Mining Anomalies using Static and Dynamic Graphs

Dhivya Eswaran

May 2020

CMU-CS-20-108

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

Thesis Committee:
Christos Faloutsos, Chair
Aarti Singh
Zico Kolter
Nina Mishra, Amazon

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

This research was sponsored by the National Science Foundation under grant numbers IIS1247489 and IIS1408924, the Space and Naval Warfare Systems Center under grant number N6600112C2008, the Army under grant number W911NF0920053, Boeing under grant number A0192142015002U, and the PNC Center for Financial Services Innovation. 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, graphs, time-series, unsupervised algorithms, semi-supervised algorithms, streaming algorithms

To my mother, and grandparents, who have always been there for me. I love you all.

Abstract

Detection of anomalies, i.e. rare or unusual patterns, is a pressing problem in a number of contexts such as security, health care, finance and the web. Anomalies such as review fraud and network intrusion attacks encode suspicious, fraudulent or malicious behavior and do not just influence people into making sub-optimal decisions but also steadily erode their trust in businesses. As such, algorithms to detect ongoing anomalies and warn against upcoming anomalies have high impact for businesses and end-users alike.

This thesis considers the problem of anomaly detection by developing principled, scalable algorithms that detect unusual behavior or events by leveraging connectivity and temporal information. These approaches are useful for large dynamic complex datasets having strong relational and temporal characteristics, with multiple entities interacting with each other and also evolving over time. Such datasets are generated in a multitude of diverse contexts today, with examples ranging from e-commerce logs to online social networks to the internet-of-things.

The first half of the thesis focuses on anomaly detection in graphs where only static connectivity information is known. Given a graph, and a few labeled vertices, how can we infer the labels for the remaining vertices? For example, how can we spot all fake user accounts on Amazon or Facebook from a small set of manually labeled honest and fake accounts? Compared to existing literature, our work leverages three key properties of real-world graphs, namely, heterogeneity in vertex and edge types, skewed degree distributions, and higher-order structures, to yield more accurate vertex labeling. The proposed algorithms have closed-form solutions, rigorous convergence guarantees, can be efficiently implemented using sparse matrix operations, and scale linearly with graph size.

The second half of the thesis focuses on mining anomalies from data where the connectivity structure evolves over time. In many settings, especially those relating to security and health care, the value of a newfound or anticipated anomaly lies in the moment, and not later. Thus, given a time-evolving graph (explicit or implicit), how can we detect anomalies or events in near real-time, or perhaps even early warn before their occurrence? Our algorithms can detect anomalous graph footprints such as sudden appearance or disappearance of dense subgraphs and bridge edges in near real-time, by only storing a small synopsis of the graph seen so far and requiring no supervision. We also show how to infer state-transition graph from time series data in an online manner and use that to early warn against user-labeled anomalies such as adverse medical conditions.

Throughout the thesis, a strong emphasis is placed on algorithms which are not only (a) *effective* in practice, but are also (b) *efficient*, processing millions of edges in under a few seconds on a stock laptop, and are (c) *principled*, can be reasoned about rigorously, yielding theoretical guarantees for inference, detection, or leveraging data-related insights. We demonstrate the efficacy of our algorithms in a range of applications from social networks and e-commerce to security and health care.

Acknowledgments

To my mother, my life-long mentor and cheerleader, the bravest and the most independent woman I've ever known. My biggest thank you to her for encouraging me to pursue science, to strive for and achieve greater and greater heights.

To my grandparents, for their unconditional love and support, and their confidence in me. It was their kind words, laughter, and stories every morning that kept me going when nothing else did. I will always be their *chellam*.

To my advisor Christos Faloutsos, for his never-ending words of encouragement, for his infectious enthusiasm and positivity, for his approachability and humility, for always taking the time and effort to meet with me, whenever I needed. I will always cherish the many pictionaries, charades and pizzas we shared.

To Nina Mishra and Paul Bennett, for their invaluable mentorship and guidance. My life and career would not have been the same if not for my many interactions with them. They inspire me and challenge me at every turn, instill in me the thirst for knowledge and drive my pursuit of rigorous, practical, and impactful science.

To Neil Shah, Vagelis Papalexakis, Danai Koutra and Alex Beutel for their advice, encouragement, and for calling me their *little sister*. To Leman Akoglu, Stephan Guennemann, Bryan Hooi, Srijan Kumar, Reihaneh Rabbany, and Kijung Shin, from whom I have learned a lot through collaborations. To Hyun Ah Song, Hemank Lamba, Shubhranshu Shekhar, Namyong Park, Minji Yoon, and Catalina Vajiac for the many experiences and conversations we shared. To Deborah Cavlovich, Alison Day, Tony Mareino, Ann Stetser, and Marilyn Walgora for their excellent administrative skills and for the way they simplified my life.

Finally, to my support circle of dearest friends. To Rajarshi Sengupta and Vishwanath Saragadam, for being my go-to pillars of support during these intense and transformational years. Thank you for encouraging me, actively checking on my progress, cheering me up, cooking for me, and most of all, just being there for me, near or far. To Kartik Gupta for always keeping me on my toes, and for the clarity our conversations bring into my life. To Abhinav Garlapati, Aishwarya Padmakumar, and Aarati Kakaraparthy whom I can always count on for support. To Arun Kannawadi and Lekha Mohan, for mafia nights, travels, and unexpected friendships that blossomed and persisted despite all odds. To Arun Suggala and Deepoo Kumar, for the many runs we undertook together; their patience and perseverance never ceases to amaze me. To Bhavya Balu, Travis Dick, Varun Gangal, Qin Gu, Saurabh Kadekodi, Adithya Phillip, Jagannath Saragadam, Anchit Sood, Deepak Thipeswamy and the many more people I interacted with through Indian Graduate Student Association, summer internships at Facebook, Amazon and Microsoft Research, as well as other avenues at Carnegie Mellon University.

Thank you all for keeping me sane, and adding incredible experiences, meaning, richness, fun and excitement to the past five years of my life.

Contents

Part I

Introduction

Chapter 1

Introduction

Large-scale data mining has become a focal point of research in computer sciences and social sciences in recent years. Statistics[1] show that over 2.5 quintillion bytes of new data is generated worldwide every day from commercial transactions, social networks, system log data, electronic sensors and more. Much of this data has strong relational and temporal characteristics, capturing multiple entities interacting with each other and also evolving over time. Thus, it can be naturally modeled as a graph. Our thesis provides effective and scalable algorithms to analyze and garner insights from graph data, and specifically, detect anomalies or deviations from typical patterns.

1.1 Problem

Anomaly detection is a pressing problem for various critical tasks such as security, finance and the web. Anomalies–such as review or rating fraud–encode suspicious, fraudulent or malicious behavior and do not just influence people into making sub-optimal decisions but also steadily erode their trust in businesses. As such, algorithms to detect ongoing anomalies and warn against upcoming anomalies have high impact for businesses and end-users alike.

An immediate challenge in detecting anomalies lies in defining what anomalies or outliers are. One of the earliest definitions dates back to 1980, when Hawkins [Haw80] observes: "*An outlier is an observation that differs so much from other observations as to arouse suspicion that it was generated by a different mechanism*". The decided vagueness of this definition makes anomaly mining a challenging and open-ended problem. A more useful and meaningful definition of anomaly is possible only under a given context or application. Anomalies in our work are motivated by online social networks, e-commerce, communication, transportation and the internet-of-things, to name a few.

[1] https://leftronic.com/big-data-statistics/

1.2 Organization

This thesis is organized in two parts. In the first part, we focus on the case where only static connectivity information is known, and the goal is to infer a particular discrete characteristic of vertices, e.g., whether a user is honest or fraudulent, when given access to limited labeled data. In the second part, we mine anomalies from data where the connectivity evolves over time. Our primary focus here is on real-time detection and early warning so as to enable timely corrective or preventive measures against anomalies.

Table 1.1 provides an overview of this thesis.

Static Graphs (Part II)	**[S1]** Leveraging Heterogeneity (Chapter 2) [EGF+17b] [PDF] **[S2]** Leveraging Confidence (Chapter 3) [EGF17a] [PDF] **[S3]** Leveraging Higher-Order Structures (Chapter 4) [EKF20][PDF]
Dynamic Graphs (Part III)	**[D1]** Anomalous Dense-Subgraph Detection (Chapter 5) [EFGM18] [PDF] **[D2]** Anomalous Edge Detection (Chapter 6) [EF18] [PDF] **[D3]** Early Warning of User-Input Anomalies (Chapter 7) [EFMN19] [PDF]

Table 1.1: Overview of completed, ongoing and proposed work.

1.3 Overview of Part II: Static Graphs

Static graphs, which contain only connectivity information, are a common data representation used in practice when temporal information is not present. In this part, we consider how to conduct transductive learning or semi-supervised learning on static graphs. This is natural way to cast the problem of fraud detection in online platforms like social networks and e-commerce, when a few manually labeled honest and fraudulent accounts are provided, and we want to exploit this information via the graph structure to identify more fraudulent accounts. Our work builds upon existing literature by leveraging three key characteristics of real-world graphs: (a) heterogeneity in vertex and edge types (ZooBP, Chapter 2), (b) skewed degree distribution to incorporate confidence (NetConf, Chapter 3), (c) higher-order network structures (HOLS, Chapter 4), as detailed below.

Leveraging Heterogeneity

"Given a heterogeneous network, with vertices of different types – e.g., products, users and sellers from an online recommendation site like Amazon – and labels for a few vertices ('honest', 'suspicious', etc), can we find a closed formula for Belief Propagation (BP), exact or approximate? Can we say whether it will converge?"

BP, traditionally an inference algorithm for graphical models, exploits so-called "network effects" to perform graph classification tasks when labels for a subset of vertices are provided; and it has been successful in numerous settings like fraudulent entity detection in online retailers and classification in social networks. However, it does not have a closed-form nor does it provide convergence guarantees in general – leading to non-exact solutions on graphs which contain loops. Our goal in this work to derive a fast and accurate approximation of belief propagation for heterogeneous graphs.

Contributions

- *Generality*: ZooBP works on heterogeneous graphs with multiple types of nodes and edges.
- *Closed-Form Solution*: ZooBP gives a closed-form solution as well as convergence guarantees.
- *Scalability*: ZooBP is linear on the graph size and is up to $600\times$ *faster* than BP, running on graphs with 3.3 million edges in a few seconds.
- *Practice*: Applied on real data (a Flipkart e-commerce network with users, products and sellers), ZooBP identifies fraudulent users with a near-perfect precision of 92.3% over the top 300 results.

Leveraging Confidence

"Given a friendship network, how certain are we that a vertex has a particular label? How can we propagate these certainties through the network?"

Traditional semi-supervised methods which propagate labels or beliefs suffer from a major limitation that they do not take uncertainty in the labels or beliefs into account. Consequently, while propagating information, these methods treat vertices with certain and uncertain beliefs with equal weight, resulting in counter-intuitive responses. In this work, we formulate a degree-based notion of confidence (or uncertainty) in beliefs and show how iterative passing of beliefs along with their confidences can lead to better vertex labeling.

Contributions

- *Axioms*: We state axioms that any node classification algorithm should satisfy.
- *Theory*: NetConf is grounded in a Bayesian-theoretic framework to model uncertainties, has a closed-form solution and comes with precise convergence guarantees.
- *Practice*: NetConf is easy to implement and scales linearly with the number of edges in the graph. On experiments using real world data, NetConf always matches or outperforms BP while taking less processing time.

Leveraging Higher-Order Structures

"Do higher-order network structures aid graph semi-supervised learning? How can we leverage them in a principled and efficient manner within an algorithmic framework?"

Traditional graph SSL algorithms tend to be limited by the fact that all the neighbors of a vertex are not equal. A typical user in a friendship network has many acquaintances, but only a few close friends who belong to a small tightly-knit circle. In fact, prior research has shown that vertices with a strong connection participate in several higher-order structures, such as dense subgraphs and cliques. Thus, we hypothesize that leveraging the higher-order structure between vertices is crucial to accurately label the vertices. In this work, we develop an information-theoretic metric to rigorously test the hypothesis on data, and develop Higher-Order Label Spreading (HOLS) algorithm to incorporate the signal present in higher-order structures.

Contributions

- *Metric:* We create an information-theoretic metric to quantify the homogeneity of labels in higher-order structures in graphs. We show that across four diverse real-world networks, higher-order structures exhibit more homogeneity of labels compared to edges.
- *Algorithm:* We create an algorithm, HOLS, for label spreading using higher-order structures. HOLS has strong theoretical guarantees and reduces to standard label spreading in the base case.
- *Practice:* We show that higher-order label spreading using triangles in addition to edges (HOLS-3) is up to 4.7% better than label spreading using edges alone, and outperforms all baselines leading to statistically significantly higher accuracy in all-but-one cases. HOLS-3 is also fast and scalable to large graphs, running under 2 minutes in graphs with over 21 million edges.

1.4 Overview of Part III: Dynamic Graphs

In this part, we focus on near real-time detection and early warning of anomalies, as the connectivity structure evolves over time. In Chapter 5 [EFGM18], we propose a randomized sketching-based approach to detect anomalous dense subgraphs in near real-time by only storing a small synopsis of the graph seen so far. Our work in Chapter 6 [EF18] considers a similar setting, but detects anomalous edges using a novel sampling technique. In Chapter 7 [EFMN19], we show how to learn an interpretable graph representation from time-series data and use it to early warn against user-input anomalies in the presence of interventions. These are detailed below.

Anomalous Dense-Subgraph Detection

"Given a sequence of weighted, directed or bipartite graphs, each summarizing a snapshot of activity in a time window, how can we spot anomalous graphs containing the sudden appearance or disappearance of large dense subgraphs (e.g. near bicliques) in near real-time using sublinear memory?"

This problem has several important applications: detecting attacks such as port scan and denial of service in network communication logs, identifying interesting or fraudulent behavior which create spikes of activity in user-user communication logs (e.g. scammers who operate fast and in bulk), discerning important events such as holidays or large delays which create abnormal traffic in/out flow to certain locations, to name a few. Our goal in these settings is to devise an algorithm which can detect dense subgraph anomalies in near real-time using limited memory and has provable guarantees for detection.

Contributions

- *Theoretical Guarantee:* SpotLight randomized graph sketching guarantees that an anomalous graph is mapped 'far' away from 'normal' instances in the sketch space with high probability for appropriate choice of parameters.
- *Efficiency:* SpotLight supports fast updates and scoring and hence can be efficiently maintained over stream; further, it can detect the sudden appearance or disappearance of anomalous dense subgraphs in sublinear space and constant time per edge.
- *Practice:* Experiments show that SpotLight improves accuracy by at least 8.4% compared to prior approaches, while processing millions of edges within a few minutes.

Anomalous Edge Detection

"Given a stream of edges from a time-evolving (un)weighted (un)directed graph, how can we detect anomalous edges in near real-time using sublinear memory?"

The goal here is to detect whether an edge is anomalous or not *immediately* after it appears, unlike SpotLight which waits for all edges in a single graph snapshot to arrive before flagging anomalies. Naturally, the requirement of per-edge decision limits the anomalies that can be detected, e.g., dense subgraphs can be spotted more easily at the graph level than on a per-edge basis. However, the advantage of such an edge streaming model is that the flagged anomalies can be used right away to curtail the impact of malicious activities and kick-start recovery processes in a timely-manner, e.g., terminating a scam phone call *when it is still ongoing*.

Contributions

- *Burst Resistance:* SedanSpot provably downsamples edges from bursty periods of network traffic, leading to lower sample corruption in the face of lockstep behavior of anomalies.
- *Holistic Scoring:* SedanSpot takes into account the whole (sampled) graph while scoring the anomalousness of an edge, giving diminishing importance to far-away neighbors.
- *Efficiency:* SedanSpot supports fast updates and scoring and hence can be efficiently maintained over stream; further, it can detect anomalous edges in sublinear space and constant time per edge.
- *Practice:* Experiments show that SedanSpot is fast and accurate, and outperforms the state-of-the-art by 270% in terms of AUC while taking $3\times$ less time.

