resulting in a matrix. However, these approaches ignore **seasonal** patterns. Taxi activity typically follows a daily bimodal pattern, peaking at morning and evening peak hours. In addition, standard matrix factorization cannot capture **drift**, or changes in the components over time: such as population growth, or people entering or leaving a community.




(a) Accurate (b) Scales linearly (c) Detects anomalies

Figure 8.1: **SMF is accurate, scales linearly, and detects anomalies.** (a) Forecast error of SMF compared to state-of-the-art baselines. (b) Running time scales linearly. (c) SMF-A detects two large epidemics, which have been previously reported in the medical literature, in a diseases dataset.

Scalability is also a major challenge, both in memory and running time, since matrix factorization often involves large numbers of entities. The entire dataset may not fit in memory, or may even have no finite size, in an **online** setting. Hence, we propose SMF (Seasonal Matrix

Factorization), a drift-and-seasonality aware matrix factorization model which can be fit using an online algorithm. Our contributions are:

- **Model:** we propose a novel matrix factorization model incorporating seasonal patterns and drift, and an online algorithm for fitting this model.
- **Effectiveness:** in experiments, SMF has lower forecasting error than baselines by 13% to 60% (Figure 8.1a), and provides interpretable results in case studies on real data.
- **Scalability:** SMF is online, and scales linearly (Figure 8.1b). In experiments, it was 12 to 103 times faster than seasonal baselines.
- **Fast Anomaly Detection:** we propose SMF-A for detecting anomalies (Figure 8.1c) in a computationally feasible way, without forecasting every possible observation in the matrix.

**Reproducibility:** our code and datasets are publicly available at `www.andrew.cmu.edu/user/bhooi/code`.

## 8.2 Background and Related Work

**Static Matrix Factorization** Matrix Factorization (MF) techniques including SVD [SKKR00], NMF [LS01], and pLSI [Hof04] have been widely explored, particularly in collaborative filtering [Kor08, Kor10, SPV14]. Other work incorporated bias terms [Kor08], and alternating least squares [BK07].

**Dynamic Matrix Factorization** Time-weighting schemes weight past data by their recency [DL05, DKA11]. Other approaches include temporal regularization [XCH+10] and Kalman filters [SPV14, COL13]. timeSVD++ [Kor10] modifies SVD with a temporal bias term. [DKM15] uses dynamic MF with priors. [XCH+10] proposes a Bayesian approach. [STF06] proposes a dynamic tensor analysis algorithm. However, none of these consider seasonal patterns.

**Seasonal Patterns in Matrix Factorization** Fold [ENLSS16] combines data at the same point in the season, then uses 3-way tensor decomposition (CPD). [THF17] similarly separates recurring patterns from outliers. CPHW [DKA11] uses 3-way CPD, then extends the temporal factor matrix using the Holt-Winters algorithm. [dARF17] uses a similar approach, also incorporating coupled tensor factorizations.

**Why not use 3-way CPD?** 3-way CPD [KB09, SDLF+17] treats the temporal dimension as a discrete, unordered variable. Hence, it cannot be directly used for forecasting, and also does not model component drift. To forecast, we could modify it, e.g. as in Fold or CPHW. Compared to Fold and CPHW, which have fixed components, SMF allows both the components and seasonal patterns to drift: this includes both drifting component strength (e.g. a community growing more active) and drifting component structure (e.g. a community changing in composition). We verify that SMF learns meaningful such drifts in taxi data in Section 8.5.3, and our experiments show that SMF outperforms Fold and CPHW in forecasting accuracy. Another difference is that both Fold and CPHW are offline algorithms, while SMF is online.

Table 8.1: Comparison between methods.

| *Property* | SVD/NMF [SKKR00, LS01] | Dynamic MF [DKM15, DL05] | 3-way CPD [KB09] | TimeCrunch [SKZ+15] | Fold [ENLSS16] | CPHW, etc [DKA11, dARF17] | **SMF** |
|---|---|---|---|---|---|---|---|
| **Component-based** | ✓ | ✓ | ✓ | | ✓ | ✓ | ✔ |
| **Seasonal patterns** | | | | ✓ | ✓ | ✓ | ✔ |
| **Drifting component strength** | | ✓ | | | | ✓ | ✔ |
| **Drifting component structure** | | | | | | | ✔ |
| **Online algorithm** | some | ✓ | some | | | | ✔ |

## 8.3 Model

### Preliminaries

The input data is a series of sparse matrices $\mathcal{A}(t), t = 1, 2, \cdots, r$. For example, in the taxi case, if there were 100 taxi rides from location $i$ to location $j$ at time $t$, then the $(i, j)$th entry of $\mathcal{A}(t)$ is 100. Table 8.2 shows the symbols used. For matrix indexing, $\mathcal{X}(:, 1:2)$ is the submatrix of $X$ with all its rows and the first 2 columns.

### 8.3.1 Proposed SMF Model

To capture the desired seasonality patterns, we introduce **seasonal weights** $w_i(t)$: $w_i(t)$ is a (scalar) multiplier that applies to component $i$ at time $t$. Thus:

$$\mathcal{A}(t) \approx \sum_{i=1}^{k} \mathbf{u}_i(t) w_i(t) \mathbf{v}_i(t)^T \tag{8.1}$$

$w_i(t)$ will allow us to capture seasonal patterns, because we will ensure that $w_i(t)$ itself is close to periodic over time. Following Figure 8.2, in matrix notation this is:

$$\mathcal{A}(t) \approx \mathcal{U}(t) \mathcal{W}(t) \mathcal{V}(t)^T \tag{8.2}$$

We model the data using smoothly varying components $\mathbf{u}$ and $\mathbf{v}$, and seasonally varying 'multipliers' $w$, which govern the seasonal patterns in the data. This model captures multiple types of change: drifting component strength corresponds to variation in $w_i(t)$. Drifting community structure corresponds to variation in $\mathbf{u}_i(t)$.

Table 8.2: Symbols and definitions

| Symbol | Definition |
|---|---|
| $\mathcal{A}(t)$ | $m \times n$ sparse data matrix at time $t$ |
| $m, n$ | Number of rows and columns in $\mathcal{A}(t)$ |
| $r$ | Number of time steps |
| $k$ | Number of components |
| $\mathbf{u}_i(t), \mathbf{v}_i(t)$ | Factorization component $i$ at time $t$ |
| $w_i(t)$ | (Scalar) seasonal multiplier $i$ at time $t$ |
| $\mathcal{U}(t)$ | $m \times k$ matrix form of $\mathbf{u}_i(t)$: $\mathcal{U}(t) = [\mathbf{u}_1(t) \cdots \mathbf{u}_k(t)]$ |
| $\mathcal{V}(t)$ | $n \times k$ matrix form of $\mathbf{v}_i(t)$: $\mathcal{V}(t) = [\mathbf{v}_1(t) \cdots \mathbf{v}_k(t)]$ |
| $\mathcal{W}(t)$ | $k \times k$ diagonal matrix: $\mathcal{W}(t) = \text{diag}(w_1(t), \cdots, w_k(t))$ |
| $s$ | Period (e.g. 7 for daily data with weekly periodicity) |
| $\alpha$ | Gradient step size hyperparameter |
| $z(t)$ | Number of nonzeroes in $\mathcal{A}(t)$ |
| $z$ | Total number of nonzeroes: $z = \sum_{t=1}^{r} z(t)$ |
| $\mathcal{X}(:, 1:2)$ | Submatrix of $\mathcal{X}$ with all rows and first 2 columns |

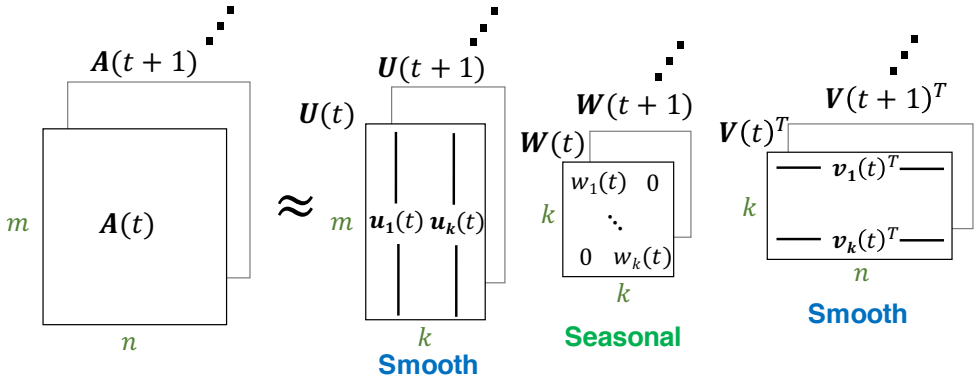



Figure 8.2: An illustration of our model. Section 8.4 explains how we model smoothness in $\mathcal{U}$ and $\mathcal{V}$, and seasonality in $w$.

## 8.4 Proposed SMF Algorithm

We now propose SMF, an online optimization algorithm, which has two steps: **Initialization**, where we use a short initial time period to train an initial model, then **Online Update**, where we repeatedly observe the next time point and update our model. Note that standard fitting methods cannot be used as they are generally offline: since we have defined $\mathcal{U}, \mathcal{W}, \mathcal{V}$ as functions of time, storing all of them simultaneously would require too much memory, and cause memory usage to grow over time.

### 8.4.1 Initialization Step

We start by initializing $\mathbf{u}$, $\mathbf{v}$, and $w$. A reasonable initialization requires a few seasons of data: we use the first 3 seasons, following a common practice for initializing seasonal ETS models [HK+07]. We thus 'stack' up $\mathcal{A}(1), \cdots, \mathcal{A}(3s)$ into a $m \times n \times 3s$ sparse tensor $\mathcal{T}_{\text{init}}$. Next, we 'fold' this into a $m \times n \times s$ sparse tensor $\mathcal{T}_{\text{fold}}$:

$$\mathcal{T}_{\text{fold}} = \frac{1}{3} \sum_{i=1}^{3} \mathcal{T}_{\text{init}}(:, :, (i-1) \cdot s + 1 : i \cdot s) \tag{8.3}$$

We then run nonnegative CP decomposition [XY13] on $\mathcal{T}_{\text{fold}}$. We use the resulting component as $\mathcal{U}(0)$ and $\mathcal{V}(0)$. For the third component, we use its value in component $i$ at time $t$ as the seasonal multipler $w_i(t)$, for $t = -s+1, \cdots, 0$. The negative indices for $t$ are used so that starting at $t = 1$, we have valid values when we access the 'previous season' of $w_i(t)$. To allow the $w_i$ to reflect component strength, we normalize $\mathbf{u}_i(0)$ by dividing by its norm $\|\mathbf{u}_i(0)\|$, compensating by multiplying $\|\mathbf{u}_i(0)\|$ into each of the $w_i(t)$ for $t = -s+1, \cdots, 0$ instead. We do the same for $\mathbf{v}_i(0)$.

### 8.4.2 Online Updates

As we receive each $\mathcal{A}(t)$ for $t = 1, 2, \cdots$, we need to update $\mathcal{U}, \mathcal{V}$ and $\mathcal{W}$ in an online way to preserve good model fit. Assume that we have fit $\mathcal{U}, \mathcal{V}$ and $\mathcal{W}$ up to time $t-1$. At time $t$, we start by setting $\mathcal{U}(t)$ and $\mathcal{V}(t)$ equal to $\mathcal{U}(t-1)$ and $\mathcal{V}(t-1)$, and $\mathcal{W}(t)$ equal to $\mathcal{W}(t-s)$. We then adjust $\mathcal{U}$ and $\mathcal{V}$ by taking a small gradient step in the direction given by minimizing error with respect to $\mathcal{A}(t)$. The gradient step keeps error with respect to $\mathcal{A}(t)$ low (i.e. $\mathcal{A}(t) \approx \mathcal{U}(t)\mathcal{W}(t)\mathcal{V}(t)^T$). Taking a small step ensures that $\mathcal{U}$ and $\mathcal{V}$ are smooth ($\mathcal{U}(t) \approx \mathcal{U}(t-1)$), while $\mathcal{W}$ is near-seasonal ($\mathcal{W}(t) \approx \mathcal{W}(t-s)$). The fitted parameters 'track' the true values over time as we perform gradient updates. Meanwhile, each update is highly efficient as it only involves gradients with respect to $\mathcal{A}(t)$.

Let $\hat{\mathcal{A}}(t) = \mathcal{U}(t-1)\mathcal{W}(t-s)\mathcal{V}(t-1)^T$. For adjusting $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$, the gradient update to $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$ can be computed by differentiating fitting error. $\alpha > 0$ determines the learning rate.

$$\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t-1) + \alpha(\mathcal{A}(t) - \hat{\mathcal{A}}(t))\mathbf{v}_i(t-1)w_i(t-s)$$
$$\mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t-1) + \alpha(\mathcal{A}(t) - \hat{\mathcal{A}}(t))^T\mathbf{u}_i(t-1)w_i(t-s)$$

Next, we ensure that the nonnegativity constraint is met by projecting $\mathbf{u}$ and $\mathbf{v}$: $\mathbf{u}_i(t) \leftarrow \max(0, \mathbf{u}_i(t))$ and $\mathbf{v}_i(t) \leftarrow \max(0, \mathbf{v}_i(t))$, where $\max$ is applied elementwise. Finally, we renormalize $\mathbf{u}_i(t)$ and $\mathbf{v}_i(t)$ by dividing by their norms $\|\mathbf{u}_i(t)\|$ and $\|\mathbf{v}_i(t)\|$ respectively while multiplying these norms into $w_i(t)$:

$$w_i(t) \leftarrow w_i(t-s) \cdot \|\mathbf{u}_i(t)\| \cdot \|\mathbf{v}_i(t)\| \tag{8.4}$$
$$\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t)/\|\mathbf{u}_i(t)\|; \quad \mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t)/\|\mathbf{v}_i(t)\| \tag{8.5}$$

This allows our seasonality pattern $w$ to adapt over time, while also ensuring that the normalization constraint is met for $\mathbf{u}$ and $\mathbf{v}$.

**Algorithm 8.1:** Online Updates

**Input** : Sparse matrices $\mathcal{A}(1), \cdots, \mathcal{A}(r)$, initialization for $\mathcal{U}, \mathcal{V}, \mathcal{W}$
**Output**: $\mathcal{U}, \mathcal{V}, \mathcal{W}$

1 **for** $t = 1$ **to** $r$ **do**
2  ▷Perform gradient updates
3  $\hat{\mathcal{A}}(t) = \mathcal{U}(t-1)\mathcal{W}(t-s)\mathcal{V}(t-1)^T$
4  $\mathcal{U}(t) \leftarrow \mathcal{U}(t-1) + \alpha(\mathcal{A}(t) - \hat{\mathcal{A}}(t))\mathcal{V}(t-1)\mathcal{W}(t-s)$
5  $\mathcal{V}(t) \leftarrow \mathcal{V}(t-1) + \alpha(\mathcal{A}(t) - \hat{\mathcal{A}}(t))^T\mathcal{U}(t-1)\mathcal{W}(t-s)$
6  **for** $i = 1$ **to** $k$ **do**
7  ▷Project onto nonnegative constraint
8  $\mathbf{u}_i(t) \leftarrow \max(0, \mathbf{u}_i(t))$, $\mathbf{v}_i(t) \leftarrow \max(0, \mathbf{v}_i(t))$
9  ▷Renormalize and multiply into $w$
10  $w_i(t) \leftarrow w_i(t-s) \cdot \|\mathbf{u}_i(t)\| \cdot \|\mathbf{v}_i(t)\|$
11  $\mathbf{u}_i(t) \leftarrow \mathbf{u}_i(t)/\|\mathbf{u}_i(t)\|$
12  $\mathbf{v}_i(t) \leftarrow \mathbf{v}_i(t)/\|\mathbf{v}_i(t)\|$
13  **end**
14 **end**

Note that only the last time step of $\mathcal{U}$ and $\mathcal{V}$, and the last $s$ time steps of $\mathcal{W}$ are needed at any time. This prevents memory usage from growing over time.

### 8.4.3 Speeding up Updates

$\hat{\mathcal{A}}(t)$ is a dense $m \times n$ matrix, so explicitly forming it is inefficient both in running time and memory. Instead, we can rewrite the gradient updates in Lines 4 and 5 of Algorithm 8.1 into a more efficiently computable form. Let $z(t)$ be the number of nonzeroes in $\mathcal{A}(t)$.

**Lemma 8.1**

$(\mathcal{A}(t) - \hat{\mathcal{A}}(t))\mathcal{V}(t-1)$ can be computed in $O(kz(t) + k^2(m+n))$ time.

*Proof.*

$$\begin{aligned}
&(\mathcal{A}(t) - \hat{\mathcal{A}}(t))\mathcal{V}(t-1) \\
&= (\mathcal{A}(t) - \mathcal{U}(t-1)\mathcal{W}(t-s)\mathcal{V}(t-1)^T)\mathcal{V}(t-1) \\
&= \mathcal{A}(t)\mathcal{V}(t-1) - \mathcal{U}(t-1)\mathcal{W}(t-s)(\mathcal{V}(t-1)^T\mathcal{V}(t-1)).
\end{aligned}$$

Performing the $\mathcal{V}(t-1)^T\mathcal{V}(t-1)$ multiplication produces a $k \times k$ matrix, so the subsequent multiplications are fast. Specifically, performing $\mathcal{V}(t-1)^T\mathcal{V}(t-1)$ takes $O(nk^2)$ time, while multiplying it by $\mathcal{U}(t-1)$ takes $O(mk^2)$ time. $\mathcal{A}(t)\mathcal{V}(t-1)$ requires $O(kz(t)+nk)$ time, which add up to give $O(kz(t) + k^2(m+n))$. ■

Letting $z = \sum_{t=1}^{r} z(t)$ be the number of nonzeroes:

**Lemma 8.2**

Algorithm 8.1 is $O(kz + rk^2(m+n))$.

*Proof.* By Lemma 8.1, lines 4 and 5 take $O(kz(t)+k^2(m+n))$ time. Lines 7 to 12 are $O(m+n)$, so the inner loop (line 6) is $O(k(m+n))$. For the outer loop, summing $O(kz(t)+k^2(m+n))$ over $t = 1, \cdots, r$ gives $O(kz + rk^2(m+n))$. ■

### 8.4.4 Forecasting

Given any $t > r$, we forecast $\mathcal{A}(t)$ using the most recent $\mathcal{U}$ and $\mathcal{V}$ (i.e. $\mathcal{U}(r)$ and $\mathcal{V}(r)$) and the $\mathcal{W}$ in the most recent season at the time corresponding to $t$ (e.g. forecasting next Monday using last Monday), i.e. $t_{\text{seas}}$ where $t_{\text{seas}} = r - (r - t \bmod s)$, where mod is the modulo operation:

$$\hat{\mathcal{A}}(t) = \mathcal{U}(r)\mathcal{W}(t_{\text{seas}})\mathcal{V}(r)^T \tag{8.6}$$

### 8.4.5 Anomaly Detection: SMF-A Algorithm

Having fit the above model, how do we identify anomalies, e.g. an epidemic, or a road diversion? How anomalous is entity $i$ at time $t$? We measure anomalousness of an entity at time $t$ is by its **residuals**: i.e. the difference between its observed data at time $t$, and our model's fitted values. If a large anomaly occurred at time $t$, this difference will be large.

Define the fitted values as $\tilde{\mathcal{A}}(t) = \mathcal{U}(t)\mathcal{W}(t)\mathcal{V}(t)^T$. Then for any entity $i$ in the first mode (i.e. row $i$), its anomalousness is the sum of squared differences between the data and the fitted values:

**Definition 8.1: Row Anomalousness**

$$\text{Anom}_i(t) = \sum_{j=1}^{n} (\mathcal{A}_{ij}(t) - \tilde{\mathcal{A}}_{ij}(t))^2 \tag{8.7}$$

Anomalousness along the second mode is analogous.

Eq. (8.7) is infeasible to compute directly as $\tilde{\mathcal{A}}(t)$ is a dense $m \times n$ matrix. We now show that Eq. (8.7) can be computed more efficiently: to do this, we first rewrite $\text{Anom}_i(t)$ into a form such that the slowest part of its computation can be precomputed and then re-used when computing $\text{Anom}_i(t)$ for each $i$. Across $m$ entities, this provides large savings (up to a factor of $m$). In the following, we suppress the '$t$' notation since all terms are taken at time $t$.

**Lemma 8.3**

An equivalent, faster to compute form is:

$$\begin{aligned}\text{Anom}_i = &\mathcal{U}(i,:)\mathcal{W}\mathcal{V}^T\mathcal{V}\mathcal{W}\mathcal{U}(i,:)^T \\ &+ \sum_{i,j:\mathcal{A}_{ij}>0}\left((\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 - \tilde{\mathcal{A}}_{ij}^2\right).\end{aligned}$$

*Proof.* Note that $(\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 = \tilde{\mathcal{A}}_{ij}^2$ when $\mathcal{A}_{ij}=0$. Hence:

$$\begin{aligned}\text{Anom}_i &= \sum_{j=1}^{n}(\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 \\ &= \sum_{j:\mathcal{A}_{ij}=0}\tilde{\mathcal{A}}_{ij}^2 + \sum_{j:\mathcal{A}_{ij}>0}(\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 \\ &= \sum_{j=1}^{n}\tilde{\mathcal{A}}_{ij}^2 + \sum_{j:\mathcal{A}_{ij}>0}\left((\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 - \tilde{\mathcal{A}}_{ij}^2\right) \\ &= \|\mathcal{U}(i,:)\mathcal{W}\mathcal{V}^T\|_2^2 + \sum_{j:\mathcal{A}_{ij}>0}\left((\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 - \tilde{\mathcal{A}}_{ij}^2\right). \\ &= \mathcal{U}(i,:)\mathcal{W}\mathcal{V}^T\mathcal{V}\mathcal{W}\mathcal{U}(i,:)^T \\ &\quad + \sum_{j:\mathcal{A}_{ij}>0}\left((\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 - \tilde{\mathcal{A}}_{ij}^2\right).\end{aligned}$$

■

The key point is that $\mathcal{W}\mathcal{V}^T\mathcal{V}\mathcal{W}$ can be computed once, then re-used for all $i$, greatly reducing runtime:

**Lemma 8.4**

Computing $\text{Anom}_i$ for all $i$ is $O(kz(t)+k^2(m+n))$.

*Proof.* Computing $\mathcal{W}\mathcal{V}^T\mathcal{V}\mathcal{W}$ takes $O(mk^2)$ time. Then, computing $\mathcal{U}(i,:)\mathcal{W}\mathcal{V}^T\mathcal{V}\mathcal{W}\mathcal{U}(i,:)^T$ takes $O(k^2)$ for each $i$, thus $O(nk^2)$ overall. Computing the next term $\sum_{j:\mathcal{A}_{ij}>0}\left((\mathcal{A}_{ij}-\tilde{\mathcal{A}}_{ij})^2 - \tilde{\mathcal{A}}_{ij}^2\right)$ for every row $i$ takes $O(z(t)k)$ time, since computing $\tilde{\mathcal{A}}_{ij}$ for each nonzero $\mathcal{A}_{ij}$ takes $O(k)$. Thus the total runtime is $O(kz(t)+k^2(m+n))$. ■

### Identifying Anomalous Events

Given the $\text{Anom}_i(t)$ scores, how do we identify when an anomaly occurred? One way would be to sum $\text{Anom}_i(t)$ over entities, and plot the resulting time series. However, some entities have much higher natural variation and thus larger typical values of $\text{Anom}_i(t)$ than others, and summing in this way would drown out other true anomalies. Hence, we instead use a 'majority vote' approach that aims for both accuracy and interpretability: intuitively, time points with much higher $\text{Anom}_i(t)$ scores than the next highest time point are particularly suspicious.

**Definition 8.2: Weighted Vote**

Each entity $i$ votes for time point $t_i^{(1)} = \arg\max_t \text{Anom}_i(t)$, and the **weight** of this vote is $\text{Anom}_i(t_i^{(1)}) - \text{Anom}_i(t_i^{(2)})$, where $t_i^{(2)}$ is the time of the next highest $\text{Anom}_i(t)$.