Early Warning of User-Input Anomalies

"How can we early warn against user-input anomalies such an denial of service attack or an adverse health condition in near real-time? More challengingly, how do we learn to early warn from data containing confounding interventions (e.g. medicines) while remaining interpretable to the human decision maker?"

Our SpotLight and SedanSpot algorithms can detect anomalies that have already occurred. While this is a useful primitive to have, ideally, we want to be alerted in advance of upcoming anomalies so that preventive actions–e.g., safeguarding against expected network attacks, pulling over to the side of the road before a seizure–may be taken. Here, we develop SmokeAlarm to infer state-transition graph from time series data in an online manner and use that to early warn against user-labeled anomalies such as adverse medical conditions.

Contributions

- *Principles:* We lay out three characteristics of an ideal early warning system, namely, dominance, precedence and intervention-awareness.
- *Algorithm:* In line with these, we propose SmokeAlarm which learns from past labeled data containing interventions offline and can produce early warnings online.
- *Interpretability:* SmokeAlarm learns state-based progression models in the presence and absence of interventions, which are "bi-inspectable" by the human decision maker.
- *Practice:* Extensive experiments on synthetic and real-world data show that SmokeAlarm outperforms baselines (by $16 - 38\%$ in terms of AUC, with an average lead time of 6.1 hours before the onset of septic shock), while scaling linearly with data size and also leading to intuitive, interesting discoveries in practice.

Part II

Static Graphs

Overview: Static Graphs

Given a large static graph and labels for a few vertices,
how can we infer the most likely labels for all remaining vertices?

Static graphs, which contain only connectivity information, are a common data representation used in practice when temporal information is not present. In this part, we consider how to conduct transductive learning or semi-supervised learning on static graphs. This is a natural way to cast the problem of fraud detection in online platforms like social networks and e-commerce, when a few manually labeled honest and fraudulent accounts are provided, and we want to exploit this information via the graph structure to identify more fraudulent accounts. While our work is motivated primarily by anomaly detection, it is important to note that graph semi-supervised is general problem setting which has applications well beyond those that we consider, for example, in recommendation [YBZ+17] and bioinformatics [VFMV03]. Our work builds upon existing literature by leveraging three key characteristics of real-world graphs: (a) heterogeneity in vertex and edge types (ZooBP, Chapter 2), (b) skewed degree distributions (NetConf, Chapter 3), (c) higher-order network structures (HOLS, Chapter 4). The proposed algorithms have closed-form solutions, rigorous convergence guarantees, can be efficiently implemented using sparse matrix operations, and scale linearly with graph size.

Chapter 2

ZooBP: Leveraging Heterogeneity

Chapter based on work that appeared at VLDB 2017 [EGF+17b] [PDF].

Given a heterogeneous network, with nodes of different types – e.g., products, users and sellers from an online recommendation site like Amazon – and labels for a few nodes ('honest', 'suspicious', etc), can we find a closed formula for Belief Propagation (BP), exact or approximate? Can we say whether it will converge?

BP, traditionally an inference algorithm for graphical models, exploits so-called "network effects" to perform graph classification tasks when labels for a subset of nodes are provided; and it has been successful in numerous settings like fraudulent entity detection in online retailers and classification in social networks. However, it does not have a closed-form nor does it provide convergence guarantees in general. In this chapter, we derive ZooBP, a method to perform fast BP on undirected heterogeneous graphs with provable convergence guarantees. ZooBP has the following advantages: (1) *Generality*: It works on heterogeneous graphs with multiple types of nodes and edges; (2) *Closed-Form Solution*: ZooBP gives a closed-form solution as well as convergence guarantees; (3) *Scalability*: ZooBP is linear on the graph size and is up to $600\times$ *faster* than BP, running on graphs with *3.3 million edges* in a few seconds. (4) *Effectiveness*: Applied on real-world data (a Flipkart e-commerce network with users, products and sellers), ZooBP identifies fraudulent users with a near-perfect precision of *92.3 %* over the top 300 results.

2.1 Introduction

Suppose we are given users, software products, reviews ('likes') and manufacturers; and that we know there are two types of users (honest, dishonest), three types of products (high-quality-safe, low-quality-safe, malware), and two types of sellers (malware, non-malware). Suppose that we also know that user 'Smith' is 'honest', while seller 'evil-dev' sells malware. Given this information, the BP algorithm allows us to infer the types of all other nodes – but will it converge? Can we have a closed formula for the beliefs of all nodes in the above setting?

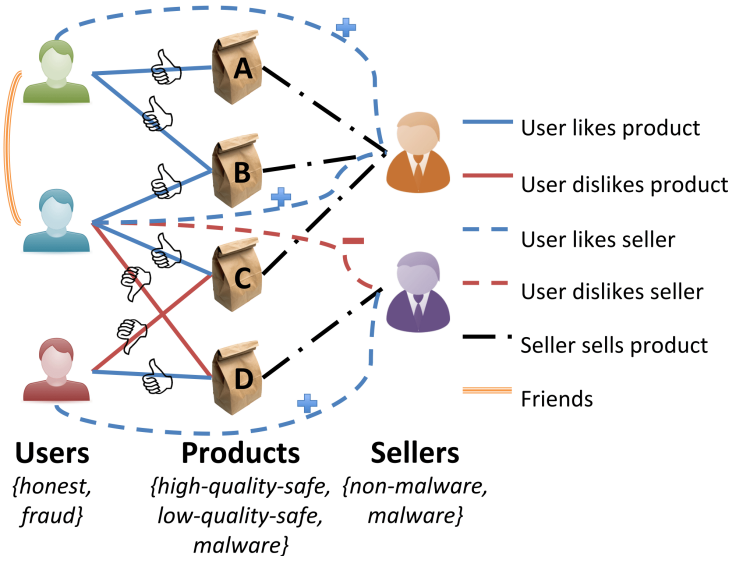


Figure 2.1: ZooBP can handle any undirected, weighted, heterogeneous multi graph

The generic problem for BP is informally given by:

Informal Problem 2.1: General BP

Given

- a large heterogeneous graph (as, e.g., in Figure 2.1),
- for each node type, a set of classes (labels) (e.g. honest/dishonest for user nodes)
- the compatibility matrices for each edge-type, indicating the affinity between the nodes' classes (labels)
- initial beliefs about a node's class (label) for a few nodes in the graph

Find the most probable class (label) for each node.

This problem is found in many other scenarios besides the above mentioned one: in a health-insurance fraud setting, for example, we could have patients (honest or accomplices), doctors (honest or corrupt) and insurance claims (low or expensive or bogus). The textbook solution to this problem is loopy Belief Propagation (BP, in short) – an iterative message-passing algorithm, which, in general, offers no convergence guarantees.

Informal Problem 2.2: BP- Closed-Form Solution

Given a setting like the general BP. **Find** an accurate, closed-form solution for the final beliefs.

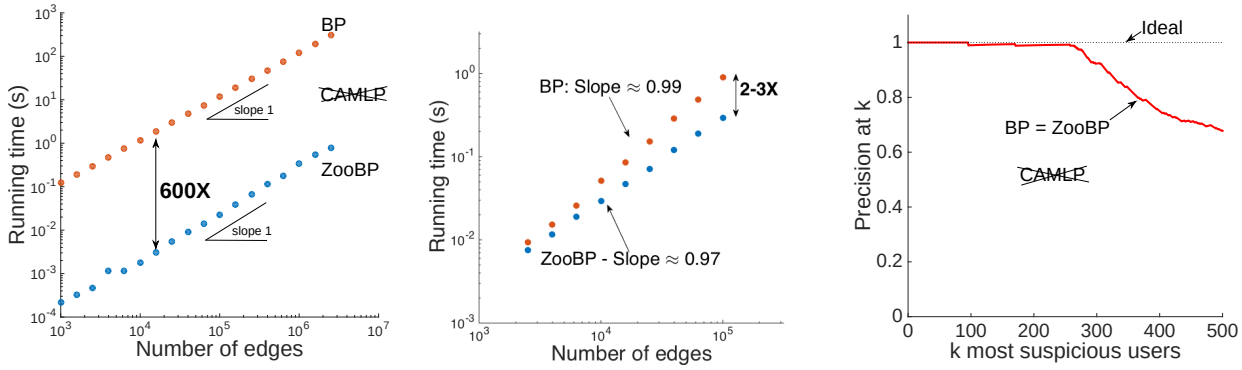


Figure 2.2: ZooBP is (a-b) *fast* up to 600 times depending on platform; (c) *effective* for fraud detection on Flipkart-(3,5) data with 3 node types and 5 edge types. Competitors such as CAMLP are not applicable to this scenario.

Here, we show how to derive a closed-form solution (Theorem 2.1) that almost perfectly approximates the result of loopy BP, using well-understood highly optimized matrix operations and with provable convergence properties. The contributions of our work are:

- **Generality**: ZooBP works on any undirected weighted heterogeneous graph with multiple edge types. Moreover, it trivially includes previous results – FaBP [KKK+11] and LinBP [GGKF15] – as special cases.
- **Closed-Form Solution**: Thanks to our closed-form solution (Theorem 2.1), we know when our method will converge (Theorem 2.2).
- **Scalability**: ZooBP is linear on the input size and it matches or outperforms BP with up to $600\times$ speed-up (Figure 2.2(a)), requiring a few seconds on a stock machine, for million-scale graphs.
- **Effectiveness**: On real-world data (product and seller reviews from Flipkart), ZooBP achieved precision of *92.3 %* in the top 300 most suspicious users (Figure 2.2(c)).

It is worth pointing out that the dramatic $600\times$ savings due to the closed formula: Matlab is very inefficient in handling loops, but there is no other choice with the traditional BP equations (Equation (2.3)). With ZooBP, however, we can replace the loops with a matrix equation (Theorem 2.1) and this allows the use of all the highly optimized matrix algorithms resulting in dramatic speed-ups. Comparisons of C++ implementations (Figure 2.2(b)) show that ZooBP never loses to BP; and it usually wins by a factor of $2\times$ to $3\times$, depending on the relative speed of additions, multiplications, and function calls (logarithms) for the given machine.

2.2 Related Work

In this section, we review related works on belief propagation and summarize prior attempts on linearization.

Propagation in Networks: Exploiting network effects improves accuracy in numerous classification tasks [JNG04, NJ00]. Such methods include random walk with restarts [TFP06], semi-supervised learning [CSZ+06b], label propagation [Zhu05] and belief propagation [Pea82]. Unlike BP, most of the proposed techniques operate on simple unipartite networks only (even though more complex graphs are omnipresent [BGHS12]) or they do not extend to scenarios of heterophily; hence we mainly focus on BP in this work.

Belief Propagation: Belief Propagation [Pea82] is an efficient inference algorithm in graphical models, which works by iteratively propagating network effects. However, there is no closed formula for its solution and it is not guaranteed to converge unless the graph has no loops [Pea14] or on a few other special cases [MK07]. Nevertheless, loopy BP works well in practice [MWJ99] and it has been successfully applied to numerous settings such as error-correcting codes [FK96, FMI], stereo matching in computer vision [SZS03, FH06], fraud detection [PCWF07, ACF13] and interactive graph exploration [CKHF11]. The success of BP has increased the interest to approximate BP and to find closed-form solutions in specialized settings.

Approximation Attempts: Koutra et. al. [KKK+11] provide a linearized approximation of BP for *unipartite graphs with two classes*. and Gatterbauer et. al. [GGKF15] extended it to multiple classes. Gatterbauer [Gat15] attempted to extend this even further to $|T|$*-partite networks*. None of the above can handle a general heterogeneous graph with multiple types of nodes and edges. Even the most general formulation above [Gat15] is limited to single edge type between two node types and hence cannot handle real world scenarios where edges naturally have a polarity (e.g., product-rating networks). In addition, edges between nodes of the same type cannot be handled(e.g., *friendship* edges). Furthermore, [Gat15] neither provides a scalable implementation[1] nor easy-to-compute convergence conditions. Independently, Yamaguchi et al [YFK16] used the degree of a node as a measure the confidence of belief to linearize BP in a completely new way. However, this also assumes a case of unipartite graphs with a single edge type.

Finally, we note that our notion of the term *residual* differs from that of Residual Belief Propagation (RBP) [EMK06]. RBP calculates residuals based on the difference in messages in the successive iterations while our residual beliefs and messages are deviations from their centered values (as we will demonstrate shortly). Our goals are also different: RBP uses residuals to derive an efficient asynchronous BP schedule whereas our interest is in linearizing BP in a heterogeneous setting and providing precise convergence guarantees.

In summary, as shown in Table 2.1, none of competitors satisfy all properties that ZooBP satisfies.

[1]This is non-trivial – naively solving for beliefs directly from Theorem 2.1 leads to an algorithm quadratic in graph size as $(\mathbf{I} - \mathbf{P} + \mathbf{Q})^{-1}$ is a dense matrix.

Table 2.1: Contrasting ZooBP against previous methods

Property	BP [YFW03], RBP [EMK06]	FaBP [KKK+11]	LinBP [GGKF15]	CAMLP [YFK16]	[Gat15]	ZooBP
>2 classes	✓		✓	✓	✓	✓
Node heterogeneity	✓				✓	✓
Unrestricted edge types	✓					✓
Closed-form solution		✓	✓	✓	✓	✓
Convergence guarantees		✓	✓	✓	✓	✓
Scalable implementation	✓	✓	✓	✓		✓

Table 2.2: Notation

Entity/Operator	**Notation**
Scalar	small or capital, italics; e.g., S, k_s
Vector	bold, small; e.g., $\mathbf{b}_u, \mathbf{m}_{uv}$
Matrix	bold, capital; e.g., $\mathbf{B}_s, \mathbf{Q}$
Vectorization	$vec(.)$
Set/Multiset	calligraphic, capital; e.g., $\mathcal{S}$
Kronecker product	$\otimes$
Direct sum of matrices	$\bigoplus$
Vector/matrix entry	Not bold; e.g., $b_u(i), H(i,j)$
Spectral radius	$\rho(.)$

2.3 Preliminaries

We will first provide mathematical definitions and results that we will use in our derivation and then introduce the basic framework of Belief Propagation. We will follow the notation given in Table 2.2.

Definition 2.1: Constant-Margin Matrix

A $p \times q$ matrix is said to be constant-margin of scale α if each row sums to $q\alpha$ and each column sums to $p\alpha$.

Definition 2.2: Matrix Vectorization [HS81]

Vectorization of an $m \times n$ matrix converts it into a $mn \times 1$ vector given by:

$$vec(\mathbf{X}) = [\mathbf{x}_1^T \dots \mathbf{x}_n^T]^T$$

where $\mathbf{x}_i$ denotes the i^{th} column vector of matrix $\mathbf{X}$.

Definition 2.3: Kronecker Product [HS81]

The Kronecker product of two matrices $\mathbf{X}_{m\times n}$ and $\mathbf{Y}_{p\times q}$ is the $mp \times nq$ matrix given by:

$$\mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} X(1,1)\mathbf{Y} & X(1,2)\mathbf{Y} & \dots & X(1,n)\mathbf{Y} \\ X(2,1)\mathbf{Y} & X(2,2)\mathbf{Y} & \dots & X(2,n)\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ X(m,1)\mathbf{Y} & X(m,2)\mathbf{Y} & \dots & X(m,n)\mathbf{Y} \end{bmatrix}$$

Definition 2.4: Centered Matrix/Vector

A matrix or a vector is said to be c-centered if the average of all its elements is c and the maximal deviation from c is small in magnitude when compared to c.

Definition 2.5: Residual Vector/Matrix

A 0-centered vector/matrix is termed as residual. The residual of a c-centered vector or matrix is obtained by subtracting c from each of its elements.

Example 2.1

$\mathbf{X} = \begin{bmatrix} 2.8 & 3 & 3.2 \\ 3.2 & 3 & 2.8 \end{bmatrix}$ is a constant-margin matrix of scale 3. It is also a 3-centered matrix, as the maximal deviation 0.2 is small compared to the overall matrix average 3. Its residual is $\begin{bmatrix} -0.2 & 0 & 0.2 \\ 0.2 & 0 & -0.2 \end{bmatrix}$.

Lemma 2.1: Roth's Column Lemma [HS81]

For any three matrices $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$,

$$vec(\mathbf{XYZ}) \quad = \quad (\mathbf{Z}^T \otimes \mathbf{X})\, vec(\mathbf{Y}) \tag{2.1}$$

Definition 2.6: Matrix Direct Sum [Ayr62]

The matrix direct sum of n square matrices $\mathbf{A}_1, \ldots, \mathbf{A}_n$ is the block diagonal matrix given by $\bigoplus_{i=1}^{n} \mathbf{A}_i = \text{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_n)$.

Belief Propagation: Belief propagation, also known as sum-product message passing, is a technique to perform approximate inference in graphical models. The algorithm starts with prior beliefs for a certain subset of nodes in a graph (e.g., $\mathring{e}_u$; prior knowledge about u's class) and then sequentially propagates from one node (say, u) to another (v) a message ($\mathring{m}_{uv}$) which represents u's belief about v's class. This process is carried out until a steady state is reached (assuming convergence). After the sequential update process, the final beliefs about a node's class (e.g., $\mathring{b}_u$; the inferred information about the class of u) are recovered from the messages that a node receives.

Equation (2.2) and Equation (2.3) give the precise updates of the BP algorithm as given by Yedidia [YFW03].

$$\mathring{b}_u(i) \quad \leftarrow \quad \frac{1}{Z_u} \mathring{e}_u(i) \prod_{v \in \mathcal{N}(u)} \mathring{m}_{vu}(i) \tag{2.2}$$

$$\mathring{m}_{vu}(i) \quad \leftarrow \quad \frac{1}{Z_{vu}} \sum_{j} \phi(i,j)\, \mathring{e}_v(j) \prod_{w \in \mathcal{N}(v) \setminus u} \mathring{m}_{wv}(j) \tag{2.3}$$

In every step of BP, the message that a node sends to another (Equation (2.3)) is computed as the product of the messages it has received from all its neighbors except the recipient itself (echo-cancellation[2]), modulated by the *discrete potential function* $\phi(i,j)$. We let $\phi(i,j)$ be the conditional probability $\mathbb{P}(i|j)$ of class i on node u given class j on node v to facilitate probabilistic interpretation. This value is computed using an *edge compatibility matrix* or *edge-potential matrix* $\mathring{\mathbf{H}}$ as: $\mathbb{P}(i|j) = \mathring{H}(i,j) / \sum_g \mathring{H}(g,j)$. The edge compatibility matrix captures the affinity of classes, i.e., the higher or more positive the value of $\mathring{H}(i,j)$ relative to other entries, the

[2]This term prevents sending the same message that was received in the previous iteration along the same edge. It helps prevent two (or more) nodes mutually reinforcing each others' beliefs.

Table 2.3: Edge Compatibility Matrix for Example 2.2.

↓ People/ Articles →	Conservative	Progressive	Neutral
Republican	0.367	0.300	0.333
Democrat	0.300	0.367	0.333

more probable that a node with class i influences its neighbor to have class j, and vice versa. A numerical example to understand compatibility matrix follows.

Example 2.2

We have a graph on readers and news articles with edges indicating "who reads what". The edge compatibility matrix is given by Table 2.3. Due to the higher value of $\mathring{H}$(Republican, Conservative), a Republican reader is likely to pass the message that the news articles he reads are conservative. Observe that our compatibility matrix is neither square nor doubly stochastic but is constant-margin. This is intentional – we would only be dealing with constant-margin compatibility matrices in our work.

Finally, the normalization constants Z_{vu} and Z_u in Equation (2.2) and Equation (2.3) respectively ensure that the beliefs and messages sum up to a constant (typically 1) at any iteration.

2.4 Proposed Method

The goal of our work is to provide a closed-form solution to BP in arbitrary heterogeneous graphs using an intuitive principle for approximating the beliefs of nodes. The core idea is to derive a system of linear equations for beliefs which can be solved using matrix algebra to finally calculate all node beliefs in a single step of matrix operations. To do this, ZooBP borrows the basic framework of using residual compatibility matrix, beliefs and messages $(\mathbf{H}, \mathbf{b}_u, \mathbf{m}_{vu})$ instead of their non-residual counterparts $(\mathring{\mathbf{H}}, \mathring{\mathbf{b}}_u, \mathring{\mathbf{m}}_{vu})$ as in Equation (2.2) and Equation (2.3) from [KKK+11, GGKF15].

We now describe our problem setting more formally before stating our main (and most generic) results.

2.4.1 Notation and Problem Description

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected heterogeneous graph on a collection of node types $\mathcal{S}$ and edge types $\mathcal{T}$ such that

$$\mathcal{V} = \bigcup_{s \in \mathcal{S}} \mathcal{V}_s \;\; ; \quad \mathcal{E} = \bigcup_{t \in \mathcal{T}} \mathcal{E}_t$$

where $\mathcal{V}_s$ denotes the set of nodes of type s and $\mathcal{E}_t$ the multiset of edges of type t (i.e., parallel/multiple edges are allowed). Each node (edge) has a single node (edge) type. A node's type

determines the set of classes it can belong to. Let us use k_s to denote the number of classes a type-s node can belong to.

Without loss of generality, we assume that an edge type can only connect a particular pair of node types (possibly a self pair, e.g., *friendship* edge). For example, in Figure 2.1, we have separate types of edges, one for "user likes product" and another for "user likes seller", instead of having a single edge type called "like" which connects users with both products and sellers. Observe that this is not a restrictive assumption, as we could always partition a complex edge type ("like") into simpler edge types obeying this condition.

Also, note the subtle generality in our notation – we have used a separate identifier $\mathcal{E}_t$ for edges of type t, instead of referring to it through the pair of node types it connects, $\mathcal{T}_{ss'}$. This allows us to have multiple types of edges connecting the same pair of node types – in Figure 2.1, we have both "user likes product" and "user dislikes product" edges connecting users and products; and possibly we can have multiple types of edges connecting the same pair of nodes as well (for example, a user initially *likes* a product but later *dislikes* it).

Let $\mathring{\mathbf{H}}_t$ and $\mathbf{A}_t$ denote the compatibility matrix and adjacency matrix for an edge type $t \in \mathcal{T}$. If t connects nodes of type s to type s', then $\mathbf{A}_t$ is a $n_s \times n_{s'}$ matrix where each row corresponds to a node of type s and each column corresponds to a node of type s'. Similarly, $\mathring{\mathbf{H}}_t$ is a $k_s \times k_{s'}$ matrix with rows denoting classes of type-s nodes and columns denoting classes of type-s' nodes.

Further, let $\mathcal{T}_{ss'}$ denote the set of edge types connecting node types s and s' and $\mathcal{T}_{uv}$ denote the multiset of edge types (to account for parallel/multiple edges of the same type) connecting nodes u and v. (see Table 2.4).

The problem then is to conduct *transductive inference* [CSZ06a] on this graph, i.e., given initial beliefs for a subset of the nodes, to infer the most probably class for every node in the graph.

2.4.2 Key Insights

2.4.2.1 Residual Compatibility Matrices

In the case of undirected unipartite graphs, the compatibility matrix $\mathring{\mathbf{H}}_t$ turns out to be square and symmetrical; by further assuming that it was doubly stochastic, BP was successfully linearized [GGKF15]. However, in the general case, the compatibility matrices may not be square, let alone doubly stochastic. One core question is: *What kind of compatibility matrices allow a linearization of BP in this general setting?*

We first note that due to the normalization constants in Equation (2.2) and Equation (2.3), the overall scale of the compatibility matrices has no effect on the belief updates. Thus, w.l.o.g., we can fix the scale to be 1 – or using the notion from above, we can focus on 1-centered matrices. Thus, each compatibility matrix can be expressed as

$$\mathring{\mathbf{H}}_t = \mathbf{1} + \epsilon_t \mathbf{H}_t \tag{2.4}$$

where $\mathbf{1}$ is a matrix having the same dimension as $\mathring{\mathbf{H}}_t$ with all of its entries as 1 and $\mathbf{H}_t$ is the *residual compatibility matrix*. Here, ϵ_t can be viewed as the absolute strength of interaction

Table 2.4: Nomenclature

Symbol	Meaning
$\mathcal{S}$	set of node types
S	$\lvert\mathcal{S}\rvert$, the number of node types
s, s'	a type of node; an element in $\mathcal{S}$
$\mathcal{V}_s$	set of nodes belonging to type s
n_s	$\lvert\mathcal{V}_s\rvert$, the number of type-$s$ nodes
k_s	the number of classes (labels) for a type-s node
p_s	$n_s k_s$, the number of personas for all type-s nodes
$\mathbf{E}_s, \mathbf{B}_s$	$n_s \times k_s$ prior and final beliefs (resp.) for all type-s nodes
P	$\sum_{s\in\mathcal{S}} p_s$, the total number of personas for all nodes
$\mathcal{T}$	set of edge types
T	$\lvert\mathcal{T}\rvert$, the number of edge types
$\mathcal{T}_{ss'}$	set of edge types connecting node types$s$ and s'
$\mathcal{T}_{uv}$	multiset of edge types connecting nodes u and v
t, t'	a type of edge; an element in $\mathcal{T}$
$\mathbf{A}_t$	if $t \in \mathcal{T}_{ss'}$, this is the $k_s \times k_{s'}$ adjacency matrix for edge-type t
ϵ_t	interaction strength for edge-type t
$\mathring{\mathbf{H}}_t, \mathbf{H}_t$	compatibility matrix for edge-type t - given and residual (resp.)
$\mathring{\mathbf{e}}_u, \mathbf{e}_u$	prior belief vector of u - given and residual (resp.)
$\mathring{\mathbf{b}}_u, \mathbf{b}_u$	final belief vector of u - given and residual (resp.)
$\mathring{\mathbf{m}}_{vu}, \mathbf{m}_{vu}$	message vector from v to u - given and residual (resp.)
$\mathbf{D}_{st}$	$n_s \times n_s$ diagonal matrix of type-t degrees of type-s nodes
$\mathbf{P}$	$P \times P$ persona-influence matrix
$\mathbf{Q}$	$P \times P$ echo-cancellation matrix
u, v, w	nodes
i, j, g	node classes (labels)

through an edge of type t whereas $\mathbf{H}_t$ indicates the relative affinities of a pair of labels on either side of a type-t edge. Operating with the residual matrix allows us later on to derive an approximation of the BP equations.

The key insight for our result is to focus on compatibility matrices $\mathring{\mathbf{H}}_t$ that are *constant-margin*. Using this property, it follows that the residual $\mathbf{H}_t$ fulfills:

$$\sum_i \mathbf{H}_t(i,j) = \sum_j \mathbf{H}_t(i,j) = 0 \quad \forall \ t, i, j \tag{2.5}$$

Although the constant-margin constraint decreases the number of free parameters for a $p \times q$ compatibility matrix from $pq - 1$ (excluding scale) to $pq - p - q + 1$ (constraining row and column sums to be equal), we note that the set of constant-margin compatibility matrices is sufficiently expressive to model numerous real-world scenarios e.g., fraud detection in e-commerce networks [ACF13], blog/citation networks and social networks [YFK16].

To illustrate this, we derive the residual $\mathbf{H}$ for $\mathring{\mathbf{H}}$ (after re-scaling) given in Table 2.3.

$$\mathring{\mathbf{H}} = \begin{bmatrix} 1.1 & 0.9 & 1 \\ 0.9 & 1.1 & 1 \end{bmatrix} = \mathbf{1} + \overbrace{0.2}^{\epsilon} \overbrace{\begin{bmatrix} 0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \end{bmatrix}}^{\mathbf{H}}$$

In the above example (and in the rest of our work), we fix the scale of the residual $\mathbf{H}$ by holding its largest singular value at 1. This allows us to determine a unique $(\epsilon_t, \mathbf{H}_t)$ pair for a given constant-margin compatibility matrix.

2.4.2.2 Residual Beliefs and Messages

Similar to the original (non-residual) compatibility matrix $\mathring{\mathbf{H}}_t$ and its residual counterpart $\mathbf{H}_t$, we also introduce the notions for residuals of beliefs and messages. Let $\mathring{\mathbf{e}}_u, \mathring{\mathbf{b}}_u, \mathring{\mathbf{m}}_{vu}$ denote the original (non-residual) prior beliefs, final beliefs, and messages; we denote with $\mathbf{e}_u, \mathbf{b}_u, \mathbf{m}_{vu}$ their residual counterparts (see Table 2.4).

Note that the prior and final beliefs of a node sum to 1, i.e. each belief vector b_u is centered around $1/k_s$ where s denotes the type of node u. Similarly, w.l.o.g., messages can be assumed to be centered around 1 (by selecting an appropriate Z_{uv}). Therefore, the residual beliefs and messages are obtained by subtracting their respective *center value* ($1/k_s$ or 1) from their respective initial values. For a node u of type s, these would be:

$$e_u(i) = \mathring{e}_u(i) - \frac{1}{k_s};\ b_u(i) = \mathring{b}_u(i) - \frac{1}{k_s};\ m_{vu}(i) = \mathring{m}_{vu}(i) - 1$$

The normalization constraints on the original values $\mathring{e}_u, \mathring{b}_u$ and $\mathring{m}_{vu}$ thus translate to

$$\sum_i e_u(i) = \sum_i b_u(i) = \sum_i m_{vu}(i) = 0 \ \ \forall \ \ u, v$$

for the residual values.

The overall motivation behind this procedure is to rewrite BP updates in terms of the residuals. Approximating these residuals finally enables us to derive the linearized BP update equations.

2.4.3 ZooBP

Before we proceed to give our main theorem , we introduce some notation that would enable us to solve for the combined beliefs of all nodes via a single equation system. Following this, we provide an example illustrating the definitions.

Definition 2.7: Persona

We use the term "persona" to denote a particular class label of a particular node. For example, if Smith is a node who can be a "democrat" or a "republican", we have the following personas: *Smith-democrat, Smith-republican.* In general, if there are n_s nodes of type s and each can belong to any of the k_s classes, we have $p_s = n_s k_s$ personas in total, for all type-s nodes.

Let us denote the prior residual beliefs of all nodes of type s via the matrix $\mathbf{E}_s$ (see Table 2.4). Here, each row represents the prior residual information about each node, i.e. $\mathbf{e}_u$. If no prior information is given, the row is zero. Similarly, denote with $\mathbf{B}_s$ the final residual beliefs of all nodes of type s after the convergence of belief propagation.

Further, instead of representing the beliefs for each node type individually, we use a joint representation based on the following definition:

Definition 2.8: Vectorized Residual Beliefs $\mathbf{e}, \mathbf{b}$

Based on the type-s residual prior and final belief matrices $\mathbf{E}_s$ and $\mathbf{B}_s$ ($s \in \mathcal{S}$) the vectorized residual prior and final beliefs are constructed as:

$$\mathbf{e} = \begin{bmatrix} vec(\mathbf{E}_1)^T & \dots & vec(\mathbf{E}_S)^T \end{bmatrix}^T \tag{2.6}$$

$$\mathbf{b} = \begin{bmatrix} vec(\mathbf{B}_1)^T & \dots & vec(\mathbf{B}_S)^T \end{bmatrix}^T \tag{2.7}$$

We now ask: How can we describe the net influence that personas of type s exert on personas of type s'? We define a matrix $\mathbf{P}_{ss'}$ which captures exactly this (Equation (2.8)). By concatenating these matrices suitably, we also define the persona-influence matrix $\mathbf{P}$, which consolidates information about how each of the P personas in our graph affects another. Here, we provide equations to derive $\mathbf{P}$ from the graph structure $\mathbf{A}_t$ and the network effects $\mathbf{H}_t, \epsilon_t$ for each $t \in \mathcal{T}$; and later in Section 2.4.4, we will see how this term naturally emerges from our derivation.