We repeat this for each entity $i$. Then, the final anomalousness of each time point is the sum of the weighted votes given to it. This allows for interpretability: 1) the set of time points with at least 1 vote acts as a restricted set that practitioners can focus their attention on. 2) Each time point in this set has an 'explanation' in the form of the entities that voted for it. Hence, a practitioner can examine whether this entity and time point are truly anomalous.

## 8.5 Experiments

We design experiments to answer the following:

- **Q1. Accuracy:** how accurately does SMF forecast?
- **Q2. Scalability:** how does it scale?
- **Q3. Real-World Effectiveness:** does it provide meaningful components and anomalies in real data?

Our code and datasets are publicly available at `www.andrew.cmu.edu/user/bhooi/code`. We implement our algorithms in Matlab; experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM, running OS X 10.11.2. Dataset details are in Table 8.3. `Taxi` data points are hourly, with weekly periodicity ($s = 168$). `Disease` data points are weekly, with yearly periodicity ($s = 52$).

### Baselines

Our baselines are static approaches 1) SVD and 2) NMF; the seasonal approaches 3) Fold [ENLSS16] and 4) CPHW [DKA11], and 5) TSVDCWT (Truncated SVD with Collapsed Weighted Tensors) [DKA11], a dynamic (but nonseasonal) approach. [Kor10, DKM15] are also dynamic approaches, but are designed for ratings (e.g. 1 to 5 stars) and do not work in our case. For fair comparison, we use $k = 15$ components for all algorithms. Since our algorithm uses nonnegative components, we also use nonnegative CPD for the Folding and CPHW baselines.

| | Rows | Columns | Time points | Nonzero entries |
|---|---|---|---|---|
| `Taxi` | 2167 | 2167 | 2184 | 28.5M |
| `Disease` | 39 | 50 | 2601 | 0.5M |
| `Synthetic` | 5000 | 5000 | 5000 | 31M |

Table 8.3: Datasets used in our experiments.

### 8.5.1 Q1: Forecasting Accuracy

We evaluate SMF compared to baselines on `Taxi` and `Disease`. Each algorithm takes the first $r_{train}$ time steps and forecasts the next $r_{test}$. This is repeated for multiple values of $r_{train}$, and the results are averaged: for `Taxi`, we use $r_{train} = 1600, 1800, 2000$ and $r_{test} = 100$. For `Disease`, we use $r_{train} = 1000, 1500, 2000$ and $r_{test} = 500$.

#### Metrics

We use 1) **RMSE**; 2) Since RMSE values in large and sparse matrices are hard to interpret, **Time-series RMSE** aims to more directly answer questions like: 'how many taxi rides will happen each day from Brooklyn to Manhattan,' forecasting a subset of the matrix rows and columns, as a time-series. Moreover, we want this forecast to be accurate for any such subsets. Hence, in time-series RMSE, we select a random subset of rows and columns (each with probability $1/2$). Each algorithm compute a time series of its forecasted number of events within this subset of rows and columns, and compares this to the true time series using RMSE. We average this result over $10$ such random subsets.

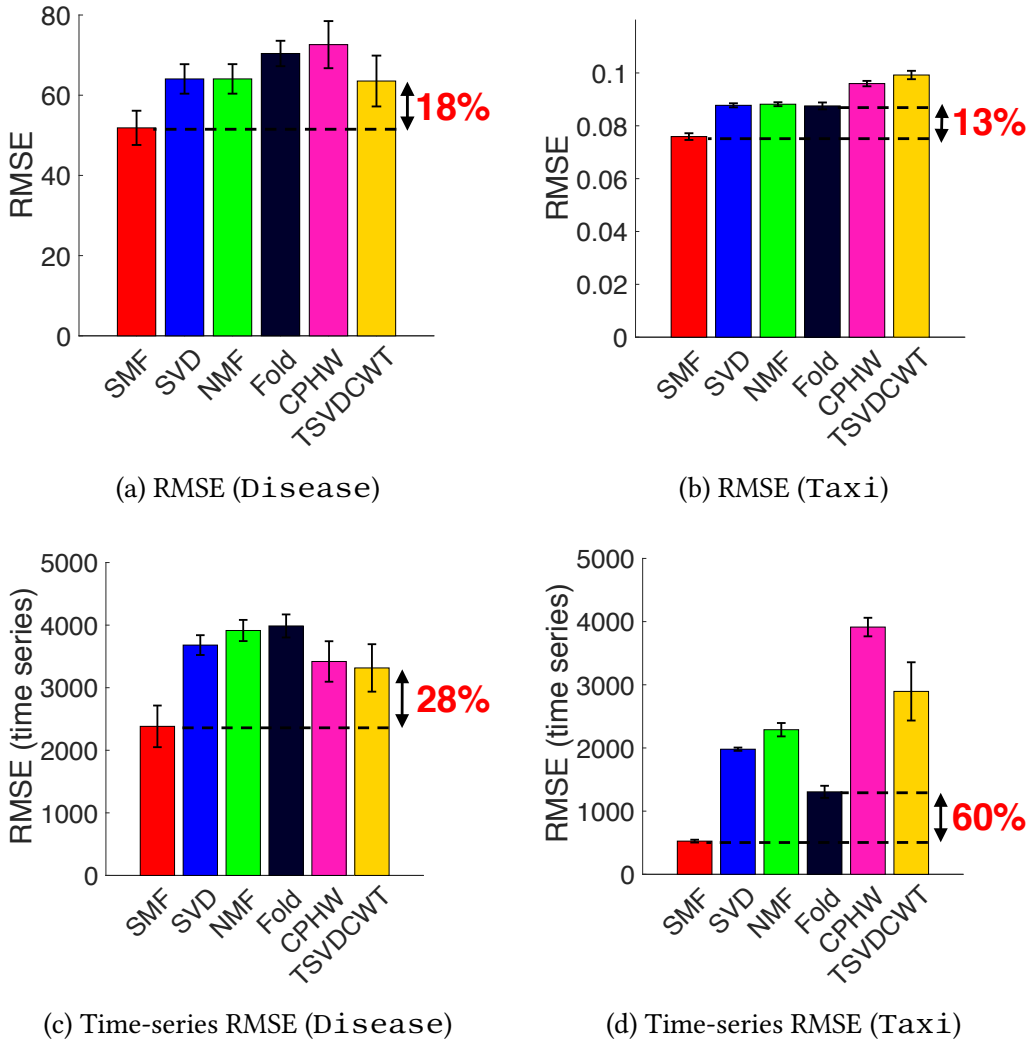
As seen in Figure 8.3, SMF outperforms baselines in accuracy. The baselines cannot capture changes in the components, or seasonal pattern, over time. Fairly large changes happen over time (e.g. see Section 8.5.3), so this causes high error.

### 8.5.2 Q2: Scalability

#### Computation time

We use a $5000 \times 5000$ matrix for $5000$ timesteps, with 200M tuples generated from a realistic power-law slice sum distribution in the first two modes (with exponent fitted to the `Taxi` dataset), and a uniform distribution in the temporal mode. After combining overlaps, there are 31M nonzeroes. For Figure 8.4, we subsample the first mode in $(1000, \cdots, 5000)$, and plot time taken against size. Each trial is averaged over $4$ repetitions. Among the seasonal baselines (CPHW and Fold), CPHW is slowest as it performs CPD on the entire tensor. Fold performs CPD on a shrunken (folded) tensor. SMF is much faster than these, requiring time comparable to NMF. Note that SVD and NMF ignore temporal information completely, operating on a static matrix, and are expected to be fast.

In an online setting where we require incremental results, the offline methods would need to be re-run for each time step, while SMF would not. Figure 8.4 does not take this into account, and runs each algorithm on the whole dataset. In such an online setting, the speedup of SMF




(a) RMSE (`Disease`)

(b) RMSE (`Taxi`)

(c) Time-series RMSE (`Disease`)

(d) Time-series RMSE (`Taxi`)

Figure 8.3: (a) **SMF outperforms baselines in accuracy:** SMF has 13% to 60% lower error than the best performing baseline. Error bars indicate one standard deviation.

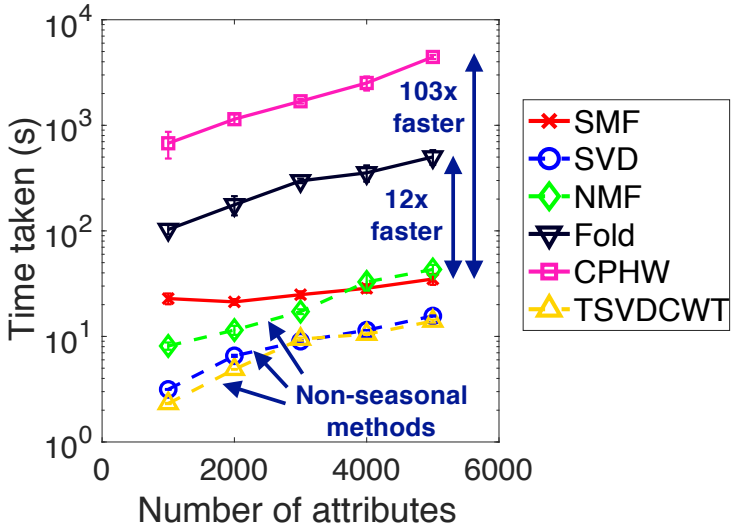
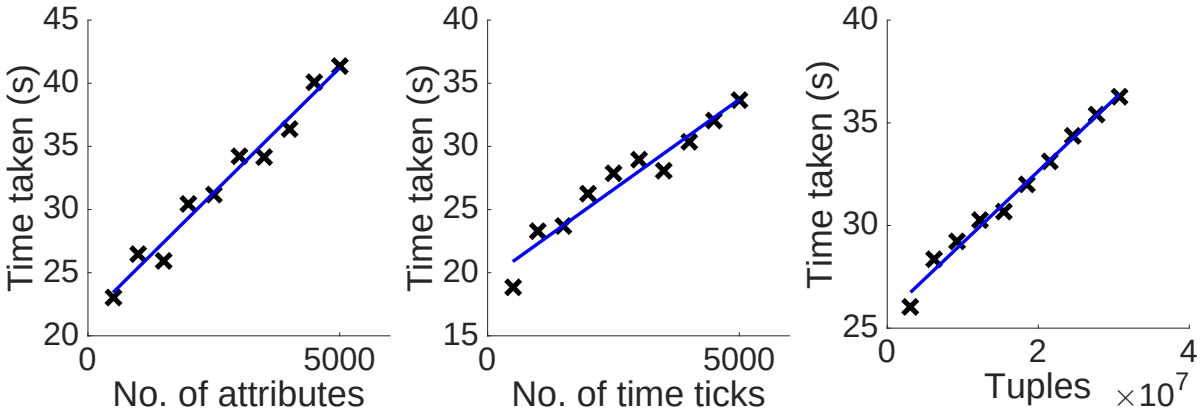



Figure 8.4: **SMF is fast:** it outperforms seasonal baselines, Fold and CPHW. Error bars (small) indicate one standard deviation.

over CPHW and Fold would be much greater. Figure 9.6 shows that SMF scales linearly in attributes (a), timesteps (b), and entries (c).



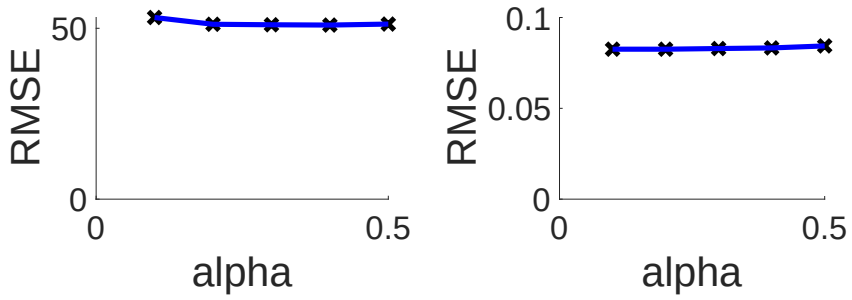

(a) Attributes (b) Time ticks (c) Nonzero entries

Figure 8.5: (a) **SMF scales linearly**.

### Parameter Selection

We select $\alpha$ using cross-validation between $\{0.1, \cdots, 0.5\}$, producing $0.3$ and $0.1$ on `Disease` and `Taxi`. The period $s$ often can be deduced from domain knowledge, but if needed, it can also be estimated by cross validation between a few reasonable candidates (daily, weekly, yearly, etc.). Figure 8.6 shows that SMF is insensitive to the hyperparameter $\alpha$, and performs well in all cases.

### 8.5.3 Q3: Real-World Effectiveness




(a) `Disease` (b) `Taxi`

Figure 8.6: **SMF is insensitive to parameter values.**

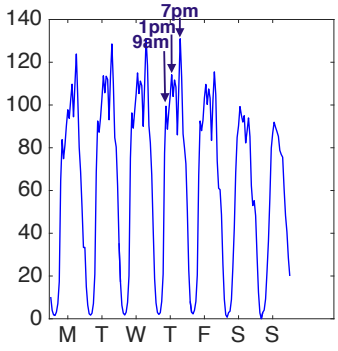
We now show that SMF provides useful and interpretable results on the `Taxi` dataset. A similar (but Manhattan-only) dataset was studied by [ENLSS16]. Figure 8.7 shows the results for 3 components, one per row. The first is concentrated around Central Park and the nearby museums, and likely to represent tourism. Its peaks coincide with mealtimes (9am, 1pm, 7pm). The second component peaks at 8-9am on weekdays, and is concentrated on the major railway stations and airports, and likely represents commuting trips, particularly in the morning 'rush-hour.' The third component peaks around Friday 10pm and Saturday midnight. This component is concentrated around southwest Manhattan, an area with a large number of bars and restaurants, suggesting entertainment related trips. In summary, our model finds meaningful seasonal components that give more insight than static approaches.

#### Drifting components

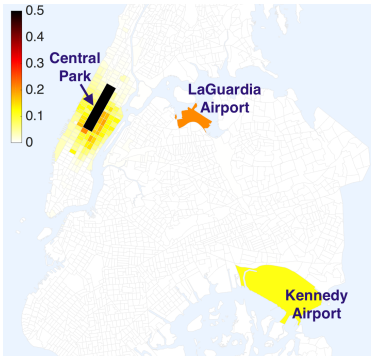
An advantage of SMF is that it allows components to drift. We use this to find meaningful patterns in the `Taxi` dataset. In New York City, prior to 2013, most pick-ups by traditional 'yellow' taxis occurred in Manhattan or at airports, resulting in low access to taxis for people living in outer areas ('boroughs') [tlc13]. In 2013, the 'green' taxi program was introduced: these pick up passengers only in outer boroughs, except at airports. Did the green taxi program improve access to taxis in the outer boroughs, and what type of trips were affected?

Figure 8.8a plots the fraction of each component that lies within Brooklyn (an outer borough). Only the red ('commute') components shows clear movement towards Brooklyn, increasing by 53%. This suggests that more and more commute-related taxi trips are departing from Brooklyn, while no such change occurs for the other two components. This supports the claim of a shift towards outer boroughs, and further reveals what type of taxi trips were most affected: commute-related trips.

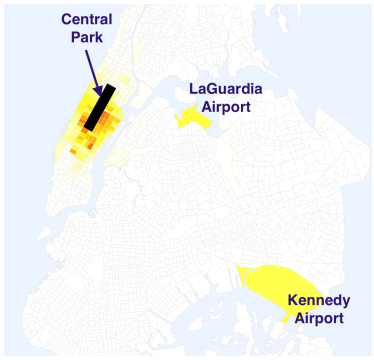
Did this change occur because of the green taxi program? We extract the top 20 locations in Brooklyn with the largest values of the 'commute' component. In these locations, Figure 8.8b plots the fraction of green and yellow taxis over time: green taxi rides rose significantly, while the yellow taxi rides decreased slightly. This suggests that green taxis were increasingly



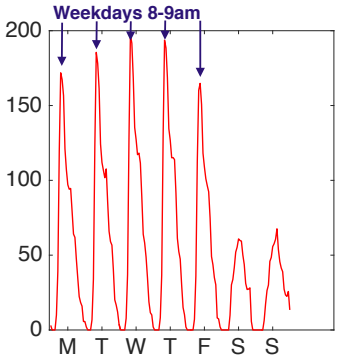

(a) 'Tourism': weekly pattern




(b) 'Tourism': pickup location

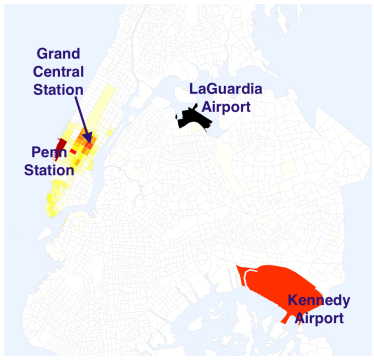



(c) 'Tourism': dropoff location

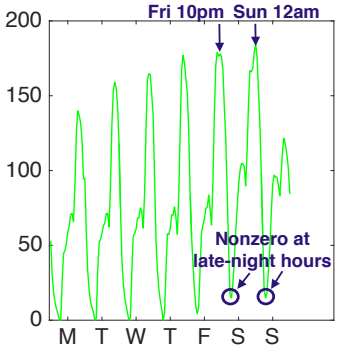



(d) 'Commute': weekly pattern

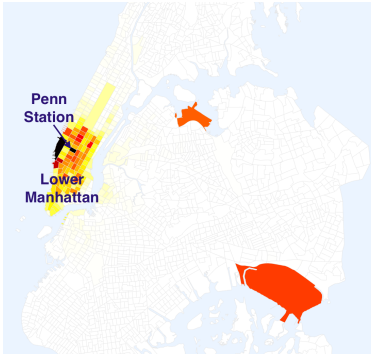

(e) 'Commute': pickup location



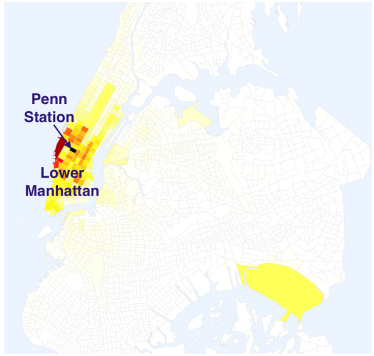

(f) 'Commute': dropoff location




(g) 'Entertainment': weekly pattern



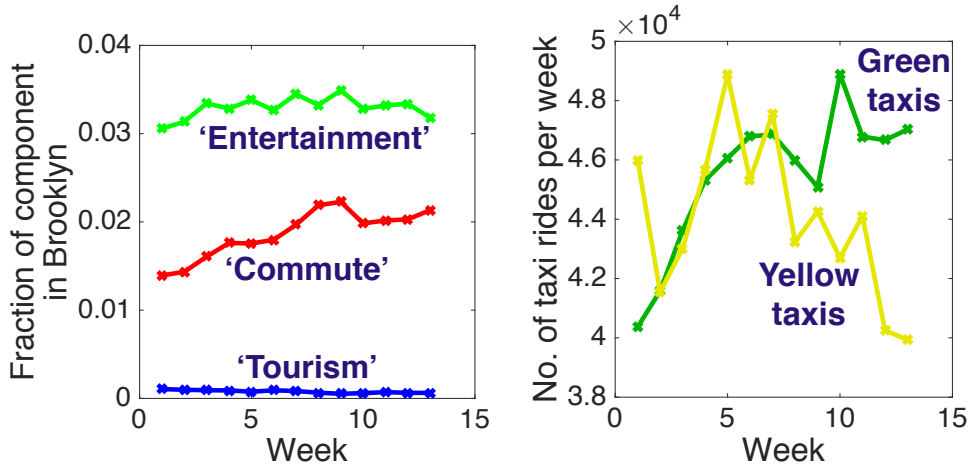

(h) 'Entertainment': pickup location




(i) 'Entertainment': dropoff location

Figure 8.7: (a) **SMF provides interpretable results with seasonal information:** the three components in the `Taxi` dataset correspond roughly to tourism-related trips (near Central Park and museums), morning rush-hour trips (airports and train stations), and entertainment (restaurants and bars).

adopted, and since green taxis only pick up passengers in outer boroughs, this supports the




(a) Shift towards Brooklyn

(b) Increase in green taxis

Figure 8.8: **SMF allows components to change over time:** (a) the 'commute' component (red) shifts toward Brooklyn, increasing by 53% in the fraction of the component in Brooklyn. (b) This can be explained by an increase in green taxis in the 'commute' component.

claim that the rise of green taxis contributed to the shift towards Brooklyn. Thus, allowing the components to change over time reveals useful new information about the dataset.

### 8.5.4 Anomaly Detection

We now evaluate the anomaly detection accuracy of SMF-A. Starting with the `Taxi` dataset, we inject 100 anomalies of two types: 'add' anomalies represent an increase in activity (e.g. a major festival), while 'scramble' anomalies represent changes in behavior (e.g. a traffic accident redirecting traffic). For the 50 'add' anomalies, we select a random 50 rows and columns, and add 500 taxi trips to this block, distributed according the same power-law distribution as Section 8.5.2. For the 50 'scramble' anomalies, we select the 200 highest degree rows and columns, and randomly permute the rows and columns, changing the position of entries in this submatrix. We compare SMF-A to DenseAlert [SHKF17], a recent anomaly detection algorithm based on dense submatrix detection. The thresholds plotted in Figures 8.9 and 8.1c are 3 standard deviations from the mean (in log-space).

**Results:** Table 8.4 shows the precision at 100 of both methods on 'add' and 'scramble' anomalies, and Figure 8.9 shows the output of SMF-A. 100 is the true number of anomalies, so the number of false positives and negatives are equal. SMF-A is much more accurate than DenseAlert, and faster. This is because SMF-A takes into account differences from expected behavior, while DenseAlert only considers density. Moreover, DenseAlert cannot catch 'scramble' anomalies as it detects high activity, while SMF-A can detect any type of deviation from expected behavior.

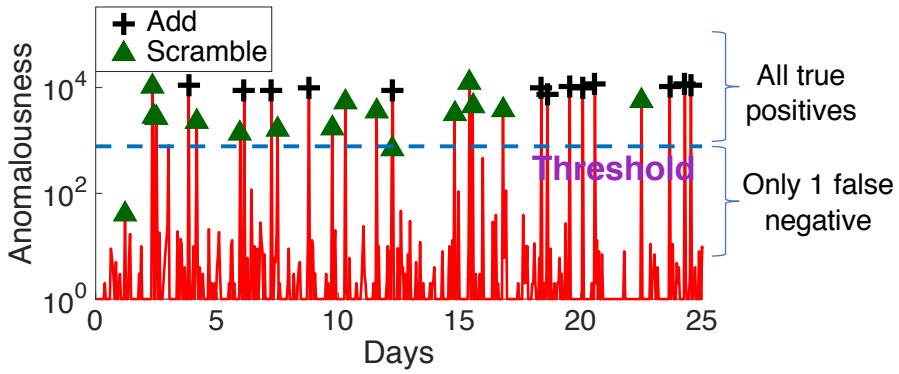



Figure 8.9: **SMF-A detects multiple types of anomalies:** it catches anomalies which add taxi trips ('add') or permute a subset of locations ('scramble'). We plot only the first 25 days, for visibility.

| | Prec. (add) | Prec. (scramble) | Runtime (sec.) |
|---|---|---|---|
| SMF-A | **1.00** | **0.86** | **304.94** |
| DENSEALERT | 0.06 | 0.06 | 1247.13 |

Table 8.4: **SMF-A outperforms baselines in anomaly detection:** precision of SMF-A in catching injected anomalies which add taxi trips ('add') or randomly reorder a subset of locations ('scramble').

Figure 8.1c shows our results of SMF-A on the `Disease` dataset. Two epidemics stand out, both of which have been reported in the medical literature: a large influenza outbreak in 1928 in Northeast US [Col30], and a measles epidemic in New York in 1946 [PBS47].