Definition 2.9: Persona-Influence Matrix P

From the type-t interaction strength, adjacency and residual compatibility matrices ϵ_t, $\mathbf{A}_t$ and $\mathbf{H}_t$, the persona-influence matrix, which summarizes the net effect a class (label) on a node (i.e., a persona) has on another, is constructed as:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{11} & \dots & \mathbf{P}_{1S} \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{S1} & \dots & \mathbf{P}_{SS} \end{bmatrix} \; ; \; \mathbf{P}_{ss'} = \sum_{t \in \mathcal{T}_{ss'}} \frac{\epsilon_t}{k_s} (\mathbf{H}_t \otimes \mathbf{A}_t) \tag{2.8}$$

where $\mathcal{T}_{ss'}$ is the set of edge types that connect node types s and s'.

Let us use the term *type-t degree* to denote the count of type-t edges incident on a node. If $t \in \mathcal{T}_{ss'}$ and $u \notin \mathcal{V}_s \cup \mathcal{V}_{s'}$, then its type-$t$ degree is defined as zero. Stacking type-t degrees of all s-type nodes diagonally in a $n_s \times n_s$ matrix, we obtain the type-t degree matrix of s-type nodes, $\mathbf{D}_{st}$.

Analogous to the question we asked before we constructed $\mathbf{P}$, we now ask: what is the net influence that a persona exerts on itself through its neighbors? It is important to account for this echo-influence and deduct it from the persona-influence matrix before solving for node beliefs. We calculate this quantity, called echo-cancellation matrix from the degree matrices $\mathbf{D}_{st}$ and the network effects $\mathbf{A}_t$ and ϵ_t for $t \in \mathcal{T}$. Again, we provide the equations here and postpone the derivation until Section 2.4.4.

Definition 2.10: Echo-Cancellation Matrix Q

From the diagonal degree matrices $\mathbf{D}_{st}$ and residual compatibility matrices $\mathbf{H}_t$ and interaction strengths ϵ_t for $t \in \mathcal{T}$, the echo-cancellation matrix may be constructed as:

$$\mathbf{Q} = \bigoplus_{i=1}^{S} \mathbf{Q}_s \quad \text{where,} \quad \mathbf{Q}_s = \sum_{s' \in \mathcal{S}} \sum_{t \in \mathcal{T}_{ss'}} \frac{\epsilon_t^2}{k_s k_{s'}} (\mathbf{H}_t \mathbf{H}_t^T \otimes \mathbf{D}_{st}) \tag{2.9}$$

for the usual meaning of $\mathcal{T}_{ss'}$. Here, $\bigoplus$ denotes the direct sum (Definition 2.6).

Observe that our persona-influence and echo-cancellation matrices are extremely sparse due to the Kronecker product with the adjacency and diagonal degree matrices, respectively; and hence can be efficiently stored in 4GB main memory for even million scale graphs!

The following example illustrates the above definitions.

Example 2.3

Continuing our previous example of a bipartite graph on readers and news articles with "who reads what" edges, let our graph now have 2 readers - R1,R2 and 2 news articles - A, B with adjacency matrix and hence, diagonal degree matrices given by:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \mathbf{D}_R = \mathbf{D}_N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Suppose also, the nodes' residual prior belief matrices are initialized as

$$\mathbf{E}_R = \begin{bmatrix} -0.03 & 0.03 \\ 0 & 0 \end{bmatrix} \mathbf{E}_N = \begin{bmatrix} 0 & 0 & 0 \\ -0.03 & 0.02 & 0.01 \end{bmatrix}$$

stating that R1 is likely to be a Republican and article B is likely to be about democracy. Then, the vectorized residual prior belief vector would be:

$$\mathbf{e} = \begin{bmatrix} -3 & 3 & 0 & 0 & 0 & 0 & 0 & -3 & 2 & 1 \end{bmatrix}^T \times 10^{-2}$$

Thus, using ϵ and $\mathbf{H}$ calculated earlier, the persona-influence and the echo-cancellation matrices can be derived as:

$$\mathbf{P} = \begin{bmatrix} \mathbf{0} & \frac{\epsilon}{2}\mathbf{H} \otimes \mathbf{A} \\ \frac{\epsilon}{3}\mathbf{H}^T \otimes \mathbf{A}^T & \mathbf{0} \end{bmatrix}; \quad \mathbf{Q} = \begin{bmatrix} \frac{\epsilon^2}{6}\mathbf{H}\mathbf{H}^T \otimes \mathbf{D}_R & \mathbf{0} \\ \mathbf{0} & \frac{\epsilon^2}{6}\mathbf{H}^T\mathbf{H} \otimes \mathbf{D}_N \end{bmatrix}$$

Now, we are ready to state our main theorem.

Theorem 2.1: ZooBP

If $\mathbf{b}, \mathbf{e}, \mathbf{P}, \mathbf{Q}$ are constructed as described above, the linear equation system approximating the final node beliefs given by BP is:

$$\mathbf{b} = \mathbf{e} + (\mathbf{P} - \mathbf{Q})\mathbf{b} \quad \text{(ZooBP)} \tag{2.10}$$

Proof. In order to bring $\mathbf{B}_s$ and $\mathbf{B}_{s'}$ out of the matrix product in Equation (2.16), we vectorize Equation (2.16) and then use Roth's column lemma (Equation (2.1)).

$$\begin{aligned} vec(\mathbf{B}_s) \quad = \quad & vec(\mathbf{E}_s) + \sum_{s' \in \mathcal{S}} \sum_{t \in \mathcal{T}_{ss'}} \frac{\epsilon_t}{k_s} (\mathbf{H}_t \otimes \mathbf{A}_t) vec(\mathbf{B}_{s'}) \\ & - \frac{\epsilon_t^2}{k_s k_{s'}} (\mathbf{H}_t \mathbf{H}_t^T \otimes \mathbf{D}_{st}) vec(\mathbf{B}_s) \end{aligned}$$

Rewriting the above equation using $\mathbf{P}_{ss'}$ and $\mathbf{Q}_s$ defined in Equation (2.8) and Equation (2.9) leads to:

$$(\mathbf{I} + \mathbf{Q}_s)vec(\mathbf{B}_s) \quad = \quad vec(\mathbf{E}_s) + \sum_{t \in \mathcal{T}_{ss'}} \mathbf{P}_{ss'} vec(\mathbf{B}_{s'}) \tag{2.11}$$

Equation (2.11) gives the update equation for beliefs of nodes of type s. Here $\mathbf{I}$ is an identity matrix of appropriate dimensions. Stacking S such matrix equations together and rewriting using $\mathbf{e}, \mathbf{b}, \mathbf{P}$ and $\mathbf{Q}$ (defined in Section 2.4.3) gives the equation in Theorem 2.1. ■

Lemma 2.2: ZooBP *

Further, if echo cancellation can be ignored, the linear equation system simplifies to:

$$\mathbf{b} = \mathbf{e} + \mathbf{P}\mathbf{b} \qquad \text{(ZooBP *)} \tag{2.12}$$

Note that our method can also easily handle weighted edges, by appropriately modifying the adjacency matrices to reflect the weights on the edges. Furthermore, ZooBP contains two existing works as special cases:

Lemma 2.3: LinBP and FaBP are Special Cases of ZooBP

For a single node-type connected by a single edge type, our formulation reduces to that of LinBP. In addition, if we constrain the nodes to belong to only two classes, our formulation reduces to that of FaBP.

Proof. Assuming a single node-type and a single edge-type, the persona-influence and echo-cancellation matrices reduce to:

$$\mathbf{P} = \frac{\epsilon}{k}\mathbf{H} \otimes \mathbf{A} \text{ and, } \quad \mathbf{Q} = \frac{\epsilon^2}{k^2}\mathbf{H}^2 \otimes \mathbf{D}$$

Using this together with the relationship between the residual compatibility matrix in both methods ($\hat{\mathbf{H}} = \frac{\epsilon}{k}\mathbf{H}$), our theorem becomes

$$\mathbf{b} = \mathbf{e} + (\mathbf{I} + \hat{\mathbf{H}} \otimes \mathbf{A} + \hat{\mathbf{H}}^2 \otimes \mathbf{D})\mathbf{b}$$

which is exactly LinBP. Thus, LinBP is a special case of ZooBP. As FaBP is a special case of LinBP, it follows that ZooBP subsumes FaBP as well. ■

2.4.4 Derivation of ZooBP

Lemma 2.4: Residual BP

For a pair of nodes $u \in \mathcal{V}_s$ and $v \in \mathcal{V}_{s'}$, BP update assignments can be approximated in terms of residual messages and beliefs as:

$$\begin{aligned} b_u(i) &\leftarrow e_u(i) + \frac{1}{k_s} \sum_{v \in \mathcal{N}_u} \sum_{t \in \mathcal{T}_{uv}} m_{vu}^{(t)}(i) \\ m_{vu}^{(t)}(i) &\leftarrow \frac{\epsilon_t}{k_{s'}} \sum_j H_t(i,j) \Big(k_{s'} b_v(j) - m_{uv}^{(t)}(j) \Big) \end{aligned}$$

where $\mathbf{m}_{vu}^{(t)}$ indicates the message vector that v passes to u through an edge of type t, $\mathcal{N}_u$ is the set of neighbors of u and $\mathcal{T}_{uv}$ is the multiset of edge types corresponding to the edges connecting u and v.

Proof. The proof makes use of the following two approximations for small residuals:

$$\begin{aligned} \ln(1 + b_u(i)) &\approx b_u(i) \\ \frac{\frac{1}{k_{s'}} + b_v(j)}{1 + m_{uv}^{(t)}(j)} &\approx \frac{1}{k_{s'}} + b_v(j) - \frac{m_{uv}^{(t)}(j)}{k_{s'}} \end{aligned}$$

The assumption of "small residuals" is reasonable because the magnitude of residual beliefs has a linear dependence on ϵ and for a given nature of network effects (homophily/heterophily/mixed), decreasing the interaction strength does not affect the accuracy of ZooBP compared to BP as we demonstrate empirically.

We start by rewriting Yedidia's belief update assignment (Equation (2.2)) for a node u belonging to type $s \in \mathcal{S}$, in terms of residual beliefs and messages.

$$\frac{1}{k_s} + b_u(i) \leftarrow \frac{1}{Z_u} \left(\frac{1}{k_s} + e_u(i) \right) \prod_{\substack{v \in \mathcal{N}(u) \\ t \in \mathcal{T}_{uv}}} \left(1 + m_{vu}^{(t)}(i) \right)$$

Now, we take logarithms and assume the residual beliefs are small compared to 1 to use the approximation $\ln(1+x) \approx x$. We obtain:

$$\begin{aligned} b_u(i) &\leftarrow -\frac{1}{k_s} \ln Z_u + e_u(i) + \frac{1}{k_s} \sum_{\substack{v \in \mathcal{N}(u) \\ t \in \mathcal{T}_{uv}}} m_{vu}^{(t)}(i) \qquad (2.13) \\ k_s \underbrace{\sum_i b_u(i)}_{=0} &\leftarrow -\sum_i \ln Z_u + k_s \underbrace{\sum_i e_u(i)}_{=0} + \sum_{\substack{v \in \mathcal{N}(u) \\ t \in \mathcal{T}_{uv}}} \underbrace{\sum_i m_{vu}^{(t)}(i)}_{=0} \end{aligned}$$

In the last step above, we sum both sides over to estimate $Z_u = 1$, which turns out to be constant for all nodes. Substituting this back into Equation (2.13) proves the first part of lemma.

To prove the second part of the lemma, we first write Yedidia's update assignment for the message that a node v of type s' passes to a node u of type s through an edge of type t, i.e., $\mathring{m}_{vu}^{(t)}$:

$$\begin{aligned}
\mathring{m}_{vu}^{(t)}(i) &\leftarrow \frac{Z_v}{Z_{vu}^{(t)}} \sum_j \frac{\mathring{H}_t(i,j)}{\sum_{i'} \mathring{H}_t(i',j)} \frac{\mathring{b}_v(j)}{\mathring{m}_{uv}^{(t)}(j)} \\
1 + m_{vu}^{(t)}(i) &\leftarrow \frac{Z_v}{Z_{vu}^{(t)}} \sum_j \frac{1 + \epsilon_t H_t(i,j)}{\sum_{i'} 1 + \epsilon_t H_t(i',j)} \frac{\frac{1}{k_{s'}} + b_v(j)}{1 + m_{uv}^{(t)}(j)}
\end{aligned}$$

We now use the following approximation (for small residuals):

$$\frac{\frac{1}{k_{s'}} + b_v(j)}{1 + m_{uv}^{(t)}(j)} \approx \frac{1}{k_{s'}} + b_v(j) - \frac{m_{uv}^{(t)}(j)}{k_{s'}}$$

along with $Z_v = 1$ and normalization constraints on residuals simplify the LHS of the above assignment update:

$$\begin{aligned}
&\frac{1}{Z_{vu}^{(t)}} \sum_j \frac{1 + \epsilon_t H_t(i,j)}{k_s} \left(\frac{1}{k_{s'}} + b_v(j) - \frac{m_{uv}^{(t)}(j)}{k_{s'}} \right) \\
&= \frac{1}{Z_{vu}^{(t)} k_s} \Bigg(1 + \underbrace{\sum_j b_v(j)}_{=0} - \frac{1}{k_{s'}} \underbrace{\sum_j m_{uv}^{(t)}(j)}_{=0} + \frac{\epsilon_t}{k_{s'}} \underbrace{\sum_j H_t(i,j)}_{=0} \\
&\qquad + \epsilon_t \sum_j H_t(i,j) b_v(j) - \frac{\epsilon_t}{k_{s'}} \sum_j H_t(i,j) m_{uv}^{(t)}(j) \Bigg)
\end{aligned}$$

Thus, we obtain:

$$1 + m_{vu}^{(t)}(i) \leftarrow \frac{1 + \sum_j \epsilon_t H_t(i,j) \left(b_v(j) - \frac{m_{uv}^{(t)}(j)}{k_{s'}} \right)}{Z_{vu}^{(t)} k_s} \tag{2.14}$$

To calculate $Z_{vu}^{(t)}$, we sum Equation (2.14) over i and use $\sum_i H_t(i,j) = \sum_i m_{vu}^{(t)}(i) = 0$. This leads to $Z_{vu}^{(t)} = \frac{1}{k_s}$. Substituting this in Equation (2.14) proves the second part of the lemma. ■

Lemma 2.5: Steady State Messages

For small residuals and after convergence of belief propagation, message propagation from a node $v \in \mathcal{V}_{s'}$ to node $u \in \mathcal{V}_s$ through an edge of type $t \in \mathcal{T}$ can be expressed in terms of the residual compatibility matrices and steady beliefs approximately as:

$$\mathbf{m}_{vu}^{(t)} = \epsilon_t \mathbf{H}_t \mathbf{b}_v - \frac{\epsilon_t^2}{k_{s'}} \mathbf{H}_t \mathbf{H}_t^T \mathbf{b}_u \tag{2.15}$$

Proof. Rewriting the message update assignment from Lemma 2.4 after expanding the message sent in the opposite direction (i.e., $m_{uv}^{(t)}(j)$) we have:

$$m_{vu}^{(t)}(i) \leftarrow \sum_j \frac{\epsilon_t H_t(i,j)}{k_{s'}} \Bigg(k_{s'} b_v(j) - \sum_g \frac{\epsilon_t H_t(g,j)}{k_s} \Big(k_s b_u(g) - m_{vu}^{(t)}(g) \Big) \Bigg)$$

At convergence, $\mathbf{m}_{vu}^{(t)}$ on both sides need to be identical. So, we replace the update sign with an equality and group similar terms together as follows:

$$m_{vu}^{(t)}(i) - \frac{\epsilon_t^2}{k_s k_{s'}} \sum_j H_t(i,j) \sum_g H_t(g,j) m_{vu}^{(t)}(g) =$$
$$\epsilon_t \sum_j H_t(i,j) b_v(j) - \frac{\epsilon_t^2}{k_{s'}} \sum_j H_t(i,j) \sum_g H_t(g,j) b_u(g)$$

This equation can then be written in matrix-vector notation as:

$$\big(\mathbf{I}_{k_s} - \underbrace{\frac{\epsilon_t^2}{k_s k_{s'}} \mathbf{H}_t \mathbf{H}_t^T}_{\mathbf{X}} \big) \mathbf{m}_{vu}^{(t)} = \epsilon_t \mathbf{H}_t \mathbf{b}_v - \frac{\epsilon_t^2}{k_{s'}} \mathbf{H}_t \mathbf{H}_t^T \mathbf{b}_u$$

The entries of $\mathbf{X} << \frac{1}{k_s}$, and thus inverse of $(\mathbf{I}_{k_s} - \mathbf{X})$ always exists and is, further, approximately $\mathbf{I}_{k_s}$ as $\mathbf{X}$ is composed of second order terms of the (low) interaction strength. This leads to Lemma 2.5. ■

Lemma 2.6: Type-s ZooBP

Using type-s prior and final residual beliefs, $\mathbf{E}_s$ and $\mathbf{B}_s$, type-t adjacency and residual compatibility matrices $\mathbf{A}_t$ and $\mathbf{H}_t$ and diagonal degree matrices $\mathbf{D}_{st}$, the final belief assignment of type-s nodes from belief propagation can be approximated by the equation system:

$$\mathbf{B}_s = \mathbf{E}_s + \sum_{s' \in \mathcal{S}} \sum_{t \in \mathcal{T}_{ss'}} \frac{\epsilon_t}{k_s} \mathbf{A}_t \mathbf{B}_{s'} \mathbf{H}_t^T - \frac{\epsilon_t^2}{k_s k_{s'}} \mathbf{D}_{st} \mathbf{B}_s \mathbf{H}_t \mathbf{H}_t^T$$

Proof. Using Lemma 2.4, the residual belief of a node $u \in \mathcal{V}_s$ can be written in vector notation as:

$$\mathbf{b}_u \leftarrow \mathbf{e}_u + \frac{1}{k_s} \sum_{v \in \mathcal{N}(u)} \sum_{t \in \mathcal{T}_{uv}} \mathbf{m}_{vu}^{(t)}$$

Substituting the steady state value of $\mathbf{m}_{vu}^{(t)}$ from Lemma 2.5, the final belief of u at convergence is:

$$\begin{aligned} \mathbf{b}_u &= \mathbf{e}_u + \frac{1}{k_s} \sum_{v \in \mathcal{N}(u)} \sum_{t \in \mathcal{T}_{uv}} \left(\epsilon_t \mathbf{H}_t \mathbf{b}_v - \frac{\epsilon_t^2}{k_{s'}} \mathbf{H}_t \mathbf{H}_t^T \mathbf{b}_u \right) \\ &= \mathbf{e}_u + \sum_{v \in \mathcal{N}(u)} \sum_{t \in \mathcal{T}_{uv}} \frac{\epsilon_t}{k_s} \mathbf{H}_t \mathbf{b}_v - \frac{\epsilon_t^2 d_u^{(t)}}{k_s k_{s'}} \mathbf{H}_t \mathbf{H}_t^T \mathbf{b}_u \end{aligned}$$

where $d_u^{(t)}$ is the type-t degree of u, i.e., the number of type-t edges incident on u. Rewriting the above equation in matrix form using type-t adjacency matrices $\mathbf{A}_t$ for $t \in \mathcal{T}$, prior and final residual type-s belief matrices $\mathbf{B}_s$ for $s \in \mathcal{S}$ and diagonal degree matrices $\mathbf{D}_{st}$ summarizing type-t degree of type-s nodes, yields Lemma 2.6:

$$\mathbf{B}_s = \mathbf{E}_s + \sum_{s' \in \mathcal{S}} \sum_{t \in \mathcal{T}_{ss'}} \frac{\epsilon_t}{k_s} \mathbf{A}_t \mathbf{B}_{s'} \mathbf{H}_t^T - \frac{\epsilon_t^2}{k_s k_{s'}} \mathbf{D}_{st} \mathbf{B}_s \mathbf{H}_t \mathbf{H}_t^T$$

■

2.4.5 Iterative Updates and Convergence

Using Theorem 2.1, the closed form solution for node beliefs is:

$$\mathbf{b} = (\mathbf{I} + \mathbf{Q} - \mathbf{P})^{-1} \mathbf{e} \tag{2.16}$$

However, in practice, computation of the inverse of a large matrix such as $(\mathbf{I} + \mathbf{Q} - \mathbf{P})$ is very expensive and is done iteratively. Hence, we propose to do iterative updates of the form:

$$\mathbf{b} \leftarrow \mathbf{e} + (\mathbf{P} - \mathbf{Q})\mathbf{b} \tag{2.17}$$

Theorem 2.2 gives precise theoretical guarantees for the convergence of these iterative updates.

Theorem 2.2: Exact Guarantees for Convergence of ZooBP

The necessary and sufficient condition for convergence of iterative updates in Equation (2.17) in terms of the persona-influence matrix $\mathbf{P}$ and echo-cancellation matrix $\mathbf{Q}$ is:

$$\text{ZooBP converges} \iff \rho(\mathbf{P} - \mathbf{Q}) < 1 \tag{2.18}$$

Proof. From the Jacobi method for solving linear equations [Saa03], we know that the update in Equation (2.17) converges for any arbitrary initialization of $\mathbf{b}$ if and only if the spectral radius of $\mathbf{P} - \mathbf{Q}$ is strictly less than 1. ■

The implicit convergence criterion poses difficulties in choosing appropriate ϵ_t to a practitioner. Thus, for practitioners' benefit, we tie all interaction strengths as $\epsilon_t = \epsilon \ \forall t$, use the fact that the spectral norm of a matrix is bounded above by any matrix norm $||\cdot||$ to provide an easier-to-use sufficient condition for convergence. This is stated in Theorem 2.3.

Theorem 2.3: Sufficient Guarantees for Convergence of ZooBP

Let $\mathbf{P}'$ and $\mathbf{Q}'$ be the persona-influence and echo-cancellation matrices obtained from Equation (2.8) and Equation (2.9) by temporarily setting $\epsilon_t = 1 \ \forall t$. If the overall interaction strength ϵ is chosen such that,

$$\epsilon \leq \frac{-||\mathbf{P}'|| + \sqrt{||\mathbf{P}'||^2 + 4\,||\mathbf{Q}'||^2}}{2\,||\mathbf{Q}'||}$$

then, ZooBP is guaranteed to converge. Here, $||\mathbf{P}'||$ and $||\mathbf{Q}'||$ can be chosen as any (possibly different) matrix norms.

Proof. By tying interaction strengths across all edges, the exact convergence criterion can be restated in terms of $\mathbf{P}'$ and $\mathbf{Q}'$ as:

$$\rho(\epsilon\mathbf{P}' - \epsilon^2\mathbf{Q}') < 1$$

Using triangle inequality, $\rho(\epsilon\mathbf{P}' - \epsilon^2\mathbf{Q}') \leq \epsilon\rho(\mathbf{P}') + \epsilon^2\rho(\mathbf{Q}')$. Also, as any matrix norm is larger than the spectral norm, this quantity is further bounded above by $\epsilon\,||\mathbf{P}'|| + \epsilon^2\,||\mathbf{Q}'||$. Thus, to ensure convergence, it is sufficient to solve for ϵ using:

$$\epsilon\,||\mathbf{P}'|| + \epsilon^2\,||\mathbf{Q}'|| - 1 < 0$$

This completes the proof. We are free to choose the matrix norms (possibly different norms for $\mathbf{P}'$ and $\mathbf{Q}'$) that would give the tightest bound on the spectral radii of these matrices. ■

2.4.6 Time and Space Complexity

Lemma 2.7: Time and Space Complexity

The space and per-iteration time complexity of ZooBP is linear in the total number of nodes and edges, i.e., $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ and is given by

$$\mathcal{O}\left(\sum_{s\in\mathcal{S}} k_s n_s + \sum_{s'\in\mathcal{S}}\sum_{t\in\mathcal{T}_{ss'}} k_s(k_s + k_{s'})\text{NNZ}(\mathbf{A}_t)\right) \tag{2.19}$$

where $n_s = |\mathcal{V}_s|$ is the number of s-type nodes, k_s is the corresponding number of classes and $\text{NNZ}(\mathbf{A}_t) = |\mathcal{E}_t|$ is the number of non-zero elements in $\mathbf{A}_t$ (i.e., edges of type-t).

Proof. We begin by computing an upper limit on the number of non-zeros of $\mathbf{P}$ and $\mathbf{Q}$:

$$\begin{aligned}
\text{NNZ}(\mathbf{P}) &= \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}} \text{NNZ}(\mathbf{P}_{ss'}) \\
&\leq \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{t\in\mathcal{T}_{ss'}} k_s k_{s'}\text{NNZ}(\mathbf{A}_t) \\
\text{NNZ}(\mathbf{Q}) &= \sum_{s\in\mathcal{S}} \text{NNZ}(\mathbf{Q}_s) \\
&\leq \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{t\in\mathcal{T}_{ss'}} k_s^2\text{NNZ}(\mathbf{A}_t)
\end{aligned}$$

where we have used $\text{NNZ}(\mathbf{X}\otimes\mathbf{Y}) = \text{NNZ}(\mathbf{X})\cdot\text{NNZ}(\mathbf{Y})$. Using the above, we can bound the non-zeros of $\mathbf{P}-\mathbf{Q}$ as:

$$\text{NNZ}(\mathbf{P}-\mathbf{Q}) \leq \sum_{s\in\mathcal{S}}\sum_{s'\in\mathcal{S}}\sum_{t\in\mathcal{T}_{ss'}} k_s(k_s + k_{s'})\text{NNZ}(\mathbf{A}_t) \tag{2.20}$$

Space Complexity can be computed as the space required to store the sparse matrix $\mathbf{P}-\mathbf{Q}$ and the dense $\sum_{s\in\mathcal{S}} k_s n_s$-dimensional prior and belief vectors. **Per-iteration Time Complexity** (Equation (2.19)) is estimated from the number of unit operations (addition/multiplication) involved in computing the LHS of the iterative update (Equation (2.17)). The breakdown for each operation is given below.

Computation	Unit ops.
Subtraction of $\mathbf{Q}$ from $\mathbf{P}$	$\mathcal{O}\left(\text{NNZ}(\mathbf{P}-\mathbf{Q})\right)$
Multiplication of $(\mathbf{P}-\mathbf{Q})$ and $\mathbf{b}$	$\mathcal{O}\left(\text{NNZ}(\mathbf{P}-\mathbf{Q})\right)$
Addition of $\mathbf{e}$ and $(\mathbf{P}-\mathbf{Q})\mathbf{b}$	$\mathcal{O}\left(\sum_{s\in\mathcal{S}} k_s n_s\right)$

■

Table 2.5: Sample $\mathbf{H}_+, \mathbf{H}_-$ for product rating networks

$\mathbf{H}_+$	*Good*	*Bad*
Honest	0.5	-0.5
Fraud	-0.5	0.5

$\mathbf{H}_-$	*Good*	*Bad*
Honest	-0.5	0.5
Fraud	0.5	-0.5

2.4.7 Case Study - Product-Rating Network

In the following section, we introduce a case study of our ZooBP for product-rating networks (which are signed bipartite networks) – other complex scenarios can also be represented easily. The goal is to classify users and products as fraudulent or not.

Let $\mathcal{G} = (\mathcal{V}_u \cup \mathcal{V}_p, \mathcal{E}_+ \cup \mathcal{E}_-)$ be a product rating network where $\mathcal{V}_u$ is the set of users and $\mathcal{V}_p$ is the set of products. Let $n_p = |\mathcal{V}_p|$ be the number of products and $n_u = |\mathcal{V}_u|$ the number of users. The edge sets $\mathcal{E}_+$ and $\mathcal{E}_-$ represent the positive and negative ratings respectively.

Given the edge sets, we denote the corresponding $n_u \times n_p$ adjacency matrices as $\mathbf{A}_+$ and $\mathbf{A}_-$. Here, the rows correspond to users and columns correspond to products. Furthermore, let us use the term *positive degree* to denote the number of positive ratings given to a product or by a user (depending on the node type). Let $\mathbf{D}_{u+}, \mathbf{D}_{p+}$ be the $n_u \times n_u$ and $n_p \times n_p$ diagonal matrices of positive degree for users and products respectively. Similarly, we define diagonal degree matrices of *negative degree* for users and products - $\mathbf{D}_{u-}, \mathbf{D}_{p-}$.

Further, let the residual compatibility matrices for positive and negative edges be $\mathbf{H}_+$ and $\mathbf{H}_-$ and the corresponding edge interaction strengths be ϵ_+ and ϵ_-. Here, the rows correspond to user-classes and columns correspond to product-classes. In general, one would expect honest users to give positive ratings to good products, while positive ratings for fraudulent products are less likely; fraudsters in contrast might give positive ratings to fraudulent products. Thus, the matrices $\mathbf{H}_+$ and $\mathbf{H}_-$ might be instantiated as in Table 2.5 – of course, in our model, any other constant-margin instantiation can be picked as well.

In general, considering the setting of product rating networks, the persona-influence and echo-cancellation matrices are given by:

$$\mathbf{P} = \begin{bmatrix} \mathbf{0} & \frac{\epsilon_+}{2}\mathbf{H}_+ \otimes \mathbf{A}_+ + \frac{\epsilon_-}{2}\mathbf{H}_- \otimes \mathbf{A}_- \\ (\frac{\epsilon_+}{2}\mathbf{H}_+ \otimes \mathbf{A}_+ + \frac{\epsilon_-}{2}\mathbf{H}_- \otimes \mathbf{A}_-)^T & \mathbf{0} \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} \frac{\epsilon_+^2}{4}\mathbf{H}_+\mathbf{H}_+^T \otimes \mathbf{D}_{u+} + \frac{\epsilon_-^2}{4}\mathbf{H}_-\mathbf{H}_-^T \otimes \mathbf{D}_{u-} & \mathbf{0} \\ \mathbf{0} & \frac{\epsilon_+^2}{4}\mathbf{H}_+^T\mathbf{H}_+ \otimes \mathbf{D}_{p+} + \frac{\epsilon_-^2}{4}\mathbf{H}_-^T\mathbf{H}_- \otimes \mathbf{D}_{p-} \end{bmatrix}$$

These matrices can now be used to compute the final beliefs using Theorem 2.1 (ZooBP).

Besides this general solution, let us focus on the case where we set compatibility matrices as in Table 2.5 and tie the interaction strengths across both types of edges $\epsilon_+ = \epsilon_- =: \epsilon$.

Let us now define the total adjacency matrix ($\mathbf{A}$) and total diagonal degree matrix ($\mathbf{D}$) as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_+ - \mathbf{A}_- \\ (\mathbf{A}_+ - \mathbf{A}_-)^T & \mathbf{0} \end{bmatrix}; \quad \mathbf{D} = \begin{bmatrix} \mathbf{D}_{u+} + \mathbf{D}_{u-} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{p+} + \mathbf{D}_{p-} \end{bmatrix}$$

Using these, Theorem 2.4 provides a compact closed-form solution for ZooBP (proof omitted for brevity).