## 8.6 Conclusion

We propose SMF, a drift and seasonality aware, online matrix factorization algorithm, and SMF-A, a fast anomaly detection algorithm. In contrast to existing methods (Fold and CPHW), our model uses smoothly varying components $\mathbf{u}$ and $\mathbf{v}$, and seasonally varying multiplers $w$ to model seasonality. Our contributions are as follows:

- **Model:** we propose a novel matrix factorization model incorporating seasonal patterns and drift, and an online algorithm for fitting this model.
- **Effectiveness:** in experiments, SMF has lower forecasting error than baselines by 13% to 60% (Figure 8.1a), and provides interpretable results in case studies on real data.
- **Scalability:** SMF is online, and scales linearly (Figure 8.1b). In experiments, it was 12 to 103 times faster than seasonal baselines.

- **Fast Anomaly Detection:** we propose SMF-A for detecting anomalies (Figure 8.1c) in a computationally feasible way, without forecasting every possible observation in the matrix.

# Part III

# Graphs with Sensors

# Overview: Graphs with Sensors

*Given time-varying data from sensors arranged in a graph, how can we detect unusual events?*

In this part, we consider data from sensors placed on a static graph, where each sensor provides time series data: for example, electrical sensors placed on nodes of the power grid, where we would like to automatically detect anomalous events, such as due to the failure of electrical components, or the similar problem of detecting traffic accidents using traffic speed sensors. Our first ChangeDAR approach uses a **localized change detection** approach, aiming to detect sudden changes that are experienced by a group of nearby nodes, forming a connected subgraph. Our next GridWatch approach considers a domain-aware approach targeted at power grid applications specifically, aiming to find **power grid anomalies** involving the failure of electrical lines or other components. We also propose an approach for placing a given budget of sensors, in order to more effectively detect anomalies.

# Chapter 9

# ChangeDAR: Localized Anomaly Detection in Graphs with Sensors

Chapter based on work published in CIKM18 [PDF].

In this chapter, we consider event detection in graphs with sensors. Given electrical sensors placed on the power grid, how can we automatically determine when electrical components (e.g. power lines) fail? Or, given traffic sensors which measure the speed of vehicles passing over them, how can we determine when traffic accidents occur? Both these problems involve detecting change points in a set of sensors on the nodes or edges of a graph. To this end, we propose ChangeDAR (Change Detection And Resolution), which detects changes in an online manner, and reports *when* and *where* the change occurred in the graph, with theoretical guarantees.

## 9.1 Introduction

How do we detect change points using sensors placed on a subset of the nodes or edges of a graph? In the power grid setting, this question is motivated by the need to quickly detect electrical component failures using sensor data. Such failures can occur due to severe weather, human or equipment failure, or even adversarial intrusion, and have major costs: estimates [Ami11] suggest that reducing outages in the U.S. grid could save $\$49$ billion per year and reduce emissions by $12$ to $18\%$. To achieve this goal, there is a need to use sensor data to quickly identify in real-time when parts of the grid fail, so as to quickly respond to the problem. Similarly, in the traffic setting, a large network of traffic speed detectors spans freeway systems in major metropolitan areas - our goal is to use them to automatically detect notable changes, such as traffic accidents.

In both cases, the changes that we wish to detect are highly **localized**, both in time and with respect to network structure: power line failures affect a localized set of power lines due to redirection of current through neighboring lines, and the same holds for traffic accidents, which slow traffic in neighboring roads.

An additional challenge is that we want to detect change points in an **online** manner: both power grid and traffic data are high-volume and received in real time, since the data comes from sensors which are continuously monitoring. This motivates us to develop fast methods that work in this online setting. When each new data point is received, the algorithm should update itself efficiently - for our algorithm, each update requires constant time, and bounded memory, regardless of the length of the stream.

Thus, our goal is an online, localized change detection algorithm:

**Informal Problem 9.1: Online Localized Change Detection**

- **Given** a fixed-topology graph $\mathcal{G}$ (e.g. road network or power grid), and time-series sensor values on a subset of the nodes and edges of the graph, received in a streaming manner,
- **Find** change points incrementally, each consisting of a time $t$ and a localized region of the graph where the change occurred.

Change detection in time-series [SK74, CF15] and graphs [DLJL09, SFPY07] has been studied extensively (expanded in Section 9.2). Our work differs in two key aspects. Most work focuses on dynamically evolving graphs with changing nodes or edges [DLJL09, SFPY07]. In our case the graph topology is fixed, and only sensor values on nodes and edges are evolving. Second, most work in this area finds the time of a change, without being able to localize the change to parts of the graph.

Figure 9.1 shows an example of our method on traffic data. Given a road graph and the traffic speed over various sensors (indicated by the nodes in the left plots), our method detected a traffic accident. The accident caused a change point (drops in vehicle speed) over a localized set of sensors (red nodes in the left plots). Our method exploits the localized nature of this change to accurately detect it, and outputs both the change time and localization.

Our contributions are as follows:

1. **Algorithm:** We propose novel information theoretic optimization objectives for 1) scoring and 2) detecting localized changes, and propose two algorithms, ChangeDAR-S and ChangeDAR-D respectively, to optimize them.
2. **Theoretical Guarantees:** We show that both algorithms provide constant-factor approximation guarantees (Theorems 9.2 and 9.4).
3. **Effectiveness:** Our algorithms detect traffic accidents and power line failures in a power grid with 75% higher F-measure than comparable baselines in experiments.
4. **Scalability:** Our full algorithm is online and near-linear in the graph size and the number of time ticks (Figure 9.6).

**Reproducibility:** our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/changedar/`.

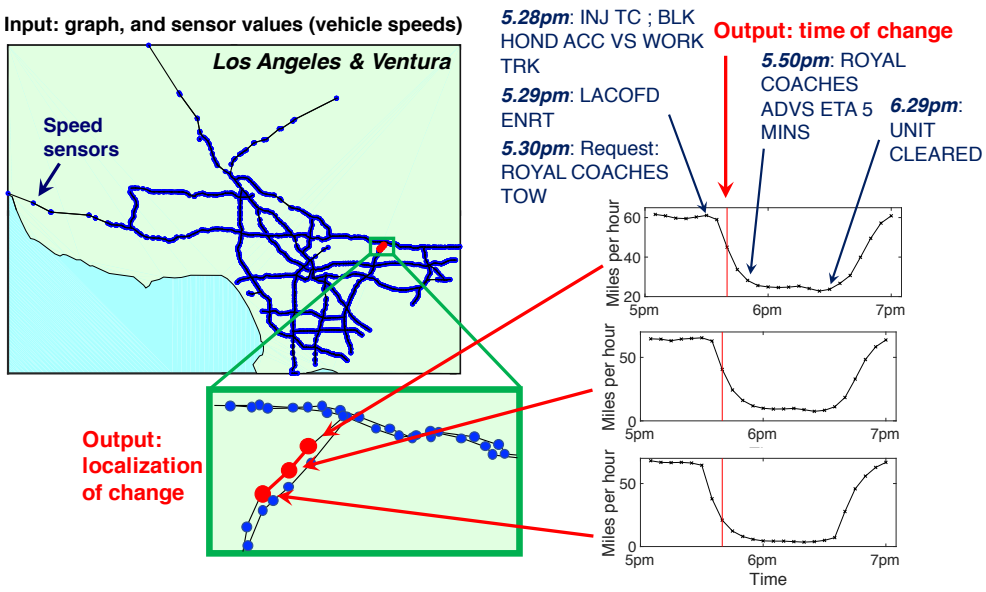



Figure 9.1: **ChangeDAR correctly locates a traffic accident** (red): the 3 time-series on the right show drops in average speed at 3 consecutive points on a highway. ChangeDAR outputs the time (red vertical lines) and location (red nodes) of the change, and we verify the accident against the traffic report by the California Highway Patrol (blue text at top-right) [pem18].

## 9.2 Related Work

**Multivariate Change Detection** [AC17] reviews time-series change detection methods. Multivariate change detection methods aim to segment a time-series into two or more regimes, such as greedy binary segmentation [SK74, CF15], or slower but exact dynamic programming (DP) [JSB+05]. PELT [KFE12] applies a pruning step to perform DP in linear time. Other approaches include the Group Fused Lasso (GFL) [BV11], Bayesian change detection [AM07], nonparametric methods [LYCS13, MJ14], support vector machines [DDD05], and neural networks [LLJ02]. These methods do not consider graph structure, and hence do not detect localized changes.

**Change Detection in Dynamic Graphs** Change detection methods for dynamic graphs, or graphs which change over time, have been proposed [DLJL09, SFPY07, PC15]. These include Bayesian methods [PC15], and distance-based methods, which define a distance metric on graphs: based on diameter [GKW06], node or edge weights [PDGM10, Pin05], connectivity [KVF13], or subgraphs [MBS13]. Except for [PDGM10, Pin05], these do not apply to our setting, as they assume a changing graph over time, while we have a fixed graph with sensor values changing over time. For [PDGM10, Pin05], we apply them by treating our sensors as node or edge weights. While they do not perform localization, we still use them as baselines in our experiments.

**Graph-based Anomaly Detection and Scan Statistics** A number of methods consider anomaly detection using graph scan statistics [MBR+13, CN14, MSN13, RAGT14], which search for a highly anomalous area in static and temporal graphs. [CCV17] uses coloring to efficiently optimize graph scan statistics, and [CBVD18] incorporates uncertainty in the observed data. [SSR13] defines the Spatial Scan Statistic while [SRS16] defines the Graph Fourier Scan Statistic (GFSS), which quantify the spatial locality of a signal. These methods focus on time intervals containing unusual activity, while our goal is change detection, which involves a shift between 'regimes' (i.e. generative processes) before and after the change. In the power grid case, our goal is to detect equipment failures, which involve a difference in regimes before and after the failure, but not necessarily a temporal 'burst' of activity at any time.

Table 10.1 summarizes existing change point detection methods. ChangeDAR differs from existing methods in that it detects localized change points using an online algorithm, and provides theoretical guarantees.

Table 9.1: Comparison of change detection approaches applicable to sensor data on a graph. The last 2 rows refer to automatically detecting the number of changes, and how many nodes and edges each change affects, respectively.

| *Property* | PELT [KFE12] | GFL [BV11] | WeightDist [Pin05] | VertexRank [PDGM10] | **ChangeDAR** |
|---|---|---|---|---|---|
| **Graph-based** | | | | ✓ | ✔ |
| **Localization** | | | | | ✔ |
| **Online Algorithm** | | | | ✓ | ✔ |
| **Provable Guarantees** | ✓ | ✓ | | | ✔ |
| **Auto Detect # of Changes** | | | ✓ | ✓ | ✔ |
| **Auto Detect Size of Change** | | | | | ✔ |

## 9.3 Background

Before delving into the details of our problem statement and proposed algorithms, we briefly review two graph-theoretic concepts that we will make use of later, namely Prize-Collecting Steiner Tree (PCST) and Maximum Weight Independent Set (MWIS).

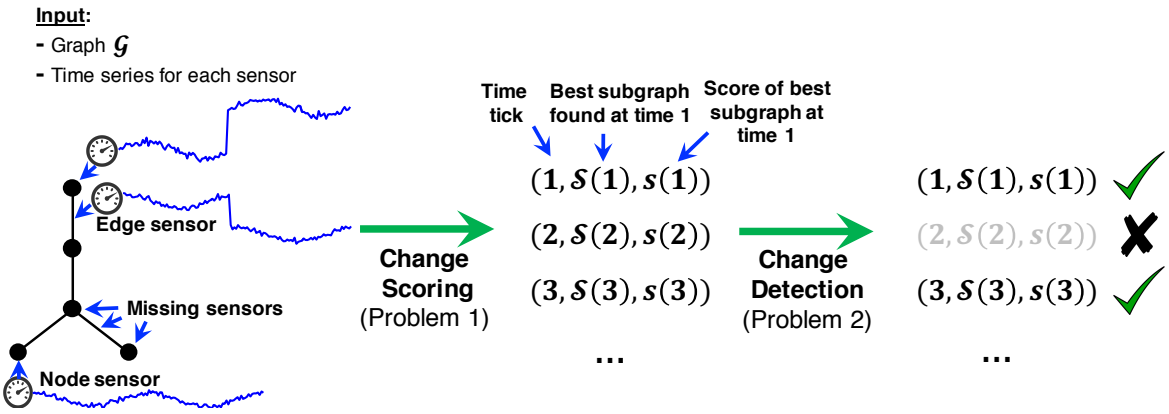



Figure 9.2: **Outline of steps:** given a graph with time-series sensors on some nodes and edges, Change Scoring selects the best subgraph where a change occurred at each time, while Change Detection selects the best subset of change points out of these.

### 9.3.1 Prize-Collecting Steiner Tree (PCST)

The prize-collecting Steiner tree problem [BGSLW93] finds a connected subgraph of a graph that maximizes total *profit* values on the subgraph nodes while minimizing *cost* of the edges in the subgraph. Intuitively, imagine nodes represent cities, and the *cost* of each edge is the cost of building a road between the two cities, and the *profit* of a node is the profit from that city joining the road network. Then PCST aims to construct a road network to maximize net profit (or minimize net cost).

- **Given** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with non-negative node profits $p(v) \; \forall \; v \in \mathcal{V}$ and non-negative edge costs $c(e) \; \forall \; e \in \mathcal{E}$;
- **Output** a connected subgraph $\mathcal{S} = (\mathcal{V}_\mathcal{S}, \mathcal{E}_\mathcal{S})$ of $\mathcal{G}$ that minimizes the profit in nodes *not* chosen, plus the total cost of edges chosen, i.e. $\sum_{v \notin \mathcal{V}_\mathcal{S}} p(v) + \sum_{e \in \mathcal{E}_\mathcal{S}} c(e)$.

[Kar72] shows that PCST is NP-hard. [BGSLW93] introduced the PCST problem and showed a 3-approximation algorithm. [GW95] proposed a $O(n^2 \log n)$ approximation algorithm with a $2 - \frac{1}{n-1}$ factor guarantee, where $n$ is the number of nodes. [HIS15] improved this to a near-linear time ($O(m \log n)$) 2-approximation algorithm. In our setting, we will use [HIS15] on a modified graph to find localized change points.

### 9.3.2 Maximum Weight Independent Set (MWIS)

The maximum weight independent set problem finds a subset of nodes of highest weight which has no edges between them.

- **Given** a graph $\mathcal{G}$ with node weights $w(v) \; \forall \; v \in \mathcal{V}$;
- **Output** a set $\mathcal{S}$ of nodes with no edges between them maximizing $\sum_{v \in \mathcal{S}} w(v)$.

In the offline setting, approximation algorithms exist [Hal04], but in the online case, [GHK+14] showed that no meaningful worst-case guarantees are possible. Hence, follow-up work considers non-worst case analysis [GHK+14, KP09].

In our setting, we study a constrained variant of the online MWIS problem that arises from our change detection problem, and show that we can obtain constant worst-case approximation guarantees, thanks to certain constraints in our setting. We then use this to obtain theoretical guarantees for multiple change detection.

[RAGT14, CCV17] use PCST for anomaly detection, but to the best of our knowledge, both PCST and online MWIS have not been used for change detection.

## 9.4 Problem

### 9.4.1 Problem Setting

Table 10.2 shows the symbols used in this chapter.

Table 9.2: Symbols and definitions

| Symbol | Interpretation |
|---|---|
| $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ | Input graph |
| $n, m$ | Number of nodes and edges respectively |
| $w$ | Window size |
| $X_v(t)$ | Sensor value at node $v$ at time $t$ |
| $X_e(t)$ | Sensor value at edge $e$ at time $t$ |
| $k$ | Number of parameters of time-series model (see Eq. (9.3)) |
| $C_F$ | Number of bits needed to store a floating point number |
| $\Delta_v(t)$ | 'Bitsave' score at node $v$ at time $t$ (see Eq. (9.1)) |
| $\Delta_e(t)$ | 'Bitsave' score at edge $e$ at time $t$ (see Eq. (9.1)) |
| $\pi(v')$ | Profit assigned to node $v'$ |
| $c(e')$ | Cost assigned to edge $e'$ |
| $\mathcal{G}_{\text{conf}}$ | Conflict graph (see Definition 9.2) |
| $r$ | Repetitions of randomized algorithm in ChangeDAR-S |

We are given an undirected graph $\mathcal{G}$ (e.g. a power grid graph) and a stream of sensor values associated with the nodes and/or edges of the graph (e.g. voltage values measured at nodes), as illustrated in Figure 9.2. Since some applications involve sensors on nodes while others involve sensors at edges (e.g. current values measured along edges), we consider a general framework that allows for both types of sensors. Some sensors may be missing: hence, if one type of sensor is not present, we can simply consider their values as missing. Define $X_v(t)$ as the sensor value at node $v$ at time $t$, and $X_e(t)$ as the sensor value at edge $e$ at time $t$.

We consider the sensor values to arrive in an **online** manner. However, intuitively, it is unrealistic to decide if a change point exists at time $t$ given only information up to time $t$, since we have no information about what comes after. Hence, we allow for a **window** of $w$ time ticks, in which the algorithm has access to the next $w$ time ticks before needing to make a decision at time $t$. In practice, this means the algorithm reports whether a change occurred at time $t$ after a lag time of $w$ time ticks.

Hence, given data up to time $t + w$, we would like the algorithm to output if a change occurred at time $t$. This can be broken down into two sub-problems: 1) change scoring and 2) change detection.

**Problem 9.1: Online Change Scoring**

- **Given** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a stream of (possibly missing) sensor values $X_v(t)$ and $X_e(t)$ for each node $v$, each edge $e$, and time tick $t = 1, 2, \cdots$:
- **Output** (in an online manner) at time $t + w$:
  - ***Scoring***: a score $s(t)$ measuring our confidence that a change occurred at time $t$;
  - ***Localization***: the best subset of nodes and edges $\mathcal{S} = \mathcal{V}_\mathcal{S} \cup \mathcal{E}_\mathcal{S}$, where $\mathcal{V}_\mathcal{S} \subseteq \mathcal{V}$ and $\mathcal{E}_\mathcal{S} \subseteq \mathcal{E}$, at which a change occurred.

The notion of the 'best' subset where a change most likely occurred, and the interpretation of the score $s(t)$, will be formalized in an information theoretic manner in Section 9.5.1.

Given these scores, the next sub-problem is to decide which of these changes actually occurred:

**Problem 9.2: Online Change Detection**

- **Given** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a stream of sensor values $X_v(t)$ and $X_e(t)$ for each node $v$, each edge $e$, and time tick $t = 1, 2, \cdots$:
- **Output** (in an online manner) at time $t + w$: whether a change occurred at time $t$, and if so, the corresponding subset $\mathcal{S}(t)$ where the change occurred.

Figure 9.2 provides an outline of the steps in our approach. We consider Change Scoring in Section 9.5 and Change Detection in Section 9.6.

## 9.5 Change Scoring: ChangeDAR-S

We now propose an algorithm for Problem 9.1, for finding the best subset $\mathcal{S}$ and score $s(t)$ at time $t$, in an online manner.

### 9.5.1 Optimization Objective

We first define our '*Bitsave*' metric, which intuitively measures how beneficial (in terms of number of bits we save) it is to add a change point at time $t$.

### Description Length Framework

First consider the simpler question of how to encode a single time-series $Y_1, \cdots, Y_n$. The Minimum Description Length (MDL) approach states that given data $Y$, we should encode it using the model $M$ that minimizes the **description length** of $Y$, which is defined as $\text{Cost}(Y) = \text{Cost}(M) + \text{Cost}(Y|M)$, where $\text{Cost}(M)$ is the number of bits needed to encode the model $M$, and $\text{Cost}(Y|M)$ is the number of bits needed to encode the data given model $M$. The full expression for these costs depends on the type of model used, which we describe as follows.

**Model Cost:** We use a flexible approach that allows for any model family to be used for encoding a time-series, e.g. Autoregression, Seasonal Autoregression etc., resulting in a vector of fitted values $(\hat{Y}_i)_{i=1}^n$. Let $k$ be the number of parameters used: e.g. if we fit an Autoregressive $AR(p)$ model, then $k = p + 1$ (the additional 1 is for the intercept). Then the model cost $\text{Cost}(M)$ is the cost of $k$ floating point values, or $k \cdot C_F$, where $C_F$ is the cost to encode a floating point number[1]. A simple default choice (which we use in our experiments) is to set $\hat{Y}_i$ as the mean $\frac{1}{n}\sum_{j=1}^n Y_j$ for all $i$, which has $k = 1$ parameter, but more complex functions can be used, particularly when seasonality is present.

**Data Cost:** For the data cost $\text{Cost}(Y|M)$, we assume that the errors follow a normal $\mathcal{N}(0, \sigma^2)$ distribution. The log probability density of the errors $Y_i - \hat{Y}_i$ under this distribution is

$$\log P(Y_1 - \hat{Y}_1, \cdots, Y_n - \hat{Y}_n) = \log \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(Y_i - \hat{Y}_i)^2}{2\sigma^2}} \tag{9.1}$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - \hat{Y}_i)^2 + \text{const.} \tag{9.2}$$

where 'const.' does not depend on the data. By the theory on Huffman coding [CT12], the cost in bits to represent some data under a given distribution is the negative log-likelihood of the data. Intuitively, the less probable a particular outcome is, the more bits we need to encode it: e.g. encoding the outcome of a coin flip requires 1 bit, but encoding that we rolled a '1' on a fair 8-sided die needs $-\log_2 \frac{1}{8} = 3$ bits.

Taking the negative of (9.2), the data cost $\text{Cost}(Y|M)$ is $\frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - \hat{Y}_i)^2$. Thus the total cost (in bits) of encoding $Y$ is:

$$\begin{aligned} \text{Cost}(Y_1, \cdots, Y_n) &= \text{Cost}(M) + \text{Cost}(Y_1, \cdots, Y_n|M) \\ &= k \cdot C_F + \frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \end{aligned} \tag{9.3}$$

Note that we are not actually making a strong requirement that the errors be normal: Eq. (9.3) simply encodes the data such that the lower the total squared error, the fewer bits we need. We are essentially minimizing squared error, with an additional penalty for the cost of the model, in bits.

[1] We use $C_F = 32$ for standard 4-byte floats.

### Computing Bitsave Scores

We now apply Eq. (9.3) to the sensor at node $v$ at time $t$. Since we are in an online setting, bits saved have to be computed with respect to a window of size $w$. Intuitively, the bits saved from adding a change at time $t$ is the cost of encoding $X_v$ from time $t-w$ to $t+w-1$, minus the same cost if we were to add a change at time $t$:

**Definition 9.1: Bitsave**

The bitsave score from adding a change at time $t$ at node $v$ is:

$$\begin{aligned}\Delta_v(t) = &\text{Cost}(X_v(t-w), \cdots, X_v(t+w-1)) \\ &- \text{Cost}(X_v(t-w), \cdots, X_v(t-1)) \\ &- \text{Cost}(X_v(t), \cdots, X_v(t+w-1))\end{aligned} \tag{9.4}$$

The bitsave score for edge $e$, $\Delta_e(t)$, are the same, except using $X_e$ instead of $X_v$. For missing sensors, we simply set their bitsave score to 0 for all $t$.

**Lemma 9.1**

A lower bound for $\Delta_v(t)$ (or $\Delta_e(t)$) is:

$$\Delta_v(t) \geq -k \cdot C_F \tag{9.5}$$

*Proof.* When adding a change point at time $t$, the total data cost cannot increase, since we are allowed to fit the same model to both sides of the change point. Meanwhile, the model cost increases by exactly $k \cdot C_F$, since adding one change point adds an additional set of model parameters, which costs $k \cdot C_F$ bits. ■

### Localized Change Scoring Objective

The notion of *localized* changes captures the fact that in many applications, changes tend to spread along the graph: e.g. power lines going down affect a connected set of lines, and traffic congestion affects a connected set of roads. To capture this, we stipulate that $\mathcal{S}$, the subgraph of nodes and edges affected by the change, must be connected in the graph.