Theorem 2.4: ZooBP-2F

For fraud detection in product-rating networks, if ϵ denotes the desired interaction strength and $\mathbf{A}$ and $\mathbf{D}$ are the total adjacency and diagonal degree matrices of users and products as defined above, the ZooBP-2F closed-form solution is:

$$\mathbf{b} = \mathbf{e} + \left(\begin{bmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} \otimes \left(\frac{\epsilon}{2}\mathbf{A} - \frac{\epsilon^2}{4}\mathbf{D} \right) \right) \mathbf{b}$$

The exact and sufficient conditions for convergence of our ZooBP-2F can be derived as before:

Lemma 2.8: Exact Guarantees for Convergence of ZooBP-2F

For fraud detection in product rating network, the *sufficient and necessary* condition for convergence, in terms of total adjacency matrix $\mathbf{A}$, the total diagonal degree matrix $\mathbf{D}$ and interaction strength ϵ is given by: $\rho\left(\frac{\epsilon}{2}\mathbf{A} - \frac{\epsilon^2}{4}\mathbf{D}\right) < 1$

Lemma 2.9: Sufficient Guarantees for Convergence of ZooBP-2F

For fraud detection in product rating network, with $\mathbf{A}$ and $\mathbf{D}$ denoting the total adjacency and diagonal degree matrices respectively, if the interaction strength (ϵ) is chosen to satisfy

$$\epsilon < \frac{-||A|| + \sqrt{||A||^2 + 4d_{\max}}}{d_{\max}}$$

then, ZooBP $-$ $2F$ is guaranteed to converge.

2.5 Experiments

We conduct experiments to answer the following questions:

- **Q1. Accuracy:** How well can ZooBP reconstruct the final beliefs given by BP? How accurate are its predictions on real-world data?
- **Q2. (In-)Sensitivity to Interaction Strength:** How does the performance of ZooBP vary with interaction strength ϵ? What happens at the critical ϵ_* from Theorem 2.2? How sensitive is ZooBP to ϵ when $\epsilon < \epsilon_*$?

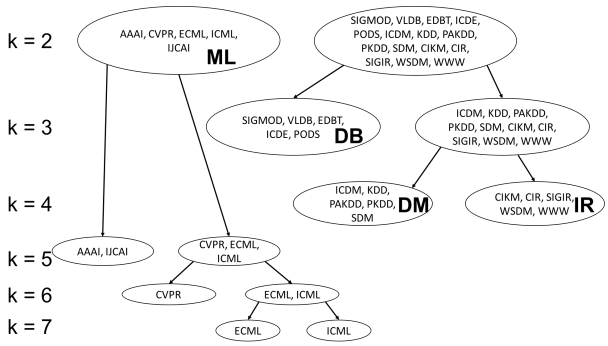
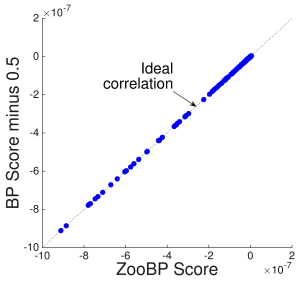
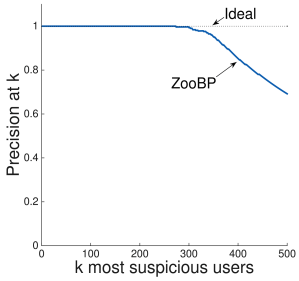


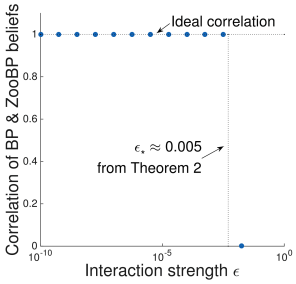
Figure 2.3: DBLP 4-area (Databases, Data Mining, Machine Learning, Information Retrieval, resp.) dataset with class hierarchy ($k = [2, \ldots, 7]$)



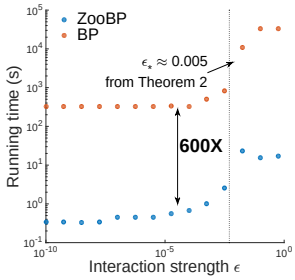
(a) ZooBP matches the beliefs of BP



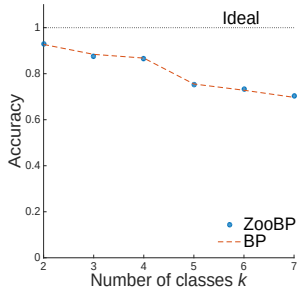
(b) ZooBP is effective in fraud detection



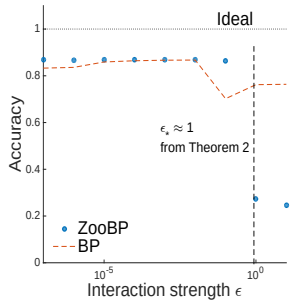
(c) ZooBP's approx. quality is independent of ϵ (when $\epsilon < \epsilon_*$)



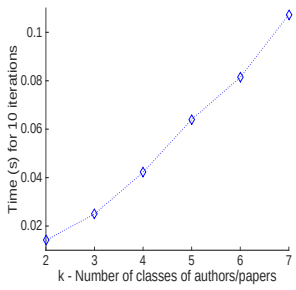
(d) ZooBP's speed gain is fairly robust to ϵ (for $\epsilon < \epsilon_*$)



(e) ZooBP is accurate on DBLP



(f) ZooBP's accuracy on DBLP ($k = 4$) is robust to ϵ (for $\epsilon < \epsilon_*$)



(g) ZooBP scales quadratically with #classes on DBLP

k	BP (s)	ZooBP (s)
2	24.0196	0.0141
3	28.8042	0.0250
4	33.4384	0.0422
5	37.6492	0.0640
6	42.1226	0.0814
7	46.5648	0.1073

(h) BP vs ZooBP: MATLAB running time for 10 iterations on DBLP with k

Figure 2.4: Experimental results on FLIPKART (a-d) and DBLP (e-h) data: ZooBP is accurate, robust, fast and scalable

Q3. Speed & Scalability: How well does ZooBP scale with the network size? How fast is ZooBP compared to BP? Why? Does the speed-up generalize to networks with arbitrary number of node-/edge-types and classes?

We now describe the data we use for our experiments.

2.5.1 Data Description and Experimental Setup

We use the following two real-world heterogeneous datasets in our experiments.

2.5.1.1 DBLP

The DBLP 4-area dataset consists of authors and the papers published by them to 12 conferences. In the original dataset, these conferences were split into four areas (DB, DM, ML, IR). To perform a deeper analysis, we varied the number of classes, k, from 2 to 7 by merging or partitioning the above areas based on the conferences (Figure 2.3). The network is bipartite (node types $\mathcal{S}$ = {author, paper}) with a single type of edges ($\mathcal{T}$ = {authorship}). The goal is to assign a class to each author and paper.

The ground truth areas for papers and authors were obtained as follows. The area of a paper is the area of the conference it is published in. The area of an author is the area which most of her papers belong to, with ties broken randomly. As homophily captures the nature of network effects in this dataset, a $k \times k$ compatibility matrix with interaction strength ϵ and residual compatibility matrix $\mathbf{H}_{k\times k} = \mathbf{I}_{k\times k} - \frac{1}{k}\mathbf{1}_{k\times k}$ was used.

In our experiments, we seeded randomly chosen 30% of the authors and papers to their ground truth areas. The prior for the correct class was set to $+k \times 0.001$ and for the wrong classes was set to -0.001. $[0, \ldots, 0]$ was used as the prior for unseeded nodes.

2.5.1.2 Flipkart

Flipkart is an e-commerce website that provides a platform for sellers to market their products to customers. The dataset consists of about 1M users, their ∼3.3M ratings to ∼512K products and ∼1.7M ratings to ∼4K sellers. In addition, we also have the connections between sellers and products. All ratings are on a scale of 1 to 5 – for simplicity, we treated 4 and 5 star ratings as positive edges, 1 and 2 star as negative edges and ignored the 3 star ratings.

We consider two versions of the data: (1) Flipkart-(2,2) (or Flipkart in short) containing only user-product rating information (node types $\mathcal{S}$ = {user, product} and edge types $\mathcal{T}$ = {positive rating for product, negative rating for product}); and (2) Flipkart-(3,5) containing all 3 node types (user, product, seller) and 5 edge types (positive rating for product, negative rating for product, positive rating for seller, negative rating for seller, seller sells product).

In both the Flipkart datasets, our goal is to classify users and products (and sellers) as fraudulent or not. $\mathbf{H}_+$ and $\mathbf{H}_-$ for both user-product and user-seller edges were chosen as in Table 2.5 and ϵ values were tied and set to 10^{-4}, unless mentioned otherwise. We used 50 manually labeled fraudsters as seed labels and initialized their prior to $[-0.05, +0.05]$ respectively for the honest and fraudulent classes. The prior for other users and all products (and all sellers) were set to $[0, 0]$.

We provide the full analysis on the DBLP and Flipkart datasets; for brevity, we only present the fraud detection precision results on Flipkart-(3,5). To compare running times on DBLP and Flipkart data with BP, we used an off-the-shelf Matlab implementation of BP for

signed bipartite networks [ACF13]. To enable a fair comparison, we implemented ZooBP also in MATLAB.

2.5.2 Q1. Accuracy

A plot of the final beliefs returned by BP and ZooBP on FLIPKART for $\epsilon = 10^{-4}$ is shown in Figure 2.4(a). Here, we have subtracted 0.5 from the BP score (see y-axis) to match the scale of beliefs from both methods. We see that all points lie on the line of slope 1 passing through the origin, showing that ZooBP beliefs are highly correlated with BP beliefs. Such a trend was observed for all the ϵ values we tried (while ensuring that $\epsilon < \epsilon_*$, the limit given by Theorem 2.2).

Upon applying ZooBP to our data, we provided the list of 500 most fraudulent users (after sorting beliefs) to the domain experts at FLIPKART, who verified our labels by studying various aspects of user behavior such as frequency and distribution of ratings and review text given by them. Figure 2.4(b) and Figure 2.2(c) depict how the precision at k changes with k over the top 500 results on FLIPKART and FLIPKART-(3,5) datasets. The high precision ($\tilde{1}00\%$ for top 250; $\tilde{7}0\%$ for top 500 users) confirms the effectiveness of ZooBP. Owing to difficulty in obtaining ground truth for all 1M users, studying recall was not possible.

Using the DBLP data, we study the performance for a graph from a different domain (citation network), with more than two classes. Figure 2.4(e) plots accuracy vs number of classes, k. The uniformly high accuracy across k suggest our performance can be expected to generalize well to networks from different domains with arbitrary classes.

In sum, our accuracy results show that (1) our assumption of constant-margin compatibility matrix is applicable in several realistic scenarios (2) our linear approximations do not lower the quality of prediction, thus making ZooBP extremely useful in practice, for solving several real world problems.

2.5.3 Q2. (In-)Sensitivity to Interaction Strength

Next, we study how the compatibility matrix (through interaction strength ϵ) influences the performance and speed of ZooBP. Figure 2.4(c) and Figure 2.4(d) summarize the results on FLIPKART.

The correlation (of BP and ZooBP beliefs) and the running time were found to be fairly constant with ϵ as long as $\epsilon < \epsilon_*$ (the limit from Theorem 2.2). As the spectral radius of $\mathbf{P} - \mathbf{Q}$ approaches 1, a slight increase in running time near ϵ_* is observed; but the correlation is still high. When $\epsilon > \epsilon_*$, the algorithm does not converge – ZooBP runs to a manually set maximum iteration count of 200. Hence, the running time suddenly increases past the dotted line, while the correlation drops to 0.0001. The resulting beliefs for high interaction strength ($\epsilon > 0.1$) were found to be unbounded (reaching $\pm\infty$) for some nodes – making the correlation coefficient indeterminate (these are omitted in Figure 2.4(c)).

On the DBLP data, we are not restricted to study correlation but are able to analyze the actual accuracy of BP and ZooBP with varying ϵ. Figure 2.4(f) depicts the classification accuracy vs ϵ for the DBLP dataset with $k = 4$. We see that both BP and ZooBP achieve a robust high accuracy on a range of ϵ values within the convergence limit. Not surprisingly, when ϵ was increased

beyond $\epsilon_* = 1$, the performance of both methods deteriorated. This suggests our algorithm is *practically useful*, with robust approximation quality in BP's optimal range of interaction strength.

These results show that our method is fairly robust (except around and beyond ϵ_*) and not sensitive to the selection of interaction strength in general. Moreover, this value of ϵ_* is exactly as predicted by Theorem 2.2 (specifically, Result 2.8), thus validating its correctness.

Note to practitioner: Owing to the finite precision of machines, we recommend setting $\epsilon \in [0.01\epsilon^\dagger, 0.1\epsilon^\dagger]$, where $\epsilon^\dagger$ is calculated from Theorem 2.3.

2.5.4 Q3. Speed & Scalability

To examine the scalability of our method, we uniformly sampled 1K-3.3M edges from the FLIPKART data and timed BP and ZooBP (with $\epsilon = 10^{-4}$) on the resulting subgraphs. We focus on the time taken for computations alone and ignore the time to load data and initialize matrices. The results are shown in Figure 2.2(a) (MATLAB) and Figure 2.2(b) (C++).

We see that ZooBP scales linearly with the number of edges in the graph (i.e., graph size), which is same as the scalability of BP. In addition, on MATLAB, ZooBP also offers a $600\times$ *speed-up* compared to BP, which is one of its most important practical advantages. On FLIPKART dataset with 3.3M edges, ZooBP requires only 1 second to run!

What can this speed-up be attributed to? There are two primary contributing factors: **(F1)** ZooBP replaces the expensive logarithms and exponentiation operations in BP by multiplication and addition; **(F2)** ZooBP (via Theorem 2.1) converts the iterative BP algorithm into a matrix problem – it foregoes the redundant message computation and exploits optimized sparse-matrix operations.

To investigate the relative importance of the above factors, we implemented Lemma 2.4 in MATLAB. Lemma 2.4 is similar to BP except in operating on residuals directly in the linear space through lighter-weight operations and hence serves as a clean break point between BP and ZooBP to compare the speed-ups due to F1 and F2 individually. Our experimental observations are summarized below:

- **Savings A ($\sim 2\times$)** BP $\rightarrow$ Lemma 2.4 (lighter operations)
  This speed-up is not tied to MATLAB as we demonstrate through identical experiments in C++ (Figure 2.2(b)). Savings A is platform-independent with the precise speed-up factor depending on the architecture-specific relative speed of elementary floating point operations (add, multiply) and function calls (exp, log).
- **Savings B ($\sim 300\times$):** Lemma 2.4 $\rightarrow$ ZooBP (optimized sparse matrix operations of MATLAB).
  We note that although the $300\times$ savings from MATLAB implementation is largely due to its inefficient handling of loops, it may prove to be a critical factor of consideration for a number of data mining practitioners.

Can we explain the speed-up in terms of the architecture specifications? Our experiments used Intel i5 (Haswell) processor[3]. In this architecture, multiplication instructions (FMUL, FIMUL) issued up to two times more macro instructions (OPs) than addition or sub-

[3] http://www.agner.org/optimize/instruction_tables.pdf pages 189-191

traction (FADD, FSUB, FIADD, FISUB). Further, function calls (i.e., control transfer instructions such as CALL) needed 2-3 times more clock cycles compared to arithmetic operations. This is exactly the speed-up (2-3×) that ZooBP achieves over BP in C++, as shown in Figure 2.2(b).

Do the speed gains persist as the number of classes grows? The answer is 'yes'. Figure 2.4(g) and Figure 2.4(h) show the results on the DBLP data. ZooBP scales quadratically with number of class labels, as expected from Lemma 2.7; but the speed-up gains were consistently $\approx 600\times$ even as k varied (Figure 2.4(h)).

Comparison with the state-of-the-art: Table 2.1 gives the qualitative comparison of ZooBP with top competitors. Only BP (and its asynchronous equivalent, RBP) can solve the general problem, but neither of them provides a closed-form solution or convergence guarantees. Still, we have provided comparison results against BP. None of the other methods (LinBP [GGKF15], FaBP [KKK+11], [Gat15]) can handle arbitrary heterogeneous graphs (e.g., Flipkart-(3,5)) and are dropped from comparison. We use CAMLP as a baseline on the DBLP data, although it cannot handle multiple node-types. In our experiments on DBLP data with $k = 4$, ZooBP practically tied CAMLP (86% vs. 87% accuracy).

In summary, our experiments show that ZooBP obtains a very high prediction accuracy on real-world data, while at the same time, being highly scalable at handling million-scale graphs.

2.6 Conclusion

We presented ZooBP, a novel framework which approximates BP with constant-margin compatibility matrices, in any undirected weighted heterogeneous graph. Our method has the following advantages:

- **Generality**: ZooBP approximates BP in *any* kind of undirected weighted heterogeneous graph, with arbitrarily many node and edge types. Moreover, it includes existing techniques like FaBP and LinBP as special cases.
- **Closed-Form Solution**: ZooBP leads to a closed form solution (Theorem 2.1, Equation (2.10)), which results in exact convergence guarantees (Theorem 2.2).
- **Scalability**: ZooBP scales linearly with the number of edges in the graph; moreover, it nevers loses, and it usually wins over traditional BP, with *up to* $600\times$ *speed-up* for Matlab implementation.
- **Effectiveness**: Applied on real-world data (Flipkart), ZooBP matches the accuracy of BP, achieving 92.3 % precision for the top 300 nodes.

Chapter 3

NetConf: Leveraging Confidence

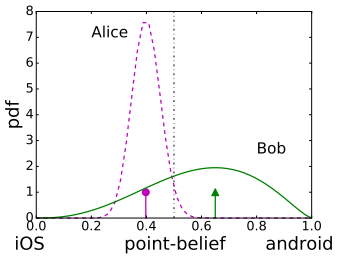
Chapter based on work that appeared at SDM 2017 [EGF17a] [PDF].

Given a friendship network, how certain are we that Smith is a progressive (vs. conservative)? How can we propagate these certainties through the network? While Belief propagation marked the beginning of principled label-propagation to classify nodes in a graph, its numerous variants proposed in the literature fail to take into account uncertainty during the propagation process. As we show, this limitation leads to counter-intuitive results for even simple graphs. Motivated by these observations, we formalize axioms that any node classification algorithm should obey and propose NetConf which satisfies these axioms and handles arbitrary network effects (homophily/heterophily) at scale. Our contributions are: (1) *Axioms*: We state axioms that any node classification algorithm should satisfy; (2) *Theory*: NetConf is grounded in a Bayesian-theoretic framework to model uncertainties, has a closed-form solution and comes with precise convergence guarantees; (3) *Practice*: Our method is easy to implement and scales linearly with the number of edges in the graph. On experiments using real world data, we always match or outperform BP while taking less processing time.

3.1 Introduction

Suppose Smith has to choose between iOS and android phones based on inputs from Alice and Bob (Figure 3.1). Alice (pink/dotted), a stubborn tech-geek, after some research believes that iOS is (60-40) better than android. Non-techie Bob (green/solid) favors android (65-35). Which phone would Smith buy? If Smith takes into account only friends' beliefs, he would be swayed by Bob towards android; however, considering their certainty/stubbornness, he would choose iOS. In an online setting, knowing the browsing and buying patterns of Alice and Bob, what ad (iOS/android phone) should we show Smith? The fundamental question is: how can we capture these notions of *certainty*/*stubbornness* and leverage them to classify nodes in a network?

Network effects appear in many real life scenarios, usually as homophily ("birds of a feather flock together"), or heterophily ("opposites attract") and occasionally a combination of both.



(a)

Author	Score	H-index
Michael J. Carey	2.23	48
Rakesh Agrawal	2.20	96
Jiawei Han	2.00	139
Hamid Pirahesh	1.94	40
David J. DeWitt	1.84	81
Serge Abiteboul	1.80	77

(b)

Author	Score	H-index
Jiawei Han	1.00	139
Annie W. Shum	1.00	-
Werner Kießling	1.00	-
Xiaofang Zhou	1.00	36
Bertram Ludäscher	1.00	45
Amarnath Gupta	1.00	-

(c)

Figure 3.1: (a) Motivation: Who sways our opinion? Alice (certain, 60-40 iOS) or Bob (uncertain, 65-35 android)? (b) Top DB authors using NetConf (c) Top DB authors using BP (ties broken randomly). H-index was obtained from google scholar or `http://web.cs.ucla.edu`.

Knowing the nature of network effects that apply in a given scenario, we may reason from observed training cases directly to test cases; this is called *transductive* inference. Belief Propagation (BP) [YFW03] has been successfully used to perform such inference in numerous areas [ACF13, CKHF11].

However, BP still suffers from one big limitation: it does not take the uncertainty of beliefs into account. Mathematically, BP computes point estimates only, as opposed to full distributions capturing the uncertainty in the beliefs. Thus, when propagating information, BP treats certain and uncertain nodes with equal weight, resulting in counter-intuitive responses, like recommending android to Smith in Figure 3.1.

The intuition pays off, as is seen from Figure 3.1(b) and Figure 3.1(c). Our method, NetConf (NETwork effects with CONfidence) takes certainty into account, and produces a sound ranking of database authors (from the DBLP co-authorship network – see Section 3.5 for more details).

The list of top five authors using NetConf (Figure 3.1(b)) includes authors who wrote many milestone database papers and collaborated with many well-known DB authors. In contrast, BP (Figure 3.1(c)) ignores certainty and results in *numerous* authors having perfect belief score and tying in first place; for several of them we could not find the h-index ('dash'). Informally, the problem we address is the following:

Problem 3.1: Node Classification (With Certainty)

- **Given** a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, labels $l_v \in \{1, 2, \ldots, k\}$ for a subset of the nodes $v \in \mathcal{V}$ (with their uncertainties) and the nature of network effects (e.g., homophily),
- **Find** the probability (belief/leaning) $b_u(i)$ that node u has label i along with a measure of certainty (stubbornness).

The main ideas behind our method are to: (i) model beliefs as Dirichlet distributions to capture uncertainty and (ii) use multinomial counts as messages to propagate these uncertainties along the edges of the network. Our contributions are as follows:

- *Theory*: We propose axioms that every network-effect method should obey; and a Bayesian theoretic model for uncertainty. These lead to our proposed NetConf, which has a closed-form solution (Theorem 3.2) and precise convergence guarantees (Theorem 3.3).
- *Practice*: NetConf is more accurate than BP, as we show with real data; it scales linearly with the number of edges and is usually faster than BP.

3.2 Background

Belief propagation (BP, in short), introduced by Judea Pearl [Pea14] is a general technique to perform *approximate inference* in various graphical models such as Bayesian networks, pairwise Markov random fields and factor graphs [YFW03]. Due to our interest in solving the node classification problem in an undirected graph, we will restrict our discussion of BP to pairwise Markov random fields.

The core idea in BP is for each node u to maintain its *belief* $\mathbf{b}_u$, a k-dimensional vector (where k is the number of classes) in which the i^{th} entry indicates the probability that node u belongs to class i. The belief of a node evolves as it receives *messages* from its neighbors. A message $\mathbf{m}_{vu}$ sent from v to u encodes v's belief about what class the node u should belong to.

Beginning with *prior beliefs* $\mathbf{e}_u$ for each node $u \in \mathcal{V}$, the algorithm iteratively propagates messages and computes beliefs guided by the following update rules.

$$b_u(i) \leftarrow \frac{1}{Z_u} e_u(i) \prod_{v \in \mathcal{N}(u)} m_{vu}(i) \tag{3.1}$$

$$m_{vu}(i) \leftarrow \sum_{j=1}^{k} H(i,j) e_v(j) \prod_{w \in \mathcal{N}(v) \setminus u} m_{wv}(j) \tag{3.2}$$

Here, Z_u is a normalization constant which ensures that the beliefs sum up to 1. The $k \times k$ matrix $\mathbf{H}$ is the *edge potential* or *compatibility matrix*, which captures the *affinity* between the classes. The larger an entry $H(i,j)$, the more likely a node with class i connects to a node with class j. Thus, it can encode any kind of *network effects* such as (a) *homophily* (Figure 3.2(a)), (b) *heterophily* (Figure 3.2(c)) or (c) a combination there of, for more than two classes.

Further, observe that, when v sends a message to u, it does not take into account the message it previously received from u. This is known as *echo-cancellation*.

The method converges to exact marginals only in graphs without loops [Pea14] and in certain special cases [MK07]. In the presence of loops, the algorithm is not guaranteed to converge to the true marginals, or even converge at all. However, in practice, *loopy* belief propagation has been found to approximate the true marginals well [MWJ99] in a variety of applications [CKHF11, FH06, PCWF07].

3.3 Axioms

Figure 7.1a demonstrated that the direct application of BP (or similar algorithms) to node classification problems in a graph often *leads to counter-intuitive results*. This phenomenon is com-

	Conservative	Progressive
Conservative	0.9	0.1
Progressive	0.1	0.9

(a)

	A	B
A	0.5	0.5
B	0.5	0.5

(b)

	Silent	Talkative
Silent	0.2	0.8
Talkative	0.8	0.2

(c)

Figure 3.2: Example edge compatibility matrices H for a binary class problem. (a) Homophily: friendship (b) No network effects: blood group (c) Heterophily: dating

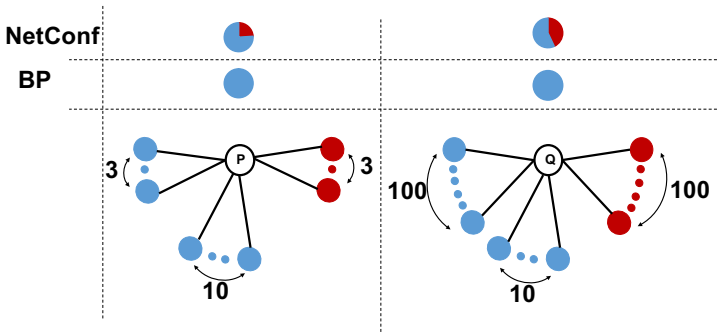


Figure 3.3: Ratio vs. Difference: BP gives a strong *blue* prediction for Q even though Q has an fairly equal number of *red* and *blue* neighbors.

mon; an another example is Figure 3.3. BP's results depend on the difference in the number of blue and red neighbors, but not the actual ratio, as one would desire.

The key to address these problems is to quantify the *uncertainty* in beliefs using distributions. In this section, we set up three axioms that our proposed method, operating on belief distributions, must obey.

Axiom 3.1: No Network Effects

In the absence of network effects, i.e., when the class labels are indifferent to each other, the final belief distribution of every node should match its prior belief distribution.

Axiom 3.2: Certainty Pulls

In the presence of network effects, all else being equal, neighbors with more *certain* belief distributions have a greater influence on a node's belief distribution. Informally, stubborn neighbors are more convincing.

Table 3.1: Notation

Entity/Operator	Notation
Scalar	lowercase, italics; e.g., n, k
Vector	bold, lowercase, without tilde; e.g., $\mathbf{b}_u, \breve{\mathbf{e}}_u$
Distribution	bold, lowercase, with tilde; e.g., $\tilde{\mathbf{b}}_u, \tilde{\mathbf{m}}_{vu}$
Matrix	bold, uppercase; e.g., $\breve{\mathbf{B}}, \mathbf{H}$
Vectorization	*vec*$(.)$
Set	calligraphic, capital; e.g., $\mathcal{V}, \mathcal{E}$
Kronecker product	$\otimes$
Vector/matrix entry	Not bold; e.g., $b_u(i), H(i,j)$
Spectral radius	$\rho(.)$

Axiom 3.3: Certainty Pools

In the presence of network effects, all else being equal, an increase in *certainty* of a neighbor's belief distribution makes a node's belief distribution more *certain*. Informally, *stubborn* neighbors make you more *stubborn*.

As we will see later, Equation (3.7) ensures that our proposed NetConf obeys Axiom 3.1, by propagating flat (uninformative) distributions. Our update rules together ensure that a node with high certainty sends a heavy-weight (as measured by L_1 norm) message according to Equation (3.8), which in turn has a greater influence on its neighbors' beliefs (Axiom 3.2) and increases their certainty (Axiom 3.3) according to Equation (3.9). These are further illustrated using an example in Section 3.5.1. We now describe our approach.

3.4 Proposed Method

In Figure 3.1, Alice is lukewarm towards iOS but very certain about her opinion, while Bob is the reverse. Thus, we need to capture both the leaning/belief of a node (e.g., preference to iOS vs android) as well as its stubbornness/certainty. At a high level, the heart of our idea is to use a Beta distribution with two parameters $(\alpha + 1, \beta + 1)$ as depicted in Figure 3.1. The leaning of a node is the ratio $\frac{\alpha}{\alpha+\beta}$, while its certainty is the height of the spike of the Beta distribution captured through $\alpha + \beta$. For a multi-class case, we generalize this to the *Dirichlet* distribution. Our approach is based on the following steps:

- **Dirichlet Beliefs:** The *D-belief* (*Dirichlet-belief*) $\breve{\mathbf{b}}_u$ of a node u is a k-d vector of reals which parameterize its belief distribution.
- **Multinomial Messages:** The *D-message* $\breve{\mathbf{m}}_{vu}$ from node v to u is a k-d vector of multinomial counts.

Table 3.2: Nomenclature

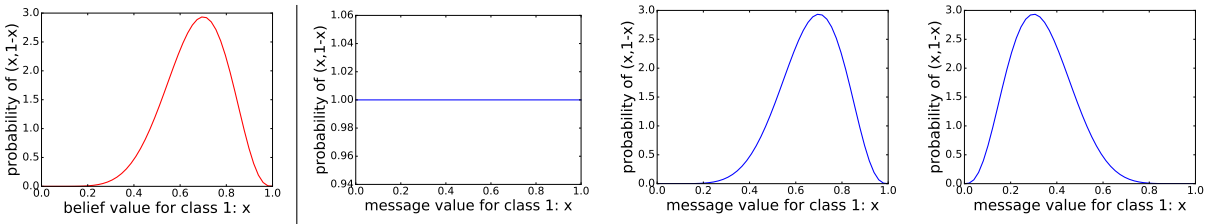
Symbol	Meaning
n	$\lvert\mathcal{V}\rvert$, #nodes in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
k	number of classes
u, v, w	nodes
i, j	classes
$\mathbf{b}_u, \mathbf{e}_u$	k-dim final, prior belief vectors of u
$\mathbf{m}_{vu}$	k-dim message vector from v to u
$\tilde{\mathbf{b}}_u, \tilde{\mathbf{e}}_u$	final, prior belief distributions of u
$\tilde{\mathbf{m}}_{vu}$	message distribution from v to u
$\breve{\mathbf{b}}_u, \breve{\mathbf{e}}_u$	k-dim final, prior D-belief vectors of u
$\breve{\mathbf{m}}_{vu}$	k-dim D-message vector from v to u
$\breve{\mathbf{B}}, \breve{\mathbf{E}}$	$n \times k$ final, prior D-belief matrices
$vec(\breve{\mathbf{B}})$, $vec(\breve{\mathbf{E}})$	$nk \times 1$ vectorized matrices $\breve{\mathbf{B}}, \breve{\mathbf{E}}$
$\mathbf{x}_u$	k-dim point belief from $\breve{\mathbf{b}}_u$ or $\breve{\mathbf{e}}_u$
ϕ	continuous potential function
$\mathbf{H}$	$k \times k$ compatibility matrix
$\mathbf{M}$	$k \times k$ modulation matrix
$\mathbf{A}$	$n \times n$ adjacency matrix
$\mathbf{D}$	$n \times n$ diagonal degree matrix

- **Network Effects:** The *modulation* matrix $\mathbf{M}$ is derived carefully from the compatibility matrix $\mathbf{H}$ (to obey Axiom 3.1).
- **NetConf Update Rules:** We derive update rules (Equation (3.8) and Equation (3.9)) in terms of D-beliefs, D-messages and modulation matrix from Yedidia's update rules (Equation (3.1) and Equation (3.2)).
- **Closed-Form Solution:** From these update rules, we derive NetConf's recursive matrix equation (Theorem 3.1), compute the closed-form solution (Theorem 3.2), and provide *necessary and sufficient* convergence guarantees (Theorem 3.3).