Hence, at time $t$, we want to detect a localized set of nodes and edges which is a good change point, i.e. provides large total bits saved. Denote by $\mathcal{S} = \mathcal{V}_\mathcal{S} \cup \mathcal{E}_\mathcal{S}$ the set of nodes and edges we are searching for, where $\mathcal{V}_\mathcal{S} \subseteq \mathcal{V}$ and $\mathcal{E}_\mathcal{S} \subseteq \mathcal{E}$.

Given subgraph $\mathcal{S}$, the *total bitsave* by adding a change point at time $t$ is the sum of $\Delta$ values on $\mathcal{S}$: i.e. $\sum_{v \in \mathcal{V}_\mathcal{S}} \Delta_v(t) + \sum_{e \in \mathcal{E}_\mathcal{S}} \Delta_e(t)$. Under the MDL framework, we then need to encode $\mathcal{S}$ itself. The simplest way is to encode each of its elements individually: $\mathcal{S} \subseteq \mathcal{V} \cup \mathcal{E}$ is a subset over $m+n$ elements, so each element takes $\log(m+n)$ bits, or $|\mathcal{S}| \log(m+n)$ bits in total. In

summary, the optimization objective to find $\mathcal{S}$ is:

$$\begin{aligned} &\max_{\mathcal{S}\subseteq\mathcal{V}} \; f_t(\mathcal{S}) = \sum_{v\in\mathcal{V}_\mathcal{S}} \Delta_v(t) + \sum_{e\in\mathcal{E}_\mathcal{S}} \Delta_e(t) - |\mathcal{S}|\log(m+n) \\ &\textbf{subject to: } \mathcal{S} \text{ is connected} \end{aligned} \tag{9.6}$$

### 9.5.2 Optimization Approach

To solve (9.6) we rewrite it into an equivalent form that can be optimized using the Prize-Collecting Steiner Tree framework, which can then be solved in near-linear time with an approximation guarantee of 2.

We first construct a new, bipartite graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$: it has a node $v'$ for each $v \in \mathcal{V}$, and a node $e'$ for each $e \in \mathcal{E}$. Then, in $\mathcal{G}'$, connect $v'$ to $e'$ iff $v$ is adjacent to $e$ in the original graph $G$.

For each $v \in \mathcal{V}$, assign the node $v'$ a *profit* of $\pi(v') = \Delta_v(t) + k \cdot C_F$. Similarly, for each $e \in \mathcal{E}$, assign the node $e'$ a *profit* of $\pi(e') = \Delta_e(t) + k \cdot C_F$. Finally, assign each edge $e' \in \mathcal{E}'$ a *cost* of $c(e') = k \cdot C_F + \log(m+n)$. Intuitively, the *profits* correspond to bits saved due to the corresponding sensor, while the *costs* correspond to a model complexity penalty in bits due to encoding each additional node $(\log(m+n))$, as well as the additional set of model parameters due to the added change point $(k \cdot C_F)$.

We will show that our optimization objective (9.6) is equivalent to the Prize-Collecting Steiner Tree objective in the new graph $\mathcal{G}'$.

$$\begin{aligned} &\min_{\mathcal{S}'\subseteq\mathcal{V}'} \; f'_t(\mathcal{S}') = \sum_{v'\notin\mathcal{V}_{\mathcal{S}'}} \pi(v') + \sum_{e'\in\mathcal{E}_{\mathcal{S}'}} c(e') \\ &\textbf{subject to: } \mathcal{S}' \text{ is connected} \end{aligned} \tag{9.7}$$

Let $\mathcal{S} \subseteq \mathcal{V} \cup \mathcal{E}$, and define the corresponding $\mathcal{S}'$ to include all nodes in $\mathcal{G}'$ corresponding to nodes and edges in $\mathcal{S}$, i.e. $\mathcal{S}' = \{v' : v \in \mathcal{V}_\mathcal{S}\} \cup \{e' : e \in \mathcal{E}_\mathcal{S}\}$.

**Lemma 9.2**

The objectives of (9.6) and (9.7) are equivalent:

$$f'_t(\mathcal{S}') = -f_t(\mathcal{S}) + C \tag{9.8}$$

where $C = \sum_{v'\in\mathcal{V}'} \pi(v') - k \cdot C_F - \log(m+n)$ is constant w.r.t. $\mathcal{S}$ and $\mathcal{S}'$.

*Proof.* $\mathcal{S}'$ has 1 node for each node or edge of $\mathcal{S}$, so it has $|\mathcal{S}|$ nodes. Moreover, any optimal $\mathcal{S}'$ must be a tree since any edge in a cycle in $\mathcal{S}'$ only increases costs without adding any profits.

Hence, $\mathcal{S}'$ has $|\mathcal{S}| - 1$ edges. Then:

$$\begin{aligned}
f'_t(\mathcal{S}') &= \sum_{v' \notin \mathcal{V}_{\mathcal{S}'}} \pi(v') + \sum_{e' \in \mathcal{E}_{\mathcal{S}'}} c(e') \\
&= \sum_{v' \in \mathcal{V}'} \pi(v') - \sum_{v' \in \mathcal{V}_{\mathcal{S}'}} \pi(v') + (|\mathcal{S}'| - 1)(\log(m+n) + k \cdot C_F) \\
&= C - \sum_{v' \in \mathcal{V}_{\mathcal{S}'}} \pi(v') + |\mathcal{S}|(\log(m+n) + k \cdot C_F) \\
&= C - (\sum_{v \in \mathcal{V}_{\mathcal{S}}} \Delta_v(t) + \sum_{e \in \mathcal{E}_{\mathcal{S}}} \Delta_e(t) - |\mathcal{S}| \log(m+n)) \\
&= C - f_t(\mathcal{S}).
\end{aligned}$$

■

Moreover, $\mathcal{S}'$ is connected if and only if its nodes form a single connected component, which is equivalent to $\mathcal{S}$ being connected. In combination with Lemma 9.2, this implies that minimizing $f'_t(\mathcal{S}')$ is equivalent to maximizing $f_t(\mathcal{S})$.

Finally, note that by Lemma 9.1, we have $\Delta_v(t) \geq -k \cdot C_F$ and $\Delta_e(t) \geq -k \cdot C_F$, which implies that $\pi(v') \geq 0 \, \forall \, v' \in \mathcal{V}_{\mathcal{S}'}$, so all profits are nonnegative. The costs, $\log(m+n)+k \cdot C_F$ are also nonnegative. Hence we can solve (9.7) in near-linear time with approximation guarantees of 2 using algorithms for Prize-Collecting Steiner Tree (as reviewed in Section 9.3.1).

Algorithm 9.1 gives the full ChangeDAR-S algorithm. We first convert $\mathcal{G}$ to the bipartite $\mathcal{G}'$ (Line 1). Then for each $t$, we compute the PCST profits (Lines 4 to 5) and costs (Line 6). We then solve the resulting PCST problem in $\mathcal{G}'$ (Line 8) to obtain the best subgraph $\mathcal{S}'$, which we then convert back to a subgraph of $\mathcal{G}$ (Line 10). Finally, the score $s(t)$ is the total bitsave of $\mathcal{S}$, defined as $f_t(\mathcal{S})$ in (9.6).

### 9.5.3 Theoretical Results

**Theorem 9.1**

Algorithm 9.1 is online, requiring bounded memory and linear time.

*Proof.* **Memory.** At time $t$, the only data we need to store over time are the sensor values within a window from time $t - w$ to $t + w$ for $O(m + n)$ sensors, used for computing (9.1), while takes $O(w(m + n))$ memory, which is bounded regardless of the stream length.

**Running time.** Computing all $m + n$ bitsave scores at time $t$ takes $O(C(m + n))$ time, where $C$ is the time to fit $\hat{Y}$ in Eq. (9.2), which depends on the model family being used. For AR, seasonal AR and the constant (mean) model, $C$ is (amortized) constant, independent of $w$, as we show in our supplementary document [sup18]. Then Lines 4 to 12 are linear in $m + n$,

**Algorithm 9.1:** CHANGEDAR-S change scoring algorithm

**Input** : Graph $\mathcal{G}$, sensor stream $X_v(t), X_e(t)$ over time, window size $w$
**Output**: For each $t$, the best change-point localized set $\mathcal{S}(t)$, and its bitsave score $s(t)$

Convert $\mathcal{G}$ to $\mathcal{G}'$ s.t.: $(v', e') \in \mathcal{E}'$ iff $v$ is adjacent to $e$ in $\mathcal{G}$
**while** *sensor values for time tick* $t + w$ *are received* **do**
  ▷Compute profits $\pi$ and costs $c$
  $\pi(v') = \Delta_t(v) + k \cdot C_F$
  $\pi(e') = \Delta_t(e) + k \cdot C_F$
  $c(\cdot) = k \cdot C_F + \log(m + n)$
  ▷Solve (9.7) using Prize-Collecting Steiner Tree
  $\mathcal{S}' = \text{PCST}(\mathcal{G}', \pi(\cdot), c(\cdot))$
  ▷$\mathcal{S}$ is the nodes and edges corresponding to $\mathcal{S}'$
  $\mathcal{S}(t) = \{v : v' \in \mathcal{S}'\} \cup \{e : e' \in \mathcal{S}'\}$
  ▷$s(t)$ is the total bitsave of $\mathcal{S}$
  $s(t) = f_t(\mathcal{S})$
**end**

while the PCST step takes $O((m + n) \log n)$ time. Hence, Algorithm 9.1 is online, requiring $O((m + n)(C + \log n))$ per time step, or $O(T(m + n)(C + \log n))$ for $T$ time ticks. ■

**Theorem 9.2**

Algorithm 9.1 provides an approximation guarantee of 2 for optimizing (9.7): letting $\mathcal{S}'$ be the returned set and $\mathcal{S}^*$ the optimal solution:

$$f'_t(\mathcal{S}') \leq 2 \cdot f'_t(\mathcal{S}^*). \tag{9.9}$$

*Proof.* The optimization objective (9.7) is in the Prize-Collecting Steiner Tree form. By [HIS15], this can be solved in near-linear time with an approximation guarantee of 2. ■

## 9.6 Change Detection: CHANGEDAR-D

So far, for each time tick $t = 1, 2, \cdots$, we have found a localized change at subgraph $\mathcal{S}(t)$, with score $s(t)$. Our goal now is to decide which of them are actually change points: i.e. to output, in an online manner, a set of time ticks $\{t_1, t_2, \cdots\} \subseteq \{1, 2, \cdots\}$ and corresponding subsets $\mathcal{S}(t_1), \mathcal{S}(t_2), \cdots$ such that a change occurred in each subset $\mathcal{S}(t_i)$ at time $t_i$.

### 9.6.1 Optimization Objective

Intuitively, since the scores $s(t)$ represent bits saved, we want to pick the best set of time ticks $\{t_1, t_2, \cdots\} \subseteq \{1, 2, \cdots\}$ maximizing total bits saved, but without selecting conflicting change points. Recall that a change point $(t, \mathcal{S}(t))$ is scored based on its bits saved in the sensors in $\mathcal{S}(t)$ from time $t - w$ to $t + w$. This means that for two change points $t_i$ and $t_j$, if their subgraphs $\mathcal{S}(t_i)$ and $\mathcal{S}(t_j)$ have nonempty intersection, and their time intervals also overlap (i.e. $|t_i - t_j| \leq 2w$), then these two change points **conflict**, i.e. they cannot both be chosen in the final set of changes. In this way we define the **conflict graph**:

**Definition 9.2: Conflict Graph**

The **conflict graph** $\mathcal{G}_{\text{conf}} = (\mathcal{V}_{\text{conf}}, \mathcal{E}_{\text{conf}})$ has a node for each time tick: $\{1, 2, \cdots\}$. For $t_i, t_j \in \{1, 2, \cdots\}$, it has an edge between $t_i$ and $t_j$ iff the two change points $(t_i, \mathcal{S}(t_i))$ and $(t_j, \mathcal{S}(t_j))$ conflict, i.e.:

$$|t_i - t_j| \leq 2w, \quad \text{and} \quad |\mathcal{S}(t_i) \cap \mathcal{S}(t_j)| > 0.$$

Our goal is to choose the set of non-conflicting change points with highest total bitsave. As an optimization objective:

$$\begin{aligned} &\max_{\mathcal{T} \subseteq \{1,2,\cdots\}} \; g(\mathcal{T}) = \sum_{t \in \mathcal{T}} s(t) \\ &\textbf{subject to: } (t_i, t_j) \notin \mathcal{E}_{\text{conf}} \; \forall \; (t_i, t_j) \in \mathcal{T} \end{aligned} \tag{9.10}$$

### 9.6.2 Optimization Approach

Objective (9.10) is equivalent to a Maximum Weight Independent Set (MWIS) problem, where we put weights of $s(t)$ on node $t$, and find the max weight set which is independent (i.e. has no edges) in graph $\mathcal{G}_{\text{conf}}$. The key challenge, however, is that we need to optimize (9.10) in an **online** manner in which we receive nodes in $\mathcal{G}_{\text{conf}}$, along with their weight and conflicting edges, incrementally. As reviewed in Section 9.3.2, [GHK+14] showed that no meaningful worst-case guarantees are possible for the general online MWIS problem.

In our case, however, we have a **constrained** online MWIS problem: note that each node can only conflict with the $2w$ nodes to its immediate past and future. This turns out to be sufficient for solving the problem with constant approximation guarantee (treating $w$ as fixed).

We optimize (9.10) using a hybrid greedy-randomized approach, which keeps track of 1 greedy solution, and $r$ randomized solutions, and returns the best solution out of these at each time. The greedy solution performs better in practice, but the randomized algorithm provides better theoretical guarantees. Hence, performing both algorithms and returning whichever gives a higher objective value gives the 'best of both worlds' in terms of both empirical performance and theoretical guarantees.

The full algorithm is given in Algorithm 9.2. Lines 8 to 15 perform a greedy choice: if $s(t)$ exceeds the score of all its conflicting neighbors (Line 9), we add $t$ into the greedy set $\mathcal{T}^{(0)}$ (Line 11) and remove all its conflicting neighbors (Line 12). Based on the changes we made to $\mathcal{T}^{(0)}$, we then update $g_{\mathcal{T}^{(0)}}$ (a variable keeping track of $g(\mathcal{T}^{(0)})$ i.e. Objective (9.10)) accordingly (Line 14).

Lines 16 to 26 perform a randomized choice: we sample a uniform random value $u_t$ from 0 to 1 for each time $t$ (Line 18). In the $i$th repetition, if $u_t$ is greater than the random values of all its neighbors, we add $t$ to the set $\mathcal{T}^{(i)}$ and remove its neighbors (Lines 21 to 22). As before we then update $g_{\mathcal{T}^{(i)}}$ accordingly (Line 24).

Finally, after each tick we return the best solution (Line 28). Notice that both the Greedy and Randomized parts always return a feasible solution with no conflicts, since we remove all of a node's neighbors when we insert it into one of the $\mathcal{T}^{(i)}$.

### 9.6.3 Theoretical Results

**Theorem 9.3**

Algorithm 9.2 is online, requiring bounded memory and linear time.

*Proof.* **Memory.** At time $t$, we only need to store the nodes and edges of $\mathcal{G}_{\text{conf}}$ for up to the past $2w$ time ticks, since all neighbors of the node at time $t$ are at time $t - 2w$ or later. This requires $O(w)$ for the nodes and at most $O(w^2)$ for the edges. For the randomized part, we need to store $r$ uniform values for each of these nodes, requiring $O(wr)$ memory. Thus the total memory usage is $O(w(r + w))$, where $r$ and $w$ are small constants.

**Running time.** Each iteration of the greedy part takes $O(w)$ to visit each neighbor, and $O(w)$ in the same way for each of the $r$ repetitions of the randomized part. Hence the overall running time is $O(wr)$ per iteration, or $O(wrT)$ overall for $T$ time ticks. We show that a constant value of $r$ is sufficient in Theorem 9.4. ■

**Theorem 9.4**

Algorithm 9.2 has constant-factor approximation guarantee in expectation: letting $\mathcal{T}_{\text{best}}$ be the output of Algorithm 9.2 and $\mathcal{T}^*$ be the optimal solution of (9.7):

$$\mathbb{E}[g(\mathcal{T}_{\text{best}})] \geq \frac{1}{4w+1} g(\mathcal{T}^*) \tag{9.11}$$

*Proof.* Consider a fixed repetition $i$ of the randomized algorithm, at any current time tick $t'$. For each node $t$, note that $t$ is included in $\mathcal{T}^{(i)}$ iff $u_t^{(i)}$ is greater than its neighbors' values. Since $t$

**Algorithm 9.2:** ChangeDAR-D change detection algorithm

**Input** : Conflict graph $\mathcal{G}_{\text{conf}}$, window size $w$, and change sets with scores $(t, \mathcal{S}(t), s(t))$ as a stream at time $t = 1, 2, \cdots$ from ChangeDAR-S

**Output**: Set of change points $\mathcal{T}_{\text{best}} = \{(t_1, \mathcal{S}(t_1)), (t_2, \mathcal{S}(t_2)), \cdots\}$

▷Change sets $\mathcal{T}^{(i)}$, and corresponding objective values $g_{\mathcal{T}^{(i)}}$

$\mathcal{T}^{(i)} = \{\}\ \forall\ i = 0, 1, \cdots, r$

$g_{\mathcal{T}^{(i)}} = 0\ \forall\ i = 0, 1, \cdots, r$

**while** *node for time tick* $t + w$ *is received* **do**

  ▷Neighbor set; i.e. nodes conflicting with $t$

  $\mathcal{N}_t = \{t' : (t, t') \in \mathcal{E}_{\text{conf}}\}$

  ▷Run **Greedy** and **Randomized** and choose the best obtained set:

  ▷ **Greedy**: if $t$'s score exceeds sum of neighbors' scores:

  **if** $s(t) > \sum_{t' \in \mathcal{N}_t} s(t')$ **then**

    ▷add $t$ into $\mathcal{T}^{(0)}$ and remove its neighbors

    $\mathcal{T}^{(0)} = \mathcal{T}^{(0)} \cup \{t\}$

    $\mathcal{T}^{(0)} = \mathcal{T}^{(0)} \setminus \mathcal{N}_t$

    ▷update objective for $\mathcal{T}^{(0)}$ based on changes to it

    $g_{\mathcal{T}^{(0)}} = g_{\mathcal{T}^{(0)}} + s(t) - \sum_{t' \in \mathcal{N}_t} s(t')$

  **end**

  ▷ **Randomized**: over $r$ repetitions:

  **for** $i$ *in* $1$ *to* $r$ **do**

    Sample $u_t^{(i)} \sim \text{Uniform}(0, 1)$

    ▷If $t$'s value is greater than its neighbors' values

    **if** $u_t^{(i)} > \max_{t' \in \mathcal{N}_t} u_{t'}^{(i)}$ **then**

      $\mathcal{T}^{(i)} = \mathcal{T}^{(i)} \cup \{t\}$

      $\mathcal{T}^{(i)} = \mathcal{T}^{(i)} \setminus \mathcal{N}_t$

      ▷update objective for $\mathcal{T}^{(i)}$ based on changes to it

      $g_{\mathcal{T}^{(i)}} = g_{\mathcal{T}^{(i)}} + s(t) - \sum_{t' \in \mathcal{N}_t} s(t')$

    **end**

  **end**

  ▷Output best subset out of Greedy and Randomized

  **Output** $\mathcal{T}_{\text{best}} = \arg\max_{\mathcal{T} \in \{\mathcal{T}^{(0)}, \cdots, \mathcal{T}^{(r)}\}} g_{\mathcal{T}}$

**end**

can only be neighbors with the time ticks from $t - 2w$ to $t + 2w$, it has at most $4w$ neighbors. Thus, $u_t^{(i)}$ is greater than all its neighbors with probability $\geq \frac{1}{4w+1}$.

We next compute the expected value of $g(\mathcal{T}^{(i)})$. In the following, (9.12) is the definition of $g$, (9.13) is by linearity of expectation, (9.14) is since $t$ is included with probability $\geq \frac{1}{4w+1}$, and

(9.15) is since $g(\mathcal{T}^*)$ is a sum of some subset of the $s(\cdot)$ score values.

$$\mathbb{E}[g(\mathcal{T}^{(i)})] = \mathbb{E}\left[\sum_{t\in\mathcal{T}^{(i)}} s(t)\right] \tag{9.12}$$

$$= \sum_{t=1}^{t'} s(t)\cdot P(t\in\mathcal{T}^{(i)}) \tag{9.13}$$

$$\geq \sum_{t=1}^{t'} \frac{s(t)}{4w+1} \tag{9.14}$$

$$\geq \frac{1}{4w+1} g(\mathcal{T}^*) \tag{9.15}$$

■

The guarantee in Theorem 9.4 is a guarantee in expectation, but can also be converted to a 'with high-probability' guarantee. Specifically,

**Theorem 9.5**

Algorithm 9.2 has constant-factor approximation guarantee with high probability: formally, for any $0<\delta,\varepsilon<1$, as long as we set $r \geq \frac{\log\delta}{\log(\frac{4w}{4w+\varepsilon})}$, then with probability at least $1-\delta$:

$$g(\mathcal{T}_{\text{best}}) \geq \frac{1-\varepsilon}{4w+1} g(\mathcal{T}^*) \tag{9.16}$$

*Proof.* For any $i$, note that $g(\mathcal{T}^{(i)}) \leq g(\mathcal{T}^*)$, and by Theorem 9.4, $\mathbb{E}[\frac{g(\mathcal{T}^{(i)})}{g(\mathcal{T}^*)}] \geq \frac{1}{4w+1}$. By Markov inequality on the nonnegative variable $1-\frac{g(\mathcal{T}^{(i)})}{g(\mathcal{T}^*)}$:

$$P(\frac{g(\mathcal{T}^{(i)})}{g(\mathcal{T}^*)} \leq \frac{1-\varepsilon}{4w+1}) = P(1-\frac{g(\mathcal{T}^{(i)})}{g(\mathcal{T}^*)} \geq \frac{4w+\varepsilon}{4w+1}) \tag{9.17}$$

$$\leq \frac{1-\mathbb{E}[\frac{g(\mathcal{T}^{(i)})}{p} g(\mathcal{T}^*)]}{\frac{4w+\varepsilon}{4w+1}} \tag{9.18}$$

$$\leq \frac{1-\frac{1}{4w+1}}{\frac{4w+\varepsilon}{4w+1}} = \frac{4w}{4w+\varepsilon} \tag{9.19}$$

Since $g(\mathcal{T}_{\text{best}})$ takes the max of $r$ independent repetitions:

$$P(g(\mathcal{T}_{\text{best}}) \leq \frac{1-\varepsilon}{4w+1}) = (\frac{4w}{4w+\varepsilon})^r \tag{9.20}$$

$$\leq \delta \text{ if } r \geq \frac{\log\delta}{\log(\frac{4w}{4w+\varepsilon})} \tag{9.21}$$

■

For example, set $\varepsilon$ and $\delta$ to small constants, e.g. $0.01$. Then this theorem implies that $P(g(\mathcal{T}_{\text{best}}) \geq \frac{0.99}{4w+1})$ holds with probability at least $0.99$ as long as $r$ is at least a constant value.

## 9.7 Experiments

We design experiments to answer the following questions:

- **Q1. Change Detection Accuracy:** how accurate are the change times detected by ChangeDAR?
- **Q2. Localization Accuracy:** how accurate are the change locations reported by ChangeDAR?
- **Q3. Scalability:** how does our method scale with data size?

Our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/changedar/`. Experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM running OS X 10.11.2. For window size $w$, small values around $5$ are a good default, as larger values might miss short-lasting changes. Hence we use $w = 5$. We set error variance $\sigma^2$ (Eq. (9.3)) to $0.05$ for the power grid data and $1.5$ for the traffic data.

Table 9.3: Datasets used

| Dataset name | Nodes | Edges | Time Ticks | Domain | Sensors |
|---|---|---|---|---|---|
| PolandHVN1 [ZMST11] | 2383 | 2896 | 480 | Power | Voltage |
| PolandHVN2 [ZMST11] | 2737 | 3506 | 480 | Power | Voltage |
| PolandHVN3 [ZMST11] | 3012 | 3572 | 480 | Power | Voltage |
| PolandHVN4 [ZMST11] | 3120 | 3693 | 480 | Power | Voltage |
| TrafficLA [pem18] | 4828 | 4868 | 2016 | Traffic | Speed |

### 9.7.1 Q1. Detection Accuracy

In this section, we compare ChangeDAR against baseline change detection approaches, in their accuracy for detecting power line failures simulated using Matpower [ZMST11], a standard power system simulation program.

#### Experimental Settings

For each graph, out of $480$ time ticks (hourly data for $20$ days) we sample $10$ random time ticks as the times when changes occur. In each such time tick, we deactivate a randomly chosen edge (i.e. no current can flow over that edge) for the remainder of the time period. As input to Matpower, we use load patterns estimated from real data [SHJ+17] from the Carnegie Mellon University (CMU) campus for $20$ days from July 29 to August 17, 2016, with its standard deviation scaled down by a factor of $10$.

Given this input, each algorithm returns a set of time ticks where it detected a change. We evaluate this using F-measure ($\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision}+\text{recall}}$) compared to the true set of anomalies.

**Baselines**

We compare ChangeDAR to the following change detection methods:

- *Dynamic Graph Change Detection:* WeightDist [Pin05] and VertexRank [PDGM10].
- *Multivariate Change Detection:* PELT [KFE12] and Group Fused Lasso (GFL) [BV11].
- *Graph-based Scan Statistics:* Graph Fourier Scan Statistic (GFSS) [SRS16].

Only our method and VertexRank are online; for VertexRank, the original algorithm actually requires an offline standard deviation, but we assume this can be done online as well. We use the offline version of VertexRank in our experiments.

For WeightDist, we select the ARMA orders using AIC, with an upper limit of 2 following the original paper [Pin05]. For VertexRank we use a threshold of 2 standard deviations, following the original paper [PDGM10]. GFL and GFSS require the number of changes (i.e. the number of time ticks where changes occur) as input: we pass in the true number of changes. For GFL we use the default LARS (Least Angle Regression) approach. GFSS returns a statistic at each time for how clustered the voltage patterns are with respect to the graph: instead of using GFSS directly (which would detect *individual* time ticks with abnormal sensor values, rather than detecting changes), since we are dealing with change points, it makes more sense to use GFSS on *differences* between the data at adjacent time points, to detect large differences. Hence, we convert all the original time series to differences in this way before using GFSS. For GFSS we set the threshold $\rho$ as the 5th-percentile smallest eigenvalue, following the original paper [SRS16].

**Results**

Figure 9.3 shows an example of a power failure correctly detected, shown by the blue cross. This change causes cascading effects in the voltage levels in the surrounding nodes, which ChangeDAR is able to detect (red circles). We evaluate the set of change times output by each method against the ground truth values using F-measure on the 4 datasets in Figure 9.4a to 9.4d. The results show that ChangeDAR outperforms the baselines by 75% or more. Since only ChangeDAR detects changes that are localized in the graph and persist over a period of time, this suggests that combining graph and temporal structure in this way is effective. Note that this occurs despite the baselines (other than VertexRank) being offline algorithms, while our method is online.

### 9.7.2 Q2. Localization Accuracy

We evaluate ChangeDAR in detecting and locating traffic accidents in the `TrafficLA` data. Figure 9.1 shows a traffic incident reported in the incident report as a traffic collision at 5.30pm on Jan 3 2018. ChangeDAR reported a localized change at the time shown by the red vertical line and at the three stations marked in red, all of which experienced sharp drops in average speed shortly after the accident.

We now evaluate the accuracy of ChangeDAR against the ground truth traffic accidents, compared to the same set of baselines (with the same settings as before). Since none of the baselines perform localization (i.e. returning the location of an anomaly), for each change point returned by a baseline, we select the sensor with the largest negative change in speed at that

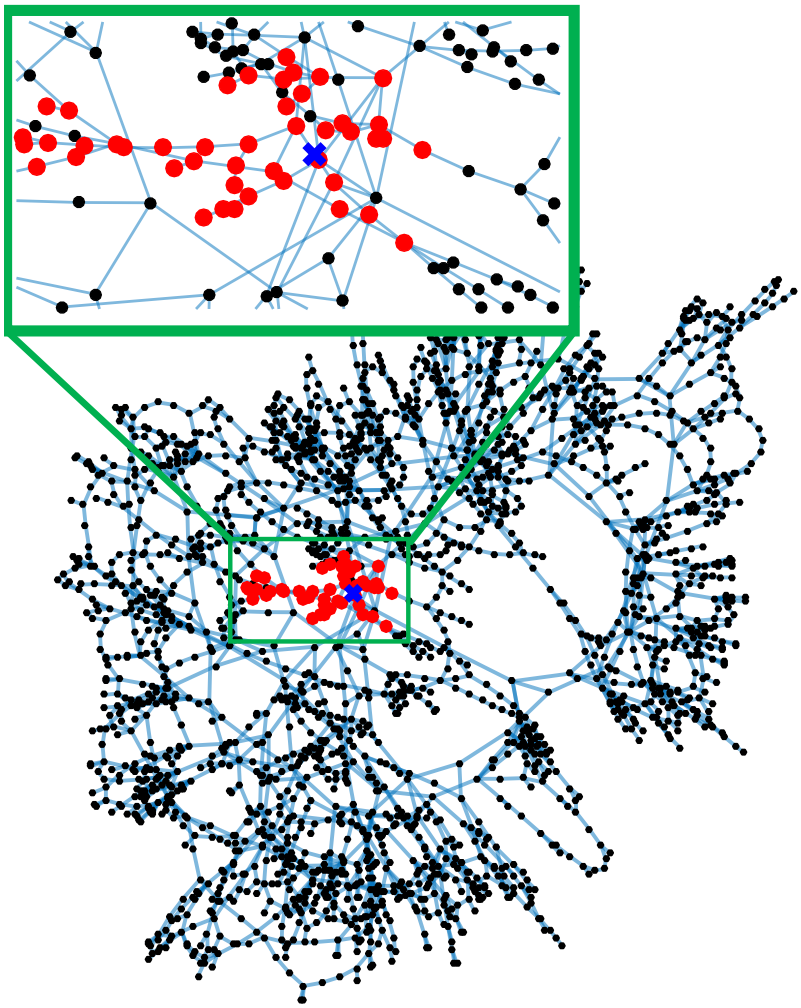
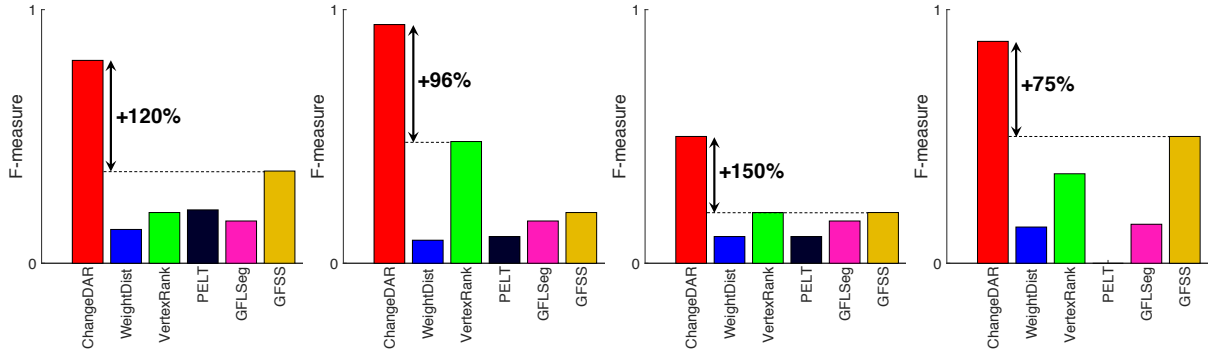


Figure 9.3: **ChangeDAR correctly detects a power failure:** the blue cross shows a power line failure, causing cascading effects in surrounding voltage levels, which are detected as a localized change-point by ChangeDAR (red circles).

time as the detected location. We only consider negative changes since traffic accidents should only result in decreases in vehicle speed.

The ground truth accidents come from incident reports by the California Highway Patrol. Each incident is accompanied by its occurrence time and location. We use all events listed as traffic collisions with duration at least 1 hour. For each change reported by an algorithm, we match it to a ground truth incident if the two occurred at most 1 hour apart, and the mean location among the algorithm's detected nodes is at most '*radius*' away from the true location, for the values of '*radius*' plotted on the x-axis of Figure 9.5.

Figure 9.5 shows that ChangeDAR outperforms the same baselines in precision and recall of locating traffic accidents, by 70% and 227% respectively, and F-measure by 124%. The fairly low precision and recall of all methods occurs because many traffic accidents do not lead to




(a) PolandHVN1 (b) PolandHVN2 (c) PolandHVN3 (d) PolandHVN4

Figure 9.4: **ChangeDAR accurately detects power line failures:** F-measure of detecting transmission line failures compared to (mostly offline) baselines.

any discernible change in vehicle speed, and conversely, traffic slowdowns may be caused by regular congestion or events that are not reported as traffic accidents.

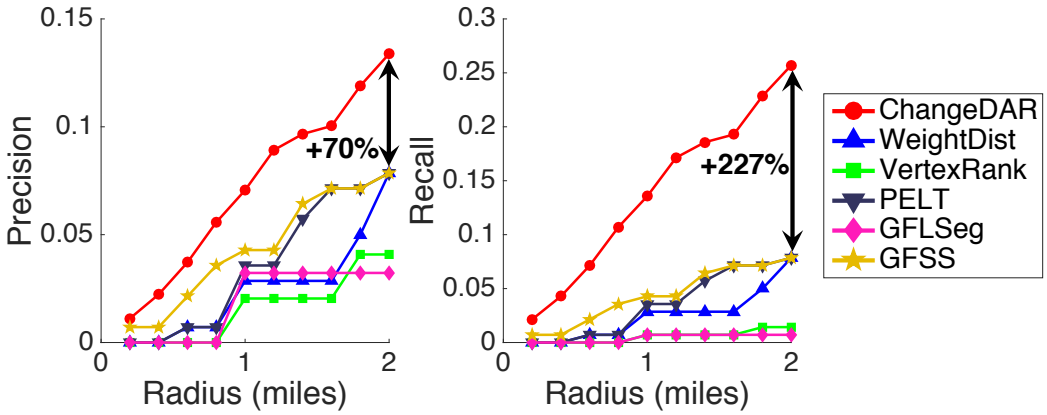


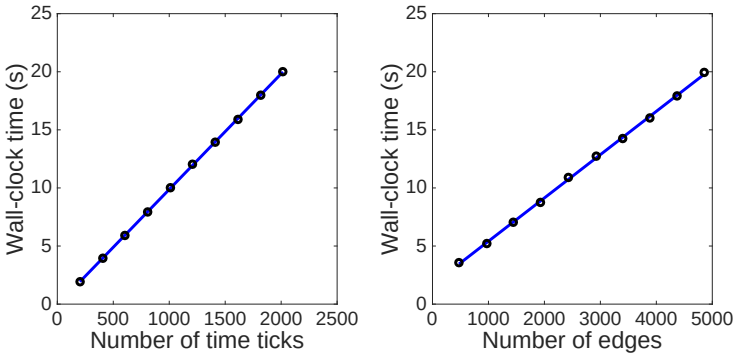

Figure 9.5: **ChangeDAR accurately locates traffic accidents**: ChangeDAR outperforms baselines in precision (left) and recall (right) on ground truth traffic accidents. The x-axis plots the radius used to determine whether a change point output by each algorithm matches a ground truth accident.

### 9.7.3 Q3. Scalability

Finally, we verify that ChangeDAR scales linearly in the number of time ticks and the number of edges in the graph. Figure 9.6a plots the wall-clock time taken for ChangeDAR to run on the `TrafficLA` dataset, varying the number of time ticks from $10\%, 20\%, \cdots, 100\%$ of the full number of time ticks. For Figure 9.6b we construct subsets of nodes by taking the $10\%, 20\%, \cdots, 100\%$ of nodes with lowest geographical latitude, and run ChangeDAR on the induced subgraphs of these subsets. Subsetting via latitude is done to prevent the graph from

separating into a large number of connected components. Figures 9.6a and 9.6b show that ChangeDAR scales linearly in both dimensions.

ChangeDAR is fast, taking 10ms on average to run on each time tick on our largest (TrafficLA) graph, with 4828 sensor values per time tick.




(a) Time ticks (b) Edges

Figure 9.6: **ChangeDAR scales linearly:** wall-clock time of ChangeDAR against (a) number of time ticks and (b) number of edges.

## 9.8 Conclusion

In this chapter, we propose online algorithms for detecting localized changes for sensor data on a graph. This type of data occurs in many settings: e.g. power grid monitoring, traffic, climate, disease surveillance, and monitoring users on social networks. Our approach uses PCST and online MWIS, which to the best of our knowledge, have not been used for change detection. Our contributions are:

1. **Algorithm:** We propose novel information theoretic optimization objectives for 1) scoring and 2) detecting localized changes, and propose two algorithms, ChangeDAR-S and ChangeDAR-D respectively, to optimize them.
2. **Theoretical Guarantees:** We show that both algorithms provide constant-factor approximation guarantees (Theorems 9.2 and 9.4).
3. **Effectiveness:** Our algorithms detect traffic accidents and power line failures in a power grid with 75% or more higher F-measure than comparable baselines in experiments.
4. **Scalability:** Our full algorithm is online and near-linear in the graph size and the number of time ticks (Figure 9.6).

**Reproducibility:** our code and data are publicly available at http://www.andrew.cmu.edu/user/bhooi/changedar/.

# Chapter 10

# GridWatch: Sensor Selection and Anomaly Detection on the Power Grid

Chapter based on work published in ECML-PKDD18 [PDF].

In this chapter, we propose an anomaly detection method designed for power-grid graphs, as well as a sensor selection approach. Given sensor readings over time from a power grid, how can we accurately detect when an electrical component has failed? We propose GridWatch, an online and linear-time detection approach, as well as a provably near-optimal sensor selection approach. We show experimentally that our methods are effective, outperforming existing approaches in accuracy by $59\%$ or more F-measure.

## 10.1 Introduction

Improving the efficiency and security of power delivery is a critically important goal, in the face of disturbances arising from severe weather, human error, equipment failure, or even intentional intrusion. Estimates [Ami11] suggest that reducing outages in the U.S. grid could save $\$49$ billion per year, reduce emissions by $12$ to $18\%$, while improving efficiency could save an additional $\$20.4$ billion per year. A key part of achieving this goal is to use sensor monitoring data to quickly identify when parts of the grid fail, so as to quickly respond to the problem.

A major challenge is scalability - power systems data can be both high-volume and received in real time, since the data comes from sensors which are continuously monitoring the grid. This motivates us to develop fast methods that work in this online (or streaming) setting. When each new data point is received, the algorithm should update itself efficiently - for our algorithm, each update requires constant time, and bounded memory, regardless of the length of the stream.

Hence, our goal is an online anomaly detection algorithm:

**Informal Problem 10.1: Online Anomaly Detection**

- **Given:** A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a subset $\mathcal{S}$ of nodes which contain sensors. For each sensor, we have a continuous stream of values of real and imaginary voltage $V(t)$ and current $I(t)$ measured by these sensors.
- **Find:** At each time $t$, compute an anomalousness score $A(t)$, indicating our confidence level that an anomaly occurred (i.e. a transmission line failed).

For cost reasons, it is generally infeasible to place sensors at every node. Hence, an important follow-up question is where to place sensors so as to maximize the probability of detecting an anomaly.

**Informal Problem 10.2: Sensor Placement**

- **Given:** A budget $k$ of the number of sensors we can afford, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a simulator that allows us to simulate sensor readings at each node.
- **Find:** A set of nodes $\mathcal{S} \subseteq \mathcal{V}$, which are the locations we should place our sensors, such that $|\mathcal{S}| = k$.

In contrast to most approaches, our anomaly detection algorithm, GridWatch-D, uses a domain-dependent approach which exploits the fact that electrical sensors consist of a voltage reading at a node as well as the current along each adjacent edge. This allows us to detect anomalies more accurately, even when using an online approach. Next, we propose GridWatch-S, a sensor placement algorithm. The main idea is to define an objective which estimates our probability of successfully detecting an anomaly, then show that this objective has the submodularity property, allowing us to optimize it with approximation guarantees using an efficient greedy algorithm.

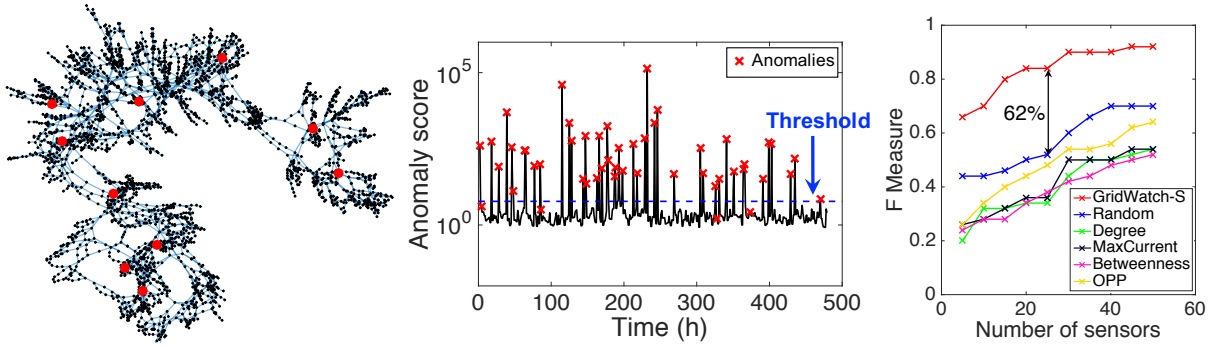
Figure 10.1a shows the sensors selected by GridWatch-S: red circles indicate positions chosen. Figure 10.1b shows the anomaly scores (black line) output by GridWatch-D, which accurately match the ground truth. Figure 10.1c shows that GridWatch-S outperforms baselines on the `case2869` data.

Our contributions are as follows:

1. **Online anomaly detection:** we propose a novel, online anomaly detection algorithm, GridWatch-D, that outperforms existing approaches.
2. **Sensor placement:** we construct an optimization objective for sensor placement, with the goal of maximizing the probability of detecting an anomaly. We show that this objective has the property of 'submodularity,' which we exploit to propose our sensor placement algorithm.
3. **Effectiveness:** Our sensor placement algorithm, GridWatch-S, is provably near-optimal. In addition, both our algorithms outperform existing approaches in accuracy by $59\%$ or more (F-measure) in experiments.

4. **Scalability:** Our algorithms scale linearly, and GridWatch-D is online, requiring bounded space and constant time per update.

**Reproducibility:** our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/code/`.




(a) Selects sensor locations (b) Anomaly scores (c) Accuracy

Figure 10.1: (a) GridWatch-S provably selects near-optimal sensor locations. Red circles indicate positions chosen for sensors, in the `case2869` graph. (b) GridWatch-D computes anomaly scores (black line) on `case2869`. Red crosses indicate ground truth - notice 100% true alarms (all black spikes above blue line are true alarms) and only 4 false dismissals (red crosses below blue line). Threshold (blue line) is 4 IQRs above the mean. (c) F-measure of GridWatch-S compared to baselines on `case2869`.

## 10.2 Background and Related Work

**Time Series Anomaly Detection.** Numerous algorithms exist for anomaly detection in univariate time series [KLLVH07]. For multivariate time series, LOF [BKNS00] uses a local density approach. Isolation Forests [LTZ08] partition the data using a set of trees for anomaly detection. Other approaches use neural networks [YJYC17], distance-based [RRS00], and exemplars [JNIH14]. However, none of these consider sensor selection.

**Anomaly Detection in Temporal Graphs.** [AMF10] finds anomalous changes in graphs using an egonet (i.e. neighborhood) based approach, while [CHS12] uses a community-based approach. [MBR+13] finds connected regions with high anomalousness. [APG+14] detect large and/or transient communities using Minimum Description Length. [AF10] finds change points in dynamic graphs, while other partition-based [AZP11] and sketch-based [RHSS16] exist for anomaly detection. However, these methods require fully observed edge weights (i.e. all sensors present), and also do not consider sensor selection.

**Power Grid Monitoring.** A number of works consider the Optimal PMU Placement (OPP) problem [BH05], of optimally placing sensors in power grids, typically to make as many nodes

as possible fully observable, or in some cases, minimizing mean-squared error. Greedy [LNI11], convex relaxation [KGW12], integer program [DDGS08], simulated annealing [BMBA93], and swarm-based [CVK08] approaches have been proposed. However, these methods do not perform anomaly detection. [RPUW07, ZGP12, MA99] consider OPP in the presence of branch outages, but not anomalies in general, and due to their use of integer programming, only use small graphs of size at most 60.

**Epidemic and Outbreak Detection.** [LKG+07] proposed CELF, for outbreak detection in networks, such as water distribution networks and blog data, also using a submodular objective function. Their setting is a series of cascades spreading over the graph, while our input data is time-series data from sensors at various edges of the graph. For epidemics, [PSV02, CHBA03] consider targeted immunization, such as identifying high-degree [PSV02] or well-connected [CHBA03] nodes. We show experimentally that our sensor selection algorithm outperforms both approaches.

Table 10.1: Comparison of related approaches: only GridWatch satisfies all the listed properties.

| *Property* | Time Series [KLLVH07], etc. | Graph-based [AMF10] | OPP [BH05], etc. | Immunization [PSV02] | GridWatch |
|---|---|---|---|---|---|
| **Anomaly Detection** | ✓ | ✓ | | | ✓ |
| **Online Algorithm** | ✓ | | | | ✓ |
| **Using Graph Data** | | ✓ | ✓ | ✓ | ✓ |
| **Sensor Selection** | | | ✓ | ✓ | ✓ |
| **With Approx. Guarantee** | | | | | ✓ |

Table 10.1 summarizes existing work related to our problem. GridWatch differs from existing methods in that it performs anomaly detection using an online algorithm, and it selects sensor locations with a provable approximation guarantee.

### 10.2.1 Background: Submodular Functions

A function $f$ defined on subsets of $\mathcal{V}$ is submodular if whenever $\mathcal{T} \subseteq S$ and $i \notin S$:

$$f(\mathcal{S} \cup \{i\}) - f(\mathcal{S}) \leq f(\mathcal{T} \cup \{i\}) - f(\mathcal{T}) \tag{10.1}$$

Intuitively, this can be interpreted as *diminishing returns*: the left side is the gain in $f$ from adding $i$ to $\mathcal{S}$, and the right side is the gain from adding $i$ to $\mathcal{T}$. Since $\mathcal{T} \subseteq S$, this says that as $\mathcal{T}$ 'grows' to $\mathcal{S}$, the gains from adding $i$ can only diminish.

[NWF78] showed that nondecreasing submodular functions can be optimized by a greedy algorithm with a constant-factor approximation guarantee of $(1 - 1/e)$. These were extended by [Svi04] to the non-constant sensor cost setting.

## 10.3 GridWatch-D Anomaly Detection Algorithm

**Preliminaries** Table 10.2 shows the symbols used in this chapter.

Table 10.2: Symbols and definitions

| Symbol | Interpretation |
|---|---|
| $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ | Input graph |
| $\mathcal{S}$ | Subset of nodes to place sensors on |
| $n$ | Number of nodes |
| $s$ | Number of scenarios |
| $\mathcal{N}_i$ | Set of edges adjacent to node $i$ |
| $V_i(t)$ | Voltage at node $i$ at time $t$ |
| $I_e(t)$ | Current at edge $e$ at time $t$ |
| $S_{ie}(t)$ | Power w.r.t. node $i$ and edge $e$ at time $t$ |
| $\Delta S_{ie}(t)$ | Power change: $\Delta S_{ie}(t) = S_{ie}(t) - S_{ie}(t-1)$ |
| $X_i(t)$ | Sensor vector for scenario $i$ at time $t$ |
| $c$ | Anomalousness threshold parameter |
| $\tilde{\mu}_i(t)$ | Median of sensor $i$ at time $t$ |
| $\tilde{\sigma}_i(t)$ | Inter-quartile range of sensor $i$ at time $t$ |
| $a_i(t)$ | Sensor-level anomalousness for sensor $i$ at time $t$ |
| $A(t)$ | Total anomalousness at time $t$ |

In this section, we are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a fixed set of sensors $\mathcal{S} \subseteq \mathcal{V}$. Each sensor consists of a central node $i$ on which voltage $V_i(t) \in \mathbb{C}$ is measured, at each time $t$. Note that complex voltages and currents are used to take phase into account, following standard practice in circuit analysis (this chapter will not presume familiarity with this). Additionally, for sensor $i$, letting $\mathcal{N}_i$ be the set of edges adjacent to $i$, we are given the current $I_e \in \mathbb{C}$ along each edge $e \in \mathcal{N}_i$.

For sensor $i$ and edge $e \in \mathcal{N}_i$, define the power w.r.t. $i$ along edge $e$ as $S_{ie}(t) = V_i(t) \cdot I_e(t)^*$, where $^*$ is the complex conjugate. We find that using power (rather than current) provides better anomaly detection in practice. However, when considering the edges around a single sensor $i$, variations in current result in similar variations in power, so they perform the same role.

### 10.3.1 Types of Anomalies

In short, we will design detectors for 3 common types of anomalies (see Figure 10.2), then combine these detectors into a score for each sensor (Definition 10.4), then finally combine these sensor-level scores into an overall score for each time tick (Definition 10.5).

Our goal is to detect single edge deletions, i.e. a transmission line failure. Single edge deletions affect the voltage and current in the graph in a complex, nonlinear way, and can manifest themselves in multiple ways.

As an illustrative example, consider the simple power grid shown by the graphs in Figure 10.2. The power grid consists of a single generator, a single load, and power lines of uniform resistance. When the edge marked in the black cross fails, current is diverted from some edges to others, causing some edges to have increased current flow (blue edges), and thus increased power, and others to have decreased current flow (red edges). Current flows are computed using a standard power grid simulator, Matpower [ZMST11].

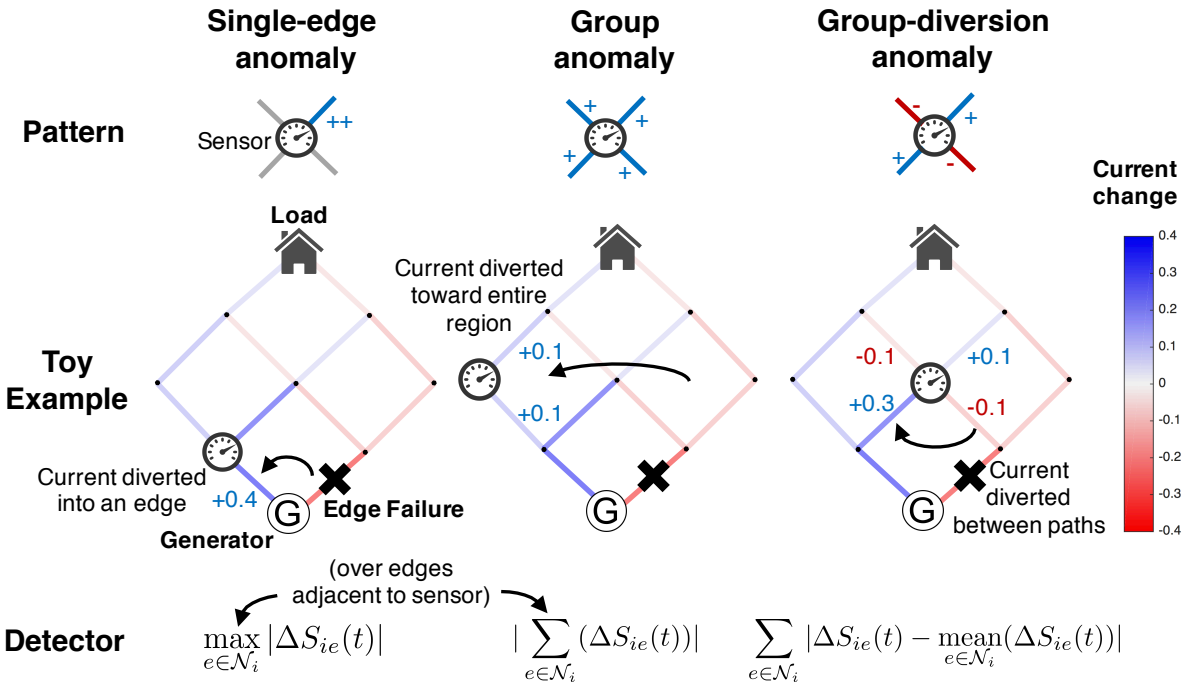



Figure 10.2: **Domain-aware model for anomalies:** edge failures divert current from one region of the graph to another, forming 3 patterns. Edge color indicates change in current due to the edge failure: blue is an increase; red is a decrease. *Left:* current from the deleted edge diverts into an edge, resulting in a highly anomalous value along a single edge. *Center:* the deletion diverts current between areas (from the right side to the left side of the graph), forming a group anomaly at the leftmost sensor, due to its multiple positive edges. *Right:* the deletion diverts current between paths, making the central sensor have multiple positive and negative edges.

In the leftmost plot, the edge deletion diverts a large amount of current into a single edge, resulting in a highly anomalous value $(+0.4)$ along a single edge. To detect single-edge anomalies, we consider the largest absolute change in power in the edges adjacent to this sensor. Formally, letting $\Delta S_{ie}(t) = S_{ie}(t) - S_{ie}(t-1)$,

**Definition 10.1: Single-Edge Detector**

The single-edge detector at sensor $i$ is:

$$x_{SE,i}(t) = \max_{e \in \mathcal{N}_i} |\Delta S_{ie}(t)| \tag{10.2}$$

In the middle plot, the edge deletion cuts off a large amount of current that would have gone from the generator toward the right side of the graph, diverting it into the left side of the graph. This results in some nodes in the left region with all their neighboring edges having positive changes (blue), such as the leftmost node. Individually, these changes may be too small to appear anomalous, but in aggregate, they provide stronger evidence of an anomaly. Hence, the group anomaly detector computes the sum of power changes around sensor $i$, then takes the absolute value:

**Definition 10.2: Group Anomaly Detector**

The group anomaly detector at sensor $i$ is:

$$x_{GA,i}(t) = |\sum_{e \in \mathcal{N}_i} (\Delta S_{ie}(t))| \tag{10.3}$$

In the right plot, the edge deletion diverts current between nearby edges. In particular, current diversions around the central node cause it to have neighbors which greatly differ from each other: $2$ positive edges and $2$ negative edges. If this diversion is large enough, this provides stronger evidence of an anomaly than simply looking at each edge individually. Hence, the group diversion detector measures the 'spread' around sensor $i$ by looking at the total absolute deviation of power changes about sensor $i$:

**Definition 10.3: Group Diversion Detector**

The group diversion detector at sensor $i$ is:

$$x_{GD,i}(t) = \sum_{e \in \mathcal{N}_i} |\Delta S_{ie}(t) - \operatorname*{mean}_{e \in \mathcal{N}_i}(\Delta S_{ie}(t))| \tag{10.4}$$

### 10.3.2 Proposed Anomaly Score

Having computed our detectors, we now define our anomaly score. For each sensor $i$, concatenate its detectors into a vector:

$$X_i(t) = [x_{SE,i}(t) \;\; x_{GA,i}(t) \;\; x_{GD,i}(t)] \tag{10.5}$$

Sensor $i$ should label time $t$ as an anomaly if any of the detectors greatly deviate from their historical values. Hence, let $\tilde{\mu}_i(t)$ and $\tilde{\sigma}_i(t)$ be the historical median and inter-quartile range (IQR)[1] [Yul19] of $X_i(t)$ respectively: i.e. the median and IQR of $X_i(1), \cdots, X_i(t-1)$. We use median and IQR generally instead of mean and standard deviation as they are robust against anomalies, since our goal is to detect anomalies.

Thus, define the sensor-level anomalousness as the maximum number of IQRs that any detector is away from its historical median:

**Definition 10.4: Sensor-level anomalousness**

The sensor-level anomalousness is:

$$a_i(t) = \left\| \frac{X_i(t) - \tilde{\mu}_i(t)}{\tilde{\sigma}_i(t)} \right\|_\infty \tag{10.6}$$

Here the infinity-norm $\|\cdot\|_\infty$ is the maximum absolute value over entries of a vector.

Finally, the **overall anomalousness** at time $t$ is the maximum of $a_i(t)$ over all sensors. Taking maximums allows us to determine the *location* (not just time) of an anomaly, since we can look at which sensor contributed toward the maximum.

**Definition 10.5: Overall anomalousness**

The overall anomalousness at time $t$ is:

$$A(t) = \max_{i \in \mathcal{S}} a_i(t) \tag{10.7}$$

Algorithm 10.1 summarizes our GridWatch-D anomaly detection algorithm. Note that we can maintain the median and IQR of a set of numbers in a streaming manner using reservoir sampling [Vit85]. Hence, the Normalize operation in Line 5 takes a value of $\Delta S_{ie}(t)$, subtracts its historical median and divides by the historical IQR for that sensor. This ensures that sensors with large averages or spread do not dominate.

[1] IQR is a robust measure of spread, equal to the difference between the 75% and 25% quantiles.

**Algorithm 10.1:** GridWatch-D online anomaly detection algorithm

**Input** : Graph $\mathcal{G}$, voltage $V_i(t)$, current $I_i(t)$
**Output**: Anomalousness score $A(t)$ for each $t$, where higher $A(t)$ indicates greater certainty of an anomaly

**for** $t$ *received as a stream:* **do**
  **for** $i \in \mathcal{S}$ **do**
    $S_{ie}(t) \leftarrow V_i(t) \cdot I_e^*(t) \; \forall \; e \in \mathcal{N}_i$ ▷Power
    $\Delta S_{ie}(t) \leftarrow S_{ie}(t) - S_{ie}(t-1)$ ▷Power differences
    $\Delta S_{i\cdot}(t) \leftarrow \textsc{Normalize}(\Delta S_{i\cdot})$
    Compute detectors $x_{SE,i}(t)$, $x_{GA,i}(t)$ and $x_{GD,i}(t)$ using Eq. (10.2) to (10.4)
    Concatenate detectors: $X_i(t) = [x_{SE,i}(t) \;\; x_{GA,i}(t) \;\; x_{GD,i}(t)]$
    $\tilde{\mu}_i(t) \leftarrow \textsc{UpdateMedian}(\tilde{\mu}_i(t-1), X_i(t))$ ▷Historical median
    $\tilde{\sigma}_i(t) \leftarrow \textsc{UpdateIQR}(\tilde{\sigma}_i(t-1), X_i(t))$ ▷Historical IQR
    $a_i(t) \leftarrow \| \frac{X_i(t) - \tilde{\mu}_i(t)}{\tilde{\sigma}_i(t)} \|_\infty$ ▷Sensor-level anomalousness
  **end**
  $A(t) = \max_{i \in \mathcal{S}} a_i(t)$ ▷Overall anomalousness
**end**

**Lemma 10.1**

GridWatch-D is online, and requires bounded memory and time.

*Proof.* We verify from Algorithm 10.1 that GridWatch-D's memory consumption is $O(|\mathcal{S}|)$, and updates in $O(|\mathcal{S}|)$ time per iteration, which are bounded (regardless of the length of the stream). ■

## 10.4 Sensor Placement: GridWatch-S

So far, we have detected anomalies using a fixed set of sensors. We now consider how to select locations for sensors to place given a fixed budget of $k$ sensors to place. Our main idea will be to construct an optimization objective for the anomaly detection performance of a subset $\mathcal{S}$ of sensor locations, and show that this objective has the 'submodularity' property, showing that a greedy approach gives approximation guarantees.

Note the change in problem setting: we are no longer monitoring for anomalies online in time series data, since we are now assuming that the sensors have not even been installed yet. Instead, we are an offline planner deciding where to place the sensors. To do this, we use a model of the system in the form of its graph $\mathcal{G}$, plugging it into a simulator such as Matpower [ZMST11] to generate a dataset of ground truth anomalies and normal scenarios, where the former contain a randomly chosen edge deletion, and the latter do not.

### 10.4.1 Proposed Optimization Objective

Intuitively, we should select sensors $\mathcal{S}$ to maximize the **probability of detecting an anomaly**. This probability can be estimated as the fraction of ground truth anomalies that we successfully detect. Hence, our optimization objective, $f(\mathcal{S})$, will be the fraction of anomalies that we successfully detect when using GridWatch-D, with sensor set $\mathcal{S}$. We will now formalize this and show that it is submodular.

Specifically, define $X_i(r)$ as the value of sensor $i$ on the $r$th anomaly, analogous to (10.5). Also define $\tilde{\mu}_i$ and $\tilde{\sigma}_i$ as the median and IQR of sensor $i$ on the full set of normal scenarios. Also let $a_i(r)$ be the sensor-level anomalousness of the $r$th anomaly, which can be computed as in Definition 10.4 plugging in $\tilde{\mu}_i$ and $\tilde{\sigma}_i$:

$$a_i(r) = \left\| \frac{X_i(r) - \tilde{\mu}_i}{\tilde{\sigma}_i} \right\|_\infty \tag{10.8}$$

Define overall anomalousness w.r.t. $\mathcal{S}$, $A(r, \mathcal{S})$, analogously to Definition 10.5:

$$A(r, \mathcal{S}) = \max_{i \in \mathcal{S}} a_i(r) \tag{10.9}$$

Given threshold $c$, anomaly $r$ will be detected by sensor set $\mathcal{S}$ if and only if $A(r, \mathcal{S}) > c$. Hence, our optimization objective is to maximize the fraction of detected anomalies:

$$\underset{\mathcal{S} \subseteq \mathcal{V}, |S| = k}{\text{maximize}} \; f(\mathcal{S}), \text{ where } f(\mathcal{S}) = \frac{1}{s} \sum_{r=1}^{s} \mathbf{1}\{A(r, \mathcal{S}) > c\} \tag{10.10}$$

### 10.4.2 Properties of Objective

Our optimization objective $f(\mathcal{S})$ is submodular: informally, it exhibits diminishing returns. The more sensors we add, the smaller the marginal gain in detection probability.

**Theorem 10.1**

Detection probability $f(\mathcal{S})$ is submodular, i.e. for all subsets $\mathcal{T} \subseteq \mathcal{S}$ and nodes $i \in \mathcal{V} \setminus \mathcal{S}$:

$$f(\mathcal{S} \cup \{i\}) - f(\mathcal{S}) \leq f(\mathcal{T} \cup \{i\}) - f(\mathcal{T}) \tag{10.11}$$

*Proof.* By definition of $f$, we have:

$$\begin{aligned} f(\mathcal{S} \cup \{i\}) - f(\mathcal{S}) &= \frac{1}{s} \sum_{r=1}^{s} \left( \mathbf{1}\{A(r, \mathcal{S} \cup \{i\}) > c\} - \mathbf{1}\{A(r, \mathcal{S}) > c\} \right) \\ &= \frac{1}{s} \sum_{r=1}^{s} \left( \mathbf{1}\{\max_{j \in \mathcal{S} \cup \{i\}} a_j(r) > c\} - \mathbf{1}\{\max_{j \in \mathcal{S}} a_j(r) > c\} \right) \\ &= \frac{1}{s} \sum_{r=1}^{s} \left( \mathbf{1}\{a_i(r) > c \wedge \max_{j \in \mathcal{S}} a_j(r) \leq c\} \right) \\ &\leq \frac{1}{s} \sum_{r=1}^{s} \left( \mathbf{1}\{a_i(r) > c \wedge \max_{j \in \mathcal{T}} a_j(r) \leq c\} \right) \quad \text{since } \mathcal{T} \subseteq \mathcal{S} \\ &= f(\mathcal{T} \cup \{i\}) - f(\mathcal{T}) \end{aligned}$$

■

**Theorem 10.2**

$f(\mathcal{S})$ is nondecreasing, i.e. $f(\mathcal{T}) \leq f(\mathcal{S})$ for all subsets $\mathcal{T} \subseteq \mathcal{S}$.

*Proof.*

$$f(\mathcal{S}) = \frac{1}{s} \sum_{r=1}^{s} A(r, \mathcal{S}) = \frac{1}{s} \sum_{r=1}^{s} \max_{j \in \mathcal{S}} a_j(r) \geq \frac{1}{s} \sum_{r=1}^{s} \max_{j \in \mathcal{T}} a_j(r) = f(\mathcal{T})$$

■

### 10.4.3 Proposed GridWatch-S Algorithm

We exploit this submodularity using an efficient greedy algorithm that starts from $\mathcal{S}$ as the empty set, and iteratively adds the best sensor to maximize $f(\mathcal{S})$, until the budget constraint $|S| = k$ is reached. Algorithm 10.2 describes our GridWatch-S algorithm.

### 10.4.4 Approximation Bound

The nondecreasing and submodularity properties of $f$ imply that Algorithm 10.2 achieves at least $1 - 1/e$ ($\approx 63\%$) of the value of the optimal sensor placement. Letting $\hat{\mathcal{S}}$ be the set returned by Algorithm 10.2, and $\mathcal{S}^*$ be the optimal set:

**Algorithm 10.2:** GridWatch-S sensor selection algorithm

**Input** : Graph $\mathcal{G}$, voltage $V_i(t)$, current $I_i(t)$, budget $k$, sensor scores $a_i(r)$ from (10.8)
**Output**: Chosen sensor set $\mathcal{S}$

$\mathcal{S} = \{\}$
Initialize $A(r) = 0 \; \forall \; r \in \mathcal{S}$ ▷Overall anomalousness is all zero since $\mathcal{S} = \{\}$
**while** $|\mathcal{S}| < k$ **do**
  **for** $i \notin \mathcal{S}$ **do**
    $\delta_i \leftarrow \frac{1}{s} \sum_{r=1}^{s} \mathbf{1}\{\max(A(r), a_i(r)) > c\}$ ▷Objective value if we added $i$ to $\mathcal{S}$
  **end**
  $i^* \leftarrow \arg\max_{i \notin \mathcal{S}} \delta_i$ ▷Greedily add the sensor that maximizes objective
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{i^*\}$
  $A(r) = \max(A(r), a_{i^*}(r)) \; \forall \; r \in \mathcal{S}$
**end**

**Theorem 10.3**

The objective value obtained by our algorithm is within $1 - 1/e$ of the optimal:

$$f(\hat{\mathcal{S}}) \geq (1 - 1/e) f(S^*) \tag{10.12}$$

*Proof.* This follows from [NWF78] since $f$ is nondecreasing and submodular. ■

## 10.5 Experiments

We design experiments to answer the following questions:

- **Q1. Anomaly Detection Accuracy:** on a fixed set of sensors, how accurate are the anomalies detected by GridWatch-S compared to baselines?
- **Q2. Sensor Selection:** how much does sensor selection using GridWatch-S improve the anomaly detection performance compared to other selection approaches?
- **Q3. Scalability:** how do our algorithms scale with the graph size?

Our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/code/`. Experiments were done on a 2.4 GHz Intel Core i5 Macbook Pro, 16 GB RAM running OS X 10.11.2.

**Data:** We use 2 graphs, `case2869` and `case9241`, which accurately represent different parts of the European high voltage network [ZMST11]. Dataset details are in Table 10.3.

Table 10.3: Datasets used

| Dataset name | Nodes | Generators | Edges | Transformers |
|---|---|---|---|---|
| case2869 [ZMST11] | 2869 | 327 | 2896 | 170 |
| case9241 [ZMST11] | 9241 | 1445 | 16049 | 1319 |

### 10.5.1 Q1. Anomaly Detection Accuracy

In this section, we compare GridWatch-D against baseline anomaly detection approaches, given a fixed set of sensors.

**Experimental Settings:** For each graph, the sensor set for all algorithms is chosen as a uniformly random set of nodes of various sizes (the sizes are plotted in the x-axis of Figure 10.3). Then, out of $480$ time ticks, we first sample $50$ random time ticks as the times when anomalies occur. In each such time tick, we deactivate a randomly chosen edge (i.e. no current can flow over that edge).

Using MatPower [ZMST11], we then generate voltage and current readings at each sensor. This requires an input time series of loads (i.e. real and reactive power at each node): we use load patterns estimated from real data [SHJ+17] recorded from the Carnegie Mellon University (CMU) campus for $20$ days from July 29 to August 17, 2016, scaled to a standard deviation of $0.3 \cdot \sigma$, with added Gaussian noise of $0.2 \cdot \sigma$, where $\sigma$ is the standard deviation of the original time series [SHJ+17].

This results in a time series of $480$ time ticks (hourly data from $20$ days), at each time recording the voltage at each sensor and the current at each edge adjacent to one of the sensors. Given this input, each algorithm then returns a ranking of the anomalies. We evaluate this using standard metrics, AUC (area under the ROC curve) and F-measure ($\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$), the latter computed on the top $50$ anomalies output by each algorithm.

**Baselines:** Dynamic graph anomaly detection approaches [AMF10, CHS12, MBR+13, APG+14, SKZ+15] cannot be used as they require graphs with fully observed edge weights. Moreover, detecting failed power lines with all sensors present can be done by simply checking if any edge has current equal to $0$, which is trivial. Hence, instead, we compare GridWatch-D to the following multidimensional time series based anomaly detection methods: Isolation Forests [LTZ08], Vector Autoregression (VAR) [Ham94], Local Outlier Factor (LOF) [BKNS00], and Parzen Window [Par62]. Each uses the currents and voltages at the given sensors as features. For VAR the norms of the residuals are used as anomaly scores; the remaining methods return anomaly scores directly.

For Isolation Forests, we use $100$ trees (following the scikit-learn defaults [PVG+11]). For VAR we select the order by maximizing AIC, following standard practice. For LOF we use $20$ neighbors (following scikit-learn defaults), and $20$ neighbors for Parzen Window.

Figure 10.3 shows that GridWatch-D outperforms the baselines, by $31\%$ to $42\%$ Area under the Curve (AUC) and $133\%$ to $383\%$ F-Measure. The gains in performance likely come from the

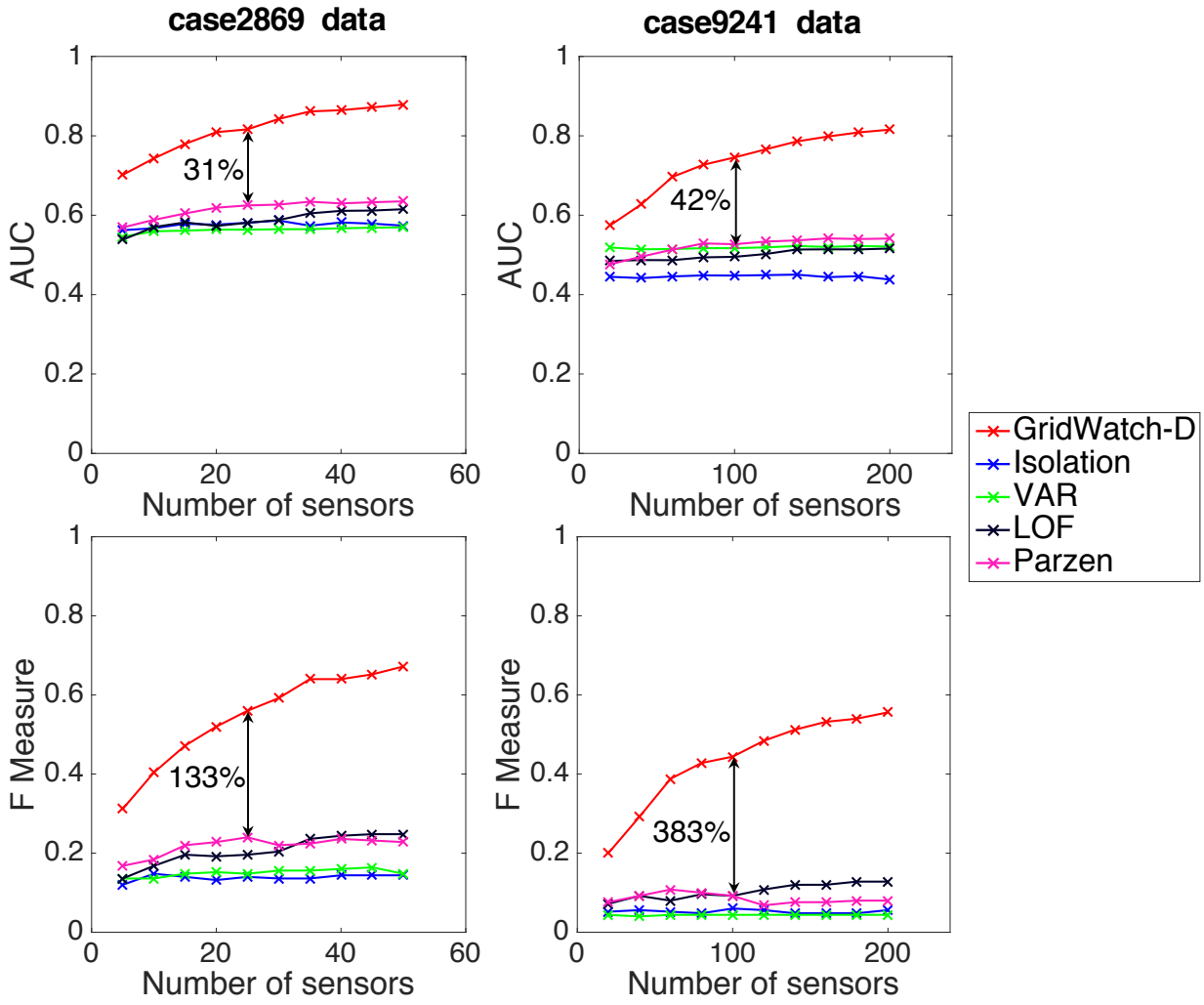



Figure 10.3: **GridWatch-D outperforms alternate anomaly detection methods:** Left plots are for `case2869`; right plots are for `case9241`.

use of the 3 domain-knowledge based detectors, which combine information from the currents surrounding each sensor in a way that makes it clearer when an anomaly occurs.

Further testing shows that GridWatch-D's 3 detectors all play a role: e.g. on `case2869`, for 50 sensors, GridWatch-D has F-measure 0.67, but only using single detectors 1, 2 or 3 (where detector 1 refers to the detector in Definition 10.1, and so on) gives F-measures of 0.51, 0.6 or 0.56 respectively.

### 10.5.2 Q2. Sensor Selection Quality

We now evaluate GridWatch-S. We use the same settings as in the previous sub-section, except that the sensors are now chosen using either GridWatch-S, or one of the following baselines. We then compute the anomaly detection performance of GridWatch-D as before on

each choice of sensors. For GridWatch-S we use $c = 15$. For our simulated data sizes, we assume 2000 anomalies and 480 normal scenarios.

**Baselines:** We use the following: randomly selected nodes (*Random*); highest degree nodes (*Degree*); nodes with highest total current in their adjacent edges (*MaxCurrent*); highest betweenness centrality [Fre78] nodes, i.e. nodes with the most shortest paths passing through them, thus being the most 'central' (*Betweenness*); a power-grid based Optimal PMU Placement algorithm using depth-first search (*OPP* [BMBA93]).

Figure 10.4 shows that GridWatch-S outperforms the baselines, by 18 to 19% Area under the Curve (AUC) and 59 to 62% F-Measure.

Figure 10.1b shows the GridWatch-S anomaly scores on the `case2869` data over time, when using the maximum 200 sensors, with red crosses where true anomalies exist. Spikes in anomaly score match very closely with the true anomalies. Threshold (blue line) is 4 IQRs above the mean (in log-space).

### 10.5.3 Q3. Scalability

Finally, we evaluate the scalability of GridWatch-D and GridWatch-S. To generate graphs of different sizes, we start with the IEEE 118-bus network [118], which represents a portion of the US power grid in 1962, and duplicate it $2, 4, \cdots, 20$ times. To keep our power grid connected, after each duplication, we add edges from each node to its counterpart in the last duplication; the parameters of each such edge are randomly sampled from those of the actual edges. We then run GridWatch-D and GridWatch-S using the same settings as the previous sub-section. Figure 10.5b shows that GridWatch-D and GridWatch-S scale linearly. The blue line is the best-fit regression line.

## 10.6 Conclusion

In this chapter, we proposed GridWatch-D, an online algorithm that accurately detects anomalies in power grid data. The main idea of GridWatch-D is to design domain-aware detectors that combine information at each sensor appropriately. We then proposed GridWatch-S, a sensor placement algorithm, which uses a submodular optimization objective. While our method could be technically applied to any type of graph-based sensor data (not just power grids), the choice of our detectors is motivated by our power grid setting. Hence, future work could study how sensitive various detectors are for detecting anomalies in graph-based sensor data from different domains.

Our contributions are as follows:

1. **Online anomaly detection:** we propose a novel, online anomaly detection algorithm, GridWatch-D that outperforms existing approaches.
2. **Sensor placement:** we construct an optimization objective for sensor placement, with the goal of maximizing the probability of detecting an anomaly. We show that this objective is submodular, which we exploit in our sensor placement algorithm.

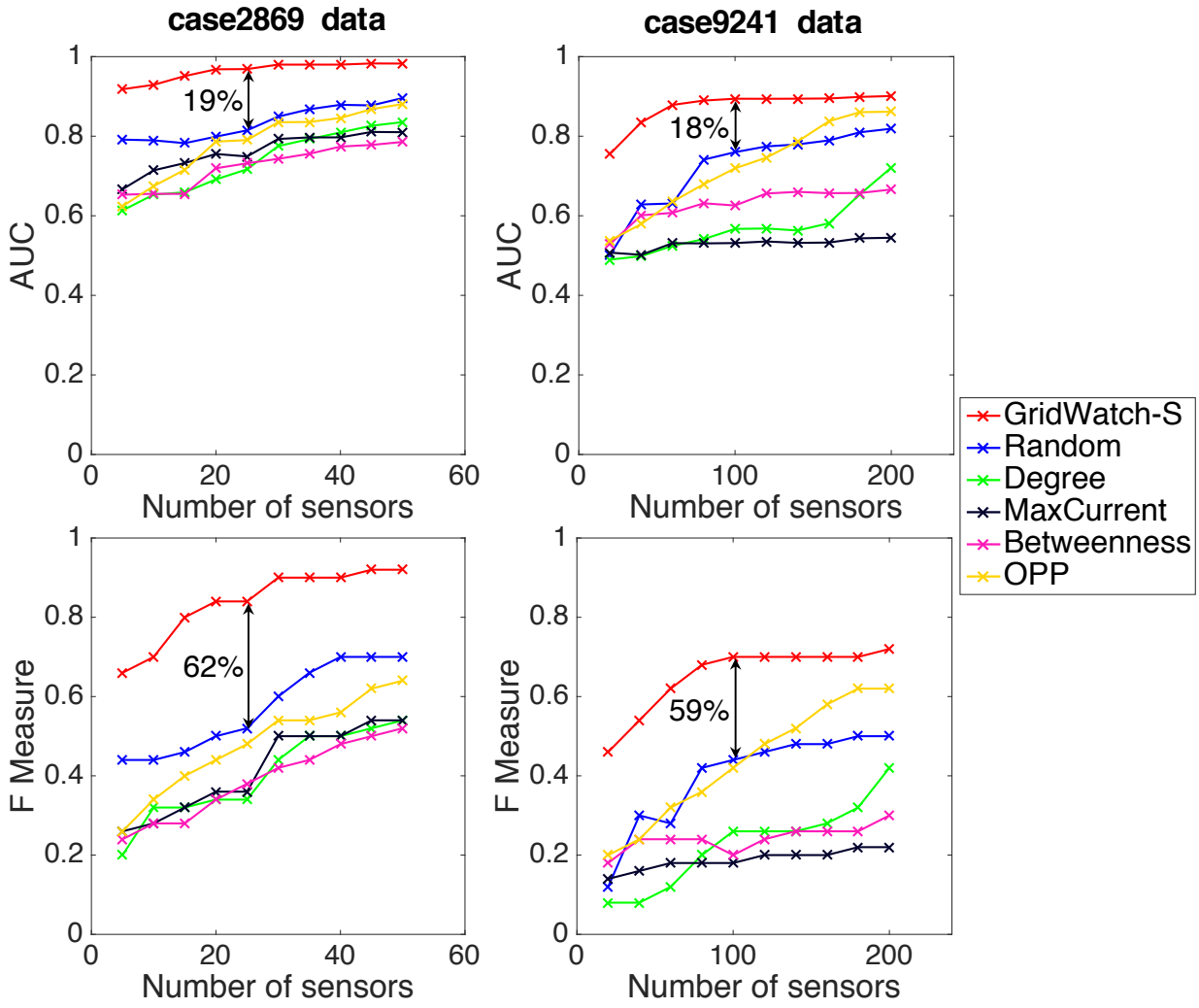
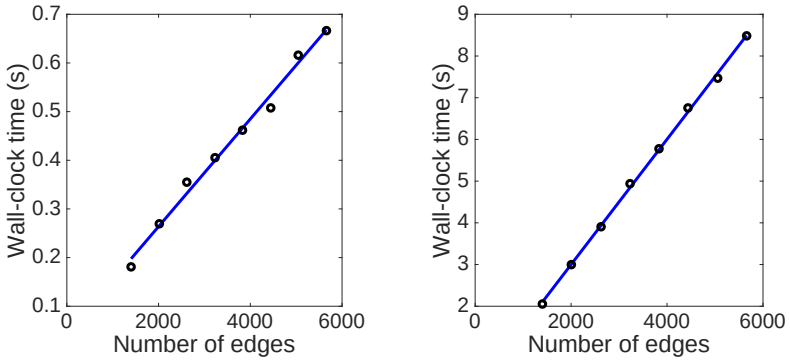



Figure 10.4: **GridWatch-S provides effective sensor selection:** sensor selection using GridWatch-S results in higher anomaly detection accuracy than other methods.

3. **Effectiveness:** Due to submodularity, GridWatch-S, our sensor placement algorithm is provably near-optimal. In addition, both our algorithms outperform existing approaches in accuracy by $59\%$ or more (F-measure) in experiments.
4. **Scalability:** Our algorithms scale linearly, and GridWatch-D is online, requiring bounded space and constant time per update.

**Reproducibility:** our code and data are publicly available at `http://www.andrew.cmu.edu/user/bhooi/code/`.




(a) GridWatch-D (b) GridWatch-S

Figure 10.5: Our algorithms scale linearly: wall-clock time of (a) GridWatch-D and (b) GridWatch-S against number of edges in $\mathcal{G}$.

# Chapter 11

# Conclusion and Future Work

## 11.1 Summary and Overarching Themes

The explosion of massive datasets, including large graphs and high-frequency time series, has created a "golden age" for data science. As a result, an overarching theme of much current research in anomaly detection is its move from classic techniques toward approaches that meet the needs of modern datasets and real-world applications, particularly in the aspects of scalability, handling streaming data, robustness against adversaries, as well as complex or heterogeneous data types. This thesis takes a step toward this goal by developing scalable algorithms for anomaly detection in graph and time series data, using approaches that allow us to deal with high-volume data streams in a scalable manner. We conclude by first summarizing the themes and approaches developed throughout this dissertation, and then extend these themes into the future.

Throughout this dissertation, we have described a number of different approaches, each designed for the needs of a particular application. How can we distill these methods into a coherent framework? The anomaly detection settings in this dissertation can be categorized based on both the **type of data** we have, as well as the **type of anomaly** we wish to detect, as follows.

- **Graphs: "How can we find unusual subgraphs?"**
  - *Anomalous Subgraphs (Non-Adversarial):* TellTail provides a probabilistic score for how measuring how abnormal a subgraph is.
  - *Anomalous Subgraphs (Adversarial):* Fraudar detects abnormal subgraphs in a more robust way, improving detection accuracy by up to *70%* F-measure over comparable baselines, and detecting a Twitter subgraph of more than *4000* accounts, a majority of which used follower-buying services.
- **Time Series: "How can we find unusual time ticks or subsequences?"**
  - *Categorical Data:* BirdNest performs anomaly detection in a categorical data setting arising from online rating systems.
  - *Numerical Data:* StreamCast considers a more traditional real-valued sensor data, aiming to find time ticks containing unusual events (e.g. power failures).

    - *Mixed Data:* For mixed categorical, numeric and ordinal data, we proposed an online nonparametric anomaly detection approach, BNB, that detects anomalies more accurately than baselines, by *61%* F-measure.
    - *Matrix-valued Data:* SMF performs forecasting and anomaly detection in a time series of matrices, capturing seasonality and drift.
- **Graphs with Sensors: "How can we find sudden events or changes located on a graph?"**
    - *Anomalous Subgraphs:* ChangeDAR detects changes occurring over a localized region of the graph, i.e. a subgraph.
    - *Anomalous Neighborhoods:* GridWatch performs anomaly detection in power grid graphs by detecting anomalous values along neighborhoods of nodes. In addition, GridWatch studies how to near-optimally select locations for new sensors to be placed on a power grid graph, improving the detection of component failures by *59%* or more F-measure.

## 11.2 Vision and Future Work

Moving forward, we aim to increase the usability of anomaly detection approaches for real-world applications. While this thesis focused on the challenges of graph and time series data, there are many more, equally important real-world challenges that need to be addressed: these include broadening the applicability of our approaches to detect more types of anomalies in different applications (e.g. trajectories, images and videos, and spatiotemporal data), designing more general approaches for collective anomaly detection (i.e. groups of anomalous points), as well as designing interpretable anomaly detection approaches.

**Reaching Out to Other Applications** Anomaly detection is broadly applicable and can find value in almost any domain, as long as significant and interesting deviations from the norm exist. Future work could study how these approaches could be applied and adapted to new settings. One particular application is to *spatiotemporal datasets*, where we have a set of sensors located spatially, where an explicit graph is not given, and has to be learned from data: e.g. air pollution sensors, weather sensors, detection of natural disasters and other phenomena. Another is to the detection of *flows or trajectories* on a graph, such as money laundering detection: which aims to detect fraudsters who use a series of bank transactions to hide the origin of funds arising from illegal activities, i.e. "cleaning dirty money." This problem shares similarities with graph fraud detection as in Fraudar (e.g. fake reviews), but requires more complex techniques capable of detecting 'flows' along the graph.

**Collective Anomalies** Collective anomaly detection aims to detect groups of anomalies behaving in a jointly unusual manner, e.g. lockstep behavior arising from spam or fraud. Dense subgraph detection on a bipartite graph, such as in our Fraudar algorithm, can be considered as a type of collective anomaly detection, but still only detects simple types of behavior, characterized by high or concentrated activity. However, more complex collective types of anomalies exist, particularly when temporal data is considered, as we have seen in Part II of this thesis:

such as the temporal patterns of fraudulent rating attacks studied in BirdNest, or the seasonality and drift patterns in SMF. Hence, future work could seek to integrate realistic temporal patterns into graph mining by carefully modelling how each node behaves temporally, and how neighboring nodes influence each others' temporal patterns. This could then lead to collective anomaly detection approaches beyond just dense subgraphs.

**Interpretability** So far, most methods focus on returning either the time of the anomaly, and / or the location in the graph where it occurs (as in ChangeDAR). Future, future work could build interpretable anomaly detection systems that provide further output about each anomaly. To be useful for practitioners, they should be designed around how practitioners analyze and respond to anomalies in real-world settings. A further step would involve an interactive anomaly detection system, which would allow pracitioners to query (e.g. based on keywords or other features in the dataset), or provide feedback (e.g. rejecting uninteresting anomalies) to seek more useful clarifying information, which the algorithm would need to efficiently update itself to respond to, in an online manner.

## 11.3 Closing Thoughts

We began this thesis with the quote:

> A capacity for surprise is an essential aspect of our mental life, and surprise itself is the most sensitive indication of how we understand our world and what we expect from it.
>
> Daniel Kahnemann, *Thinking, Fast and Slow*

To me, anomaly detection is ultimately about 'teaching computers to be surprised'. This is not an easy task: just as humans recognize deviations from normality without being taught what those deviations are, there is a similarly important role for algorithms which detect anomalies without labelled data (which is often scarce in practice); i.e. in an *unsupervised* manner. This is a fundamental challenge with numerous high-impact applications, which will lead to exciting future research.

# Bibliography


[118] Ieee power systems test case archive. http://www2.ee.washington.edu/research/pstca/. Accessed: 2017-03-15.

[AC09] Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, 2009.

[AC17] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *KIS*, 51(2):339–367, 2017.

[ACF13] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. *ICWSM*, 13:2–11, 2013.

[AF10] Leman Akoglu and Christos Faloutsos. Event detection in time series of mobile communication graphs. In *Army science conference*, pages 77–79, 2010.

[AG05] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Link-KDD*, pages 36–43. ACM, 2005.

[AGMF14] Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. Beyond blocks: Hyperbolic community detection. In *ECML-PKDD*, pages 50–65. Springer, 2014.

[AM07] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[AMF10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, pages 410–421. Springer, 2010.

[Ami11] S Massoud Amin. Us grid gets less reliable [the data]. *IEEE Spectrum*, 48(1):80–80, 2011.

[And10] Reid Andersen. A local algorithm for finding dense subgraphs. *Transaction on Algorithms*, 6(4):60, 2010.

[ANMJZ12] Hélio Almeida, Dorgival Guedes Neto, Wagner Meira Jr, and Mohammed J Zaki. Towards a better quality metric for graph cluster evaluation. *Journal of Information and Data Management*, 3(3):378, 2012.

[APG+14] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. Com2: fast automatic discovery of temporal ('comet') communities. In *PAKDD*, pages 271–283. Springer, 2014.

[ARS02] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *Latin American symposium on theoretical informatics*, pages 598–612. Springer, 2002.

[AZP11] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. Outlier detection in graph streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 399–409. IEEE, 2011.

[BCDGV14] Taposh Banerjee, Yu Christine Chen, Alejandro D Dominguez-Garcia, and Venugopal V Veeravalli. Power system line outage detection and identification - a quickest change detection approach. In *ICASSP*, pages 3450–3454. IEEE, 2014.

[BDFG03] Jérémie Bouttier, Philippe Di Francesco, and Emmanuel Guitter. Geodesic distance in planar graphs. *Nuclear Physics B*, 663(3):535–567, 2003.

[BDG+07] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 121–132. Springer, 2007.

[BDH74] August A Balkema and Laurens De Haan. Residual life time at great age. *The Annals of probability*, pages 792–804, 1974.

[BE76] Béla Bollobás and Paul Erdös. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 80, pages 419–427. Cambridge Univ Press, 1976.

[BGSLW93] Daniel Bienstock, Michel X Goemans, David Simchi-Levi, and David Williamson. A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(1-3):413–420, 1993.

[BH05] Dennis J Brueni and Lenwood S Heath. The pmu placement problem. *SIAM Journal on Discrete Mathematics*, 19(3):744–761, 2005.

[BJL+15] David M Bromberg, Marko Jereminov, Xin Li, Gabriela Hug, and Larry Pileggi. An equivalent circuit formulation of the power flow problem with current and voltage state variables. In *PowerTech, 2015 IEEE Eindhoven*, pages 1–6. IEEE, 2015.

[BJRL15] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[BK07] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*. IEEE, 2007.

[BKNS00] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[BL74] Vic Barnett and Toby Lewis. *Outliers in statistical data*. Wiley, 1974.

[BMBA93] TL Baldwin, L Mili, MB Boisen, and R Adapa. Power system observability with minimal phasor measurement placement. *IEEE Transactions on Power Systems*, 8(2):707–715, 1993.

[BMFS14] Alex Beutel, Kenton Murray, Christos Faloutsos, and Alexander J Smola. Cobafi: collaborative bayesian filtering. In *Proceedings of the 23rd international conference on World wide web*, pages 97–108. ACM, 2014.

[BPSDGA04] Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas. Models of social networks based on social distance attachment. *Physical review E*, 70(5):056122, 2004.

[Bro59] Robert Goodell Brown. *Statistical forecasting for inventory control*. McGraw/Hill, 1959.

[BV11] Kevin Bleakley and Jean-Philippe Vert. The group fused lasso for multiple change-point detection. *arXiv preprint arXiv:1106.4199*, 2011.

[BXG+13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130, 2013.

[CBK12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[CBTD01] Stuart Coles, Joanna Bawa, Lesley Trenner, and Pat Dorazio. *An introduction to statistical modeling of extreme values*, volume 208. Springer, 2001.

[CBVD18] Jose Cadena, Arinjoy Basak, Anil Vullikanti, and Xinwei Deng. Graph scan statistics with uncertainty, 2018.

[CCV17] Jose Cadena, Feng Chen, and Anil Vullikanti. Near-optimal and practical algorithms for graph scan statistics. In *SDM*. SIAM, 2017.

[CF15] Haeran Cho and Piotr Fryzlewicz. Multiple-change-point detection for high dimensional time series via sparsified binary segmentation. *JRSS(B)*, 77(2):475–507, 2015.

[CF16] Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39, 2016.

[Cha00] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[CHBA03] Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. Efficient immunization strategies for computer networks and populations. *Physical review letters*, 91(24):247901, 2003.

[CHC+07] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE TMC*, 6(6), 2007.

[CHS12] Zhengzhang Chen, William Hendrix, and Nagiza F Samatova. Community-based anomaly detection in evolutionary networks. *Journal of Intelligent In-*

*formation Systems*, 39(1):59–85, 2012.

[CN14] Feng Chen and Daniel B Neill. Non-parametric scan statistics for event detection and forecasting in heterogeneous social media graphs. In *KDD*. ACM, 2014.

[Coh08] Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16, 2008.

[Col30] Selwyn D Collins. The influenza epidemic of 1928-1929 with comparative data for 1918-1919. *American Journal of Public Health and the Nations Health*, 20(2):119–129, 1930.

[COL13] Freddy Chong Tat Chua, Richard J. Oentaryo, and Ee-Peng Lim. Modeling temporal adoptions using dynamic matrix factorization. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 91–100. IEEE, 2013.

[CPV01] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. *Communities of interest*. Springer, 2001.

[CSYP12] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *NSDI*, 2012.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[CTPK09] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven A Klooster. Detection and characterization of anomalies in multivariate time series. In *SDM*, pages 413–424. SIAM, 2009.

[CVK08] Saikat Chakrabarti, Ganesh K Venayagamoorthy, and Elias Kyriakides. Pmu placement for power system observability using binary particle swarm optimization. In *Power Engineering Conference, 2008. AUPEC'08. Australasian Universities*, pages 1–5. IEEE, 2008.

[dARF17] Miguel Ramos de Araujo, Pedro Manuel Pinto Ribeiro, and Christos Faloutsos. Tensorcast: Forecasting with context using coupled tensors (best paper award). In *Data Mining (ICDM), 2017 IEEE International Conference on*, pages 71–80. IEEE, 2017.

[DDD05] Frédéric Desobry, Manuel Davy, and Christian Doncarli. An online kernel change detection algorithm. *IEEE TSP*, 2005.

[DDGS08] Devesh Dua, Sanjay Dambhare, Rajeev Kumar Gajbhiye, and SA Soman. Optimal multistage scheduling of pmu placement: An ilp approach. *IEEE Transactions on Power delivery*, 23(4):1812–1820, 2008.

[DKA11] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *TKDD*, 5(2):10, 2011.

[DKM15] Robin Devooght, Nicolas Kourtellis, and Amin Mantrach. Dynamic matrix factorization with priors on unknown values. In *KDD*. ACM, 2015.

[DL05] Yi Ding and Xue Li. Time weight collaborative filtering. In *CIKM*. ACM, 2005.

[DLJL09] Dongsheng Duan, Yuhua Li, Yanan Jin, and Zhengding Lu. Community mining on dynamic weighted directed graphs. In *Proceedings of the 1st ACM international workshop on Complex networks meet information & knowledge management*, pages 11–18. ACM, 2009.

[DLT15] Thang N Dinh, Xiang Li, and My T Thai. Network clustering via maximizing modularity: Approximation algorithms and theoretical limits. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 101–110. IEEE, 2015.

[DVMP+08] Saverio De Vito, Ettore Massera, Marco Piga, Luca Martinotto, and Girolamo Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008.

[EKM99] Paul Embrechts, Claudia Kluppelberg, and Thomas Mikosch. Modelling extremal events. *British Actuarial Journal*, 5(2):465–465, 1999.

[ENLSS16] Lisette Espín Noboa, Florian Lemmerich, Philipp Singer, and Markus Strohmaier. Discovering and characterizing mobility patterns in urban spaces: A study of manhattan taxi data. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 537–542. International World Wide Web Conferences Steering Committee, 2016.

[FBC12] Song Feng, Ritwik Banerjee, and Yejin Choi. Syntactic stylometry for deception detection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 171–175. Association for Computational Linguistics, 2012.

[FCYJMT+15] Alceu Ferraz Costa, Yuto Yamaguchi, Agma Juci Machado Traina, Caetano Traina Jr, and Christos Faloutsos. Rsc: Mining and modeling temporal activity in social media. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 269–278. ACM, 2015.

[FLG00] Gary William Flake, Steve Lawrence, and C Lee Giles. Efficient identification of web communities. In *KDD*, pages 150–160. ACM, 2000.

[FMS14] Klaus Frick, Axel Munk, and Hannes Sieling. Multiscale change point inference. *JRSS(B)*, 76(3):495–580, 2014.

[Fre78] Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

[Fry14] Piotr Fryzlewicz. Wild binary segmentation for multiple change-point detection. *The Annals of Statistics*, 42(6):2243–2281, 2014.

[GC13] Derek Greene and Pádraig Cunningham. Producing a unified graph representation from multiple social network views. In *ACMWeb*, pages 118–121. ACM, 2013.

[GDY+17] Shaghayegh Gharghabi, Yifei Ding, Chin-Chia Michael Yeh, Kaveh Kamgar, Liudmila Ulanova, and Eamonn Keogh. Matrix profile viii: Domain agnostic online semantic segmentation at superhuman performance levels. In *ICDM*. IEEE,

2017.

[GGF14] Stephan Günnemann, Nikou Günnemann, and Christos Faloutsos. Detecting anomalies in dynamic rating data: A robust probabilistic model for rating evolution. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 841–850. ACM, 2014.

[GGMP04] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *VLDB Endowment*, pages 576–587, 2004.

[GHK+14] Oliver Göbel, Martin Hoefer, Thomas Kesselheim, Thomas Schleiden, and Berthold Vöcking. Online independent set beyond the worst-case: Secretaries, prophets, and periods. In *International Colloquium on Automata, Languages, and Programming*, pages 508–519. Springer, 2014.

[GKW06] Matthew E Gaston, Miro Kraetzl, and Walter D Wallis. Using graph diameter for change detection in dynamic networks. *Australasian Journal of Combinatorics*, 35:299, 2006.

[Gol84] Andrew V Goldberg. *Finding a maximum density subgraph*. Technical Report, 1984.

[GPW+] Zhongshu Gu, Kexin Pei, Qifan Wang, Luo Si, Xiangyu Zhang, and Dongyan Xu. Leaps: Detecting camouflaged attacks with statistical learning guided by program analysis.

[Gri93] Scott D Grimshaw. Computing maximum likelihood estimates for the generalized pareto distribution. *Technometrics*, 35(2):185–191, 1993.

[GTV11] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 87–93. IEEE, 2011.

[GVK+12] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Phani Gummadi. Understanding and combating link farming in the twitter social network. In *21st WWW*, pages 61–70. ACM, 2012.

[GW95] Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[HA18] Natascha Harth and Christos Anagnostopoulos. Edge-centric efficient regression analytics. In *2018 IEEE EDGE*, pages 93–100. IEEE, 2018.

[Hal04] Magnús M Halldórsson. Approximations of weighted independent set and hereditary subset problems. In *Graph Algorithms And Applications 2*, pages 3–18. World Scientific, 2004.

[Ham94] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[HF19] Bryan Hooi and Christos Faloutsos. Branch and border: Partition-based change detection in multivariate time series. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2019.

[HIS15] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. A nearly-linear time framework for graph-structured sparsity. In *ICML*, pages 928–937, 2015.

[HK+07] Rob J Hyndman, Yeasmin Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics, 2007.

[HL04] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[HNB16] David Hallac, Peter Nystrup, and Stephen Boyd. Greedy gaussian segmentation of multivariate time series. *arXiv preprint arXiv:1610.07435*, 2016.

[Hof04] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[HPS01] Henrique Steinherz Hippert, Carlos Eduardo Pedreira, and Reinaldo Castro Souza. Neural networks for short-term load forecasting: A review and evaluation. *IEEE Transactions on power systems*, 16(1):44–55, 2001.

[HS03] Shyh-Jier Huang and Kuang-Rong Shih. Short-term load forecasting via arma model identification including non-gaussian process considerations. *IEEE Transactions on power systems*, 18(2):673–679, 2003.

[HSB+16a] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. Birdnest: Bayesian inference for ratings-fraud detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 495–503. SIAM, 2016.

[HSB+16b] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 895–904. ACM, 2016.

[HSLF19] Bryan Hooi, Kijung Shin, Shenghua Liu, and Christos Faloutsos. Smf: Drift-aware matrix factorization with seasonal patterns. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2019.

[HSP+18] Bryan Hooi, Hyun Ah Song, Amritanshu Pandey, Marko Jereminov, Larry Pileggi, and Christos Faloutsos. Streamcast: Fast and online mining of power grid time sequences. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 531–539. SIAM, 2018.

[HT93] Clifford M Hurvich and Chih-Ling Tsai. A corrected akaike information criterion for vector autoregressive model selection. *Journal of time series analysis*, 14(3):271–279, 1993.

[JBC+15] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A general suspiciousness metric for dense blocks in multimodal data. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 781–786. IEEE, 2015.

[JBC+16] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. Spotting suspicious behaviors in multimodal data: A general metric and algorithms. *TKDE*, 28(8):2187–2200, 2016.

[JCB+14a] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catchsync: catching synchronized behavior in large directed graphs. In *20th KDD*, pages 941–950. ACM, 2014.

[JCB+14b] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. In *Advances in Knowledge Discovery and Data Mining*, pages 126–138. Springer, 2014.

[JL08] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 219–230. ACM, 2008.

[JLL10] Nitin Jindal, Bing Liu, and Ee-Peng Lim. Finding unusual review patterns using unexpected rules. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1549–1552. ACM, 2010.

[JNIH14] Michael Jones, Daniel Nikovski, Makoto Imamura, and Takahisa Hirata. Anomaly detection in real-valued multidimensional time series. In *International Conference on Bigdata/Socialcom/Cybersecurity. Stanford University, ASE*. Citeseer, 2014.

[JPS+17] Marko Jereminov, Amritanshu Pandey, Hyun Ah Song, Bryan Hooi, Christos Faloutsos, and Larry Pileggi. Linear load model for robust power system analysis. In *IEEE PES Innovative Smart Grid Technologies*, page (submitted). IEEE, 2017.

[JSB+05] Brad Jackson, Jeffrey D Scargle, David Barnes, Sundararajan Arabhi, Alina Alt, Peter Gioumousis, Elyus Gwin, Paungkaew Sangtrakulcharoen, Linda Tan, and Tun Tao Tsai. An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, 12(2):105–108, 2005.

[JXWC12] Peirong Ji, Di Xiong, Peng Wang, and Juan Chen. A study on exponential smoothing model for load forecasting. In *Power and Energy Engineering Conference (APPEEC), 2012 Asia-Pacific*, pages 1–4. IEEE, 2012.

[K+60] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[Kaz14] Nicholas D Kazarinoff. *Analytic inequalities*. Courier Corporation, 2014.

[KB09] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[KCG+15] Stanley IM Ko, Terence TL Chong, Pulak Ghosh, et al. Dirichlet process hidden markov multiple change-point model. *Bayesian Analysis*, 10(2):275–296, 2015.

[KCHP01] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *ICDM*. IEEE, 2001.

[KE11] Daniel Kahneman and Patrick Egan. *Thinking, fast and slow*, volume 1. Farrar, Straus and Giroux New York, 2011.

[KFE12] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. Optimal detection of changepoints with a linear computational cost. *JASA*, 107(500):1590–1598, 2012.

[KGW12] Vassilis Kekatos, Georgios B Giannakis, and Bruce Wollenberg. Optimal placement of phasor measurement units via convex relaxation. *IEEE Transactions on power systems*, 27(3):1521–1530, 2012.

[KHH+05] Martin Kulldorff, Richard Heffernan, Jessica Hartman, Renato Assunçao, and Farzad Mostashari. A space–time permutation scan statistic for disease outbreak detection. *PLoS medicine*, 2(3):e59, 2005.

[KK98] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[Kle99] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[KLLVH07] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11(1):1–27, 2007.

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *19th WWW*, pages 591–600. ACM, 2010.

[kon17] Wikipedia links, portuguese network dataset – KONECT, April 2017.

[Kor08] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[Kor10] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[KP09] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 508–520. Springer, 2009.

[KQR16] Ivan Kojadinovic, Jean-François Quessy, and Tom Rohmer. Testing the constancy of spearmanâĂŹs rho in multivariate time series. *Annals of the Institute of Statistical Mathematics*, 68(5):929–954, 2016.

[Kun13] Jérôme Kunegis. Konect: the koblenz network collection. In *WWW*, pages 1343–1350. ACM, 2013.

[KVF13] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *SDM*, pages 162–170. SIAM, 2013.

[KVV04] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *JACM*, 51(3):497–515, 2004.

[LADW05] Lun Li, David Alderson, John C Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2(4):431–523, 2005.

[LC98] Erich Leo Lehmann and George Casella. *Theory of point estimation*, volume 31. Springer Science & Business Media, 1998.

[LCK+10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

[LCYL17] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. *arXiv preprint arXiv:1703.07015*, 2017.

[Ley02] Michael Ley. The dblp computer science bibliography: Evolution, research issues, perspectives. In *SPIRE*, pages 1–10. Springer, 2002.

[LH07] Xiaolei Li and Jiawei Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 447–458. VLDB Endowment, 2007.

[LHK10] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1361–1370. ACM, 2010.

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM TKDD*, 1(1):2, 2007.

[LKG+07] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429. ACM, 2007.

[LLJ02] X Liu and RG Lathrop Jr. Urban change detection based on an artificial neural network. *International Journal of Remote Sensing*, 23(12):2513–2518, 2002.

[LLM10] Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010.

[LNI11] Qiao Li, Rohit Negi, and Marija D Ilić. Phasor measurement units placement for power system state estimation: A greedy approach. In *Power and Energy Society General Meeting, 2011 IEEE*, pages 1–8. IEEE, 2011.

[LNJ+10] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 939–948. ACM, 2010.

[LRBP09] Julie Letchner, Christopher Re, Magdalena Balazinska, and Matthai Philipose. Access methods for markovian streams. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 246–257. IEEE, 2009.

[LS01] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *ICDM*. IEEE, 2008.

[LYCS13] Song Liu, Makoto Yamada, Nigel Collier, and Masashi Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, 2013.

[LZG+15] Xuan Liang, Tao Zou, Bin Guo, Shuo Li, Haozhe Zhang, Shuyi Zhang, Hui Huang, and Song Xi Chen. Assessing beijing's pm2. 5 pollution: severity, weather impact, apec and winter heating. *Proc. R. Soc. A*, 471(2182):20150257, 2015.

[MA99] Fernando H Magnago and Ali Abur. A unified approach to robust meter placement against loss of measurements and branch outages. In *Power Industry Computer Applications, 1999. PICA'99. Proceedings of the 21st 1999 IEEE International Conference*, pages 3–8. IEEE, 1999.

[MAB13] José R Martí, Hamed Ahmadi, and Lincol Bashualdo. Linear power-flow formulation based on a voltage-dependent load model. *IEEE Transactions on Power Delivery*, 28(3):1682–1690, 2013.

[Mar63] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[MBR+13] Misael Mongiovi, Petko Bogdanov, Razvan Ranca, Evangelos E Papalexakis, Christos Faloutsos, and Ambuj K Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *SDM*. SIAM, 2013.

[MBS13] Misael Mongiovi, Petko Bogdanov, and Ambuj K Singh. Mining evolving network processes. In *ICDM*, pages 537–546. IEEE, 2013.

[Min00] Thomas Minka. Estimating a dirichlet distribution, 2000.

[MJ14] David S Matteson and Nicholas A James. A nonparametric approach for multiple change point analysis of multivariate data. *JASA*, 109(505):334–345, 2014.

[MKKSM13] Arash Molavi Kakhki, Chloe Kliman-Silver, and Alan Mislove. Iolaus: Securing online content rating systems. In *Proceedings of the 22nd international conference on World Wide Web*, pages 919–930. International World Wide Web Conferences Steering Committee, 2013.

[ML13] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[MS16] Yasuko Matsubara and Yasushi Sakurai. Regime shifts in streams: Real-time forecasting of co-evolving time sequences. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1045–1054. ACM, 2016.

[MSN13] Edward McFowland, Skyler Speakman, and Daniel B Neill. Fast generalized subset scan for anomalous pattern detection. *JMLR*, 14(1):1533–1561, 2013.

[New06] Mark EJ Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

[NWF78] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[OCCH11] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 309–319. Association for Computational Linguistics, 2011.

[OR15] Peter Orbanz and Daniel M Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE TPAMI*, 37(2):437–461, 2015.

[PAISM14] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *20th KDD*, pages 1346–1355. ACM, 2014.

[Par62] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[PBS47] James E Perkins, Anne M Bahlke, and Hilda Freeman Silverman. Effect of ultra-violet irradiation of classrooms on spread of measles in large rural central schools preliminary report. *American Journal of Public Health and the Nations Health*, 37(5):529–537, 1947.

[PC15] Leto Peel and Aaron Clauset. Detecting change points in the large-scale structure of evolving networks. In *AAAI*, pages 2914–2920, 2015.

[PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210. ACM, 2007.

[PDGM10] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[pem18] Caltrans performance measurement system. http://pems.dot.ca.gov/, 2018. Accessed: 2018-04-08.

[Pin05] Brandon Pincombe. Anomaly detection in time series of graphs using arma processes. *Asor Bulletin*, 24(4):2, 2005.

[PJL+16] Amritanshu Pandey, Marko Jereminov, Xin Li, Gabriela Hug, and Larry Pileggi. Aggregated load and generation equivalent circuit models with semi-empirical data fitting. In *Green Energy and Systems Conference (IGSEC), 2016 IEEE*, pages 1–6. IEEE, 2016.

[PPL91] JH Park, YM Park, and KY Lee. Composite modeling for adaptive short-term load forecasting. *IEEE Transactions on Power Systems*, 6(2):450–457, 1991.

[PSS+10] BA Prakash, M Seshadri, A Sridharan, S Machiraju, and C Faloutsos. Eigenspokes: Surprising patterns and community structure in large graphs. *PAKDD, 2010a*, 84, 2010.

[PSV02] Romualdo Pastor-Satorras and Alessandro Vespignani. Immunization of complex networks. *Physical Review E*, 65(3):036104, 2002.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[QAWZ15] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *KDD*. ACM, 2015.

[QLCZ15] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. Locally densest subgraph discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 965–974. ACM, 2015.

[QLW16] Dequan Qi, Zhonghua Li, and Zhaojun Wang. On-line monitoring data quality of high-dimensional data streams. *JSCS*, 86(11):2204–2216, 2016.

[RAGT14] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. Event detection in activity networks. In *KDD*. ACM, 2014.

[RCC+04] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *PNAS*, 101(9):2658–2663, 2004.

[RGBK15] Mahshid Zomorodi Rad, Saeed Rahati Ghuchani, Kambiz Bahaadinbeigy, and Mohammad Mahdi Khalilzadeh. Real time recognition of heart attack in a smart phone. *Acta Informatica Medica*, 23(3):151, 2015.

[RHSS16] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, and Nagiza F Samatova. A scalable approach for outlier detection in edge streams using sketch-based approximations. In *SDM*, pages 189–197. SIAM, 2016.

[RMB+10] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM TOSN*, 6(2):13, 2010.

[RPUW07] Chawasak Rakpenthai, Suttichai Premrudeepreechacharn, Sermsak Uatrongjit, and Neville R Watson. An optimal pmu placement method against measurement loss and branch outage. *IEEE transactions on power delivery*, 22(1):101–107, 2007.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *ACM SIGMOD Record*, volume 29, pages 427–438. ACM, 2000.

[RUUU12] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 1. Cambridge University Press Cambridge, 2012.

[SBGF14] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting suspicious link behavior with fbox: an adversarial perspective. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 959–964. IEEE, 2014.

[SBH+15] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnemann, Makhija, Mohit Kumar, and Christos Faloutsos. Edgecentric: Anomaly detection in edge-attributed networks. *arXiv preprint*, 2015.

[SCM+14] A Selakov, D Cvijetinović, L Milović, S Mellon, and D Bekut. Hybrid pso–svm method for short-term load forecasting during periods with significant temperature variations in city of burbank. *Applied Soft Computing*, 16:80–88, 2014.

[SDLF+17] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.

[SERF16] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms. In *ICDM*, 2016.

[SF78] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*. ACM, 2007.

[SHJ+17] Hyun Ah Song, Bryan Hooi, Marko Jereminov, Amritanshu Pandey, Larry Pileggi, and Christos Faloutsos. Powercast: Mining and forecasting power grid sequences. In *ECML-PKDD*, pages 606–621. Springer, 2017.

[SHK+10] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, 2010.

[SHKF17] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. Densealert: Incremental dense-subtensor detection in tensor streams. In *Proceedings of the 23rd*

*ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1057–1066. ACM, 2017.

[SK74] Andrew Jhon Scott and M Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, pages 507–512, 1974.

[SKKR00] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[SKZ+15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD*, pages 1055–1064. ACM, 2015.

[SLR16] Emma M. Stewart, Anna Liao, and Ciaran Roberts. Open Îijpmu: A real world reference distribution micro-phasor measurement unit data set for research and application development. 10/2016 2016.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, 2000.

[SM08] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[Smi87] Richard L Smith. Estimating tails of probability distributions. *The annals of Statistics*, pages 1174–1207, 1987.

[SPV14] John Z. Sun, Dhruv Parthasarathy, and Kush R Varshney. Collaborative kalman filtering for dynamic matrix factorization. *IEEE Trans. Signal Processing*, 62(14):3499–3509, 2014.

[SRS16] James Sharpnack, Alessandro Rinaldo, and Aarti Singh. Detecting anomalous activity on networks with the graph fourier scan statistic. *IEEE Transactions on Signal Processing*, 64(2):364–379, 2016.

[SSR13] James Sharpnack, Aarti Singh, and Alessandro Rinaldo. Changepoint detection over graphs with the spectral scan statistic. In *Artificial Intelligence and Statistics*, pages 545–553, 2013.

[STF06] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383. ACM, 2006.

[sup] Supplementary document. https://www.dropbox.com/sh/il46i2cgqfd1vrk/AABazVadhlLAp-Z8h0kKoZf5a?dl=0.

[sup18] Supplementary document. http://www.andrew.cmu.edu/user/bhooi/bnb/supplement.pdf, 2018.

[Svi04] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[TBG+13] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *SIGKDD*, pages 104–112, 2013.

[THF17] Tsubasa Takahashi, Bryan Hooi, and Christos Faloutsos. Autocyclone: Automatic mining of cyclic online activities with robust tensor factorization. In *WWW*, 2017.

[tlc13] Background on the boro taxi program. http://www.nyc.gov/html/tlc/html/passenger/shl_passenger_background.shtml, 2013. Accessed: 2017-01-25.

[TMLS09] Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient online content voting. In *NSDI*, volume 9, pages 15–28, 2009.

[Tso15] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *24th WWW*, pages 1122–1132. International World Wide Web Conferences Steering Committee, 2015.

[TTL11] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *IJCAI*, 2011.

[UC96] Graham Upton and Ian Cook. *Understanding statistics*. Oxford University Press, 1996.

[VD06] Sankar Virdhagriswaran and Gordon Dakin. Camouflaged fraud detection in domains with complex relationships. In *12th KDD*, pages 941–947. ACM, 2006.

[Vit85] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[vLDBSM16] Matthijs van Leeuwen, Tijl De Bie, Eirini Spyropoulou, and Cédric Mesnage. Subjective interestingness of subgraph patterns. *Machine Learning*, 105(1):41–75, 2016.

[VPGJ+13] Willem G Van Panhuis, John Grefenstette, Su Yon Jung, Nian Shong Chok, Anne Cross, Heather Eng, Bruce Y Lee, Vladimir Zadorozhny, Shawn Brown, Derek Cummings, et al. Contagious diseases in the united states from 1888 to the present. *NEJM*, 2013.

[VS10] Alireza Vahdatpour and Majid Sarrafzadeh. Unsupervised discovery of abnormal activity occurrences in multi-dimensional time series, with applications in wearable systems. In *SDM*, pages 641–652. SIAM, 2010.

[WGD06] Baoning Wu, Vinay Goel, and Brian D Davison. Propagating trust and distrust to demote web spam. *MTW*, 190, 2006.

[Win60] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[WLZ11] Hongning Wang, Yue Lu, and ChengXiang Zhai. Latent aspect rating analysis without aspect keyword supervision. In *17th KDD*, pages 618–626. ACM, 2011.

[WXLY11] Guan Wang, Sihong Xie, Bing Liu, and Philip S Yu. Review graph based online store review spammer detection. In *Data mining (icdm), 2011 ieee 11th international conference on*, pages 1242–1247. IEEE, 2011.

[XCH+10] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *SDM*. SIAM, 2010.

[XJRN03] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, pages 521–528, 2003.

[XWLY12] Sihong Xie, Guan Wang, Shuyang Lin, and Philip S Yu. Review spam detection via temporal pattern discovery. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 823–831. ACM, 2012.

[XY13] Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013.

[YA15] Junting Ye and Leman Akoglu. Discovering opinion spammer groups by network footprints. In *Machine Learning and Knowledge Discovery in Databases*, pages 267–282. Springer, 2015.

[YGKX08] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 3–17. IEEE, 2008.

[YH18] Hiroki Yanagisawa and Satoshi Hara. Discounted average degree density metric and new algorithms for the densest subgraph problem. *Networks*, 71(1):3–15, 2018.

[YJYC17] Subin Yi, Janghoon Ju, Man-Ki Yoon, and Jaesik Choi. Grouped convolutional neural networks for multivariate time series. *arXiv preprint arXiv:1703.09938*, 2017.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 36(4):267–278, 2006.

[YTWM04] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *DMKD*, 8(3):275–300, 2004.

[Yul19] George Udny Yule. *An introduction to the theory of statistics*. C. Griffin, limited, 1919.

[ZGP12] Yue Zhao, Andrea Goldsmith, and H Vincent Poor. On pmu location selection for line outage detection in wide-area transmission networks. In *Power and Energy Society General Meeting, 2012 IEEE*, pages 1–8. IEEE, 2012.

[ZLMZ05] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. Collecting the internet as-level topology. *ACM SIGCOMM Computer Communication Review*, 35(1):53–61, 2005.

[ZMST11] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on power systems*, 26(1):12–19, 2011.

[ZWWB15] Pei Zhang, Xiaoyu Wu, Xiaojun Wang, and Sheng Bi. Short-term load forecasting based on big data technologies. *CSEE Journal of Power and Energy Systems*, 1(3):59–67, 2015.

[ZWZJ15] Changliang Zou, Zhaojun Wang, Xuemin Zi, and Wei Jiang. An efficient online monitoring method for high-dimensional data streams. *Technometrics*, 57(3):374–387, 2015.