Table 3.1 summarizes the notation and Table 3.2 lists the frequently used symbols. The rest of the section describes the above steps in detail.

3.4.1 Dirichlet Beliefs

A principled way to model the uncertainty in k-d beliefs is through a distribution having a $k-1$-d simplex as support, namely, the *Dirichlet distribution*. Its probability density function is given by: $p(\mathbf{x}; \alpha) \propto \prod_{i=1}^{k} x_i^{\alpha_i - 1}$. The *concentration parameters* $\alpha_1, \ldots, \alpha_k$ are k real-valued numbers which control the spread of the distribution in space. Let us use D-belief ($\breve{\mathbf{b}}_u$) (analogously,



(a) Echo-cancelled belief distribution $\tilde{\mathbf{b}}_{v\setminus u}$

(b) $\tilde{\mathbf{m}}_{vu}$ is flat for no network effects ($\epsilon = 0$)

(c) $\tilde{\mathbf{m}}_{vu} \equiv \tilde{\mathbf{b}}_{v\setminus u}$, for perfect homophily ($\epsilon = +0.5$)

(d) $\tilde{\mathbf{m}}_{vu} \equiv$ flipped $\tilde{\mathbf{b}}_{v\setminus u}$, for perfect heterophily ($\epsilon = -0.5$)

Figure 3.4: Understanding the corner cases of the continuous potential function: A sample 2-class echo-cancelled belief distribution and the corresponding message distributions for the different network effects

D-prior $\breve{\mathbf{e}}_u$) to denote the parameters of u's belief distribution minus 1.

$$\tilde{\mathbf{b}}_u(\mathbf{x}_u) = \mathit{Dir}(\mathbf{x}_u; \breve{\mathbf{b}}_u + \mathbf{1}) \tag{3.3}$$

$$\tilde{\mathbf{e}}_u(\mathbf{x}_u) = \mathit{Dir}(\mathbf{x}_u; \breve{\mathbf{e}}_u + \mathbf{1}) \tag{3.4}$$

As the scale of D-belief increases, the distribution begins to get peakier (*certain*) around its mean; hence, we may quantify the *certainty in belief* as Certainty($\breve{\mathbf{b}}_u$) $= \sum_i \breve{b}_u(i)$. Our richer model for beliefs maintains only k-parameters at every node, similar to BP.

3.4.2 Multinomial Messages

If beliefs are distributions, how should we characterize messages? The key lies in interpreting Equation (3.1) as an equation that guides *Bayesian posterior estimation*:

$$\tilde{\mathbf{b}}_u(\mathbf{x}_u) \propto \tilde{\mathbf{e}}_u(\mathbf{x}_u) \prod_{v \in \mathcal{N}(u)} \tilde{\mathbf{m}}_{vu}(\mathbf{x}_u) \tag{3.5}$$

We hypothesize that the message distributions are the likelihood of *observations* made by a node about its neighbors. For tractability of estimation (using conjugacy of Dirichlet-Multinomial distributions), we let observations be *multinomial counts*. Accordingly, the message distribution $\tilde{\mathbf{m}}_{vu}$ from v to u is the likelihood of the message counts (D-message $\breve{\mathbf{m}}_{vu}$) under the belief distribution $\tilde{\mathbf{b}}_u(\mathbf{x}_u)$ of the node which receives the message:

$$\tilde{\mathbf{m}}_{vu}(\mathbf{x}_u; \breve{\mathbf{m}}_u) \propto \prod_{i=1}^{k} x_u(i)^{\breve{m}_u(i)}$$

Plugging this in Equation (3.5), we derive NetConf's first update rule:

$$\breve{\mathbf{b}}_u \leftarrow \breve{\mathbf{e}}_u + \sum_{v \in \mathcal{N}(u)} \breve{\mathbf{m}}_{vu}$$

Network effects	ϵ	Checkpoint for continuous potential function ϕ	Modulation matrix $\mathbf{M}$	Geometry (Figure 3.5)
None	0	$\int_{\mathbf{x}_v} \phi_{\epsilon=0}(\mathbf{x}_u, \mathbf{x}_v)\tilde{\mathbf{b}}_{v\setminus u}(\mathbf{x}_v) \propto 1$	$\mathbf{M} = \mathbf{0}$	Point O (origin)
Perfect homophily	0.5	$\int_{\mathbf{x}_v} \phi_{\epsilon=0.5}(\mathbf{x}_u, \mathbf{x}_v)\tilde{\mathbf{b}}_{v\setminus u}(\mathbf{x}_v) \propto \tilde{\mathbf{b}}_{v\setminus u}(\mathbf{x}_u)$	$\mathbf{M} = \mathbf{I}$	Point A (identical to $\breve{\mathbf{b}}_{v\setminus u}$)
Perfect heterophily	-0.5	$\int_{\mathbf{x}_v} \phi_{\epsilon=-0.5}(\mathbf{x}_u, \mathbf{x}_v)\tilde{\mathbf{b}}_{v\setminus u}(\mathbf{x}_v) \propto \tilde{\mathbf{b}}_{v\setminus u}(\mathbf{1} - \mathbf{x}_u)$	$\mathbf{M} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Point B (image about $x = y$)

Table 3.3: NetConf corner cases of network effects: Checkpoints for ϕ and corresponding instantiations of $\mathbf{M}$

3.4.3 Network Effects

Since now messages and beliefs are (continuous) distributions instead of vectors, the message update rule in Equation (3.2) needs to be adapted. We use a *continuous potential function* analogous to the compatibility matrix $\mathbf{H}$ in the discrete setting.

$$\tilde{\mathbf{m}}_{vu}(\mathbf{x}_u) \propto \int_{\mathbf{x}_v} \phi(\mathbf{x}_u, \mathbf{x}_v) \underbrace{\tilde{\mathbf{e}}_v(\mathbf{x}_v) \prod_{w \in \mathcal{N}(v)\setminus u} \tilde{\mathbf{m}}_{wv}(\mathbf{x}_v)}_{\tilde{\mathbf{b}}_{v\setminus u}} \tag{3.6}$$

Suppose the compatibility matrix $\mathbf{H}$ for a two class problem is $\begin{pmatrix} 0.5+\epsilon & 0.5-\epsilon \\ 0.5-\epsilon & 0.5+\epsilon \end{pmatrix}$ where ϵ indicates the nature of network effects. $\epsilon = 0.5$ is perfect homophily; $\epsilon = -0.5$ is perfect heterophily; $\epsilon = 0$ is the case of no network effects. Intermediate positive and negative values correspond to varying degrees of homophily and heterophily respectively.

Denote with ϕ_ϵ the (unknown) continuous potential function that reflects the corresponding scenario for a specific value of ϵ. Let $\tilde{\mathbf{b}}_{v\setminus u}$ be the echo-cancelled belief distribution from Equation (3.6). It is desirable that the checkpoints in Table 3.3 hold, as also illustrated in Figure. 3.4. The intuition is as follows: (1) for no network effects, the message should not prefer any belief value over the other (flat distribution); (2) for perfect homophily, a node believes about its neighbors what it believes about itself; (3) for perfect heterophily, a node believes about its neighbors the opposite of what it believes about itself.

Despite the mathematical niceness of the above formulation, it has proved hard to define a potential function that (i) preserves the functional form of message distributions, (ii) satisfies the checkpoints in Table 3.3, and (iii) ensures efficient computation. Thus, we propose to approximate the continuous potential function ϕ that operates on the distributions by a *modulation matrix* $\mathbf{M}$ that operates on the corresponding hyperparameters. Following the update rule of the belief distribution, the message update for the hyperparameters is defined by $\breve{\mathbf{m}}_{vu} \leftarrow \mathbf{M}(\breve{\mathbf{e}}_v + \sum_{w \in \mathcal{N}(v)\setminus u} \breve{\mathbf{m}}_{wv})$.

To formally define the modulation matrix $\mathbf{M}$, let us visualize the D-beliefs and D-messages for a two class problem as points on a 2D plot, as shown in Figure 3.5. The x-axis represents the D-score of a belief or message for the first class, while the y-axis represents the D-score for the second class. Let A represent the D-scores of the echo-cancelled belief of node u, i.e., $\breve{\mathbf{b}}_v - \breve{\mathbf{m}}_{uv}$. The three conditions on ϕ determine how the modulation matrix $\mathbf{M}$ is defined for the corner cases of $\epsilon = 0.5, 0, +0.5$ – these correspond to points A, O and B in Figure 3.5 respectively

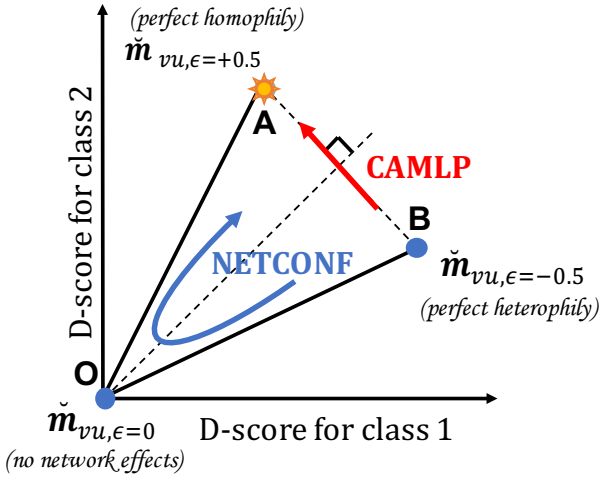


Figure 3.5: Illustration of modulated messages as a function of ϵ: Proposed NetConf follows blue arrow (as ϵ increases from -0.5 to +0.5) and sends (0,0) message for no network effects. Messages according to our competitor CAMLP [YFK16] follow the red arrow and violate Axiom 3.1 (no network effects).

(see also Table 3.3). For any intermediate positive value of ϵ, we propose a linear interpolation and transmit the D-message lying on the line AO. Similarly, for intermediate negative values of ϵ, the D-message takes a value lying on OB. Hence, the modulation matrix for a two-class problem is given by

$$\mathbf{M} = 2 \begin{pmatrix} L(\epsilon) & L(-\epsilon) \\ L(-\epsilon) & L(\epsilon) \end{pmatrix}$$

where $L(.)$ is the *Lasso operator* defined as $L(x) = x$ for $x > 0$, and 0 otherwise. This can be generalized to the k-class case as:

$$\mathbf{M} = \frac{k}{k-1} L \left(\mathbf{H} - \frac{1}{k} \right) \tag{3.7}$$

NetConf obeys Axiom 3.1: In the absence of network effects, $\mathbf{M} = \mathbf{0}$. This makes all message counts zero (i.e., message distributions flat), hence leaving the belief distributions of all nodes unchanged.

3.4.4 Putting Things Together: NetConf

The update rules for NetConf, in terms of the modulation matrix $\mathbf{M}$ (Equation (3.7)) can be summarized as:

$$\check{\mathbf{b}}_u \leftarrow \check{\mathbf{e}}_u + \sum_{v \in \mathcal{N}(u)} \check{\mathbf{m}}_{vu} \tag{3.8}$$

$$\check{\mathbf{m}}_{vu} \leftarrow \mathbf{M}(\check{\mathbf{e}}_v + \sum_{w \in \mathcal{N}(v) \setminus u} \check{\mathbf{m}}_{wv}) \tag{3.9}$$

NetConf obeys Axiom 3.2 and Axiom 3.3: A node with high certainty sends a heavy-weight (as measured by L_1 norm) message due to Equation (3.8). This increases its influence on its neighbors' beliefs (Axiom 3.2) and hence their certainty (Axiom 3.3) according to Equation (3.9).

While in principle one can simply invoke the previous two update equations several times until the messages and beliefs converge, we infer a more efficient variant that avoids computing messages at all. We will use the following notation. Let $\mathcal{G}$ be an unweighted undirected graph on n nodes, with adjacency matrix $\mathbf{A}$. Let $\mathbf{D}$ be the diagonal degree matrix, where $D(q, q) = d_q$, the degree of the q^{th} node. Also, suppose that k is the number of classes. Then, we construct the $n \times k$ D-belief matrix $\breve{\mathbf{B}}$ (and correspondingly, the D-prior matrix $\breve{\mathbf{E}}$), by stacking D-belief (resp., D-prior) row vectors of all nodes one below the other. Now, we are ready to state our main theorem.

Theorem 3.1: NetConf

For matrices $\mathbf{A}$, $\mathbf{D}$, $\breve{\mathbf{B}}$, $\breve{\mathbf{E}}$ and $\mathbf{M}$ described as above, the final D-beliefs of nodes are given by the equation system:

$$\breve{\mathbf{B}} = \breve{\mathbf{E}} + (\mathbf{A}\breve{\mathbf{B}}\mathbf{M} - \mathbf{D}\breve{\mathbf{B}}\mathbf{M}^2)(\mathbf{I} - \mathbf{M}^2)^{-1} \tag{3.10}$$

Proof. Rewriting the D-message update rule from Equation (3.9) in terms of D-belief $\breve{\mathbf{b}}_u$, we have

$$\breve{\mathbf{m}}_{vu} \leftarrow \mathbf{M}(\breve{\mathbf{b}}_v - \breve{\mathbf{m}}_{uv})$$

Plugging the message update rule for $\breve{\mathbf{m}}_{uv}$ into the above yields

$$\breve{\mathbf{m}}_{vu} \leftarrow \mathbf{M}(\breve{\mathbf{b}}_v - \mathbf{M}(\breve{\mathbf{b}}_u - \breve{\mathbf{m}}_{vu}))$$

At steady state, we can replace the update sign with an equality and solve for $\breve{\mathbf{m}}_{vu}$, in terms of the steady state D-beliefs $\breve{\mathbf{b}}_u, \breve{\mathbf{b}}_v$. This gives us

$$\breve{\mathbf{m}}_{vu} = (\mathbf{I} - \mathbf{M}^2)^{-1}(\mathbf{M}\breve{\mathbf{b}}_v - \mathbf{M}^2\breve{\mathbf{b}}_u) \tag{3.11}$$

Now, the steady state D-beliefs can be calculated from the steady state D-messages using Equation (3.8).

$$\breve{\mathbf{b}}_u = \breve{\mathbf{e}}_u + (\mathbf{I} - \mathbf{M}^2)^{-1} \sum_{v \in \mathcal{N}(u)} (\mathbf{M}\breve{\mathbf{b}}_v - \mathbf{M}^2\breve{\mathbf{b}}_u)$$

Rewriting this in matrix form using the previously defined matrices ($\breve{\mathbf{B}}, \breve{\mathbf{E}}, \mathbf{A}$ and $\mathbf{D}$) yields Equation (3.10). ■

As shown, Equation (3.10) operates on beliefs only; the messages are not explicitly required. In practice, we can use the above result to compute the final belief matrix via an efficient iterative update of the following form:

$$\breve{\mathbf{B}}^{(t+1)} = \breve{\mathbf{E}} + (\mathbf{A}\breve{\mathbf{B}}^{(t)}\mathbf{M} - \mathbf{D}\breve{\mathbf{B}}^{(t)}\mathbf{M}^2)(\mathbf{I} - \mathbf{M}^2)^{-1} \tag{3.12}$$

Dataset	Nodes	Edges	Description	Classes
POLBLOGS [AG05]	1490	19090	Political blog hyperlink network	Democrat/Republican
COAUTHOR [SHGY09]	28702	66832	Citation network	4 areas - DB, DM, AI, IR
POKEC [TZ12]	1632803	30622564	Friendship network in Slovakia	Male/female (slight heterophily)

Table 3.4: Datasets used

Weighted edges: Although our proof assumes unweighted edges, it can be easily shown that all our theorems hold for weighted adjacency matrix $\mathbf{A}$ as well.

3.4.5 Closed-Form Solution and Convergence

Before providing theoretical guarantees for our algorithm, we review two useful matrix algebra concepts.

Definition 3.1: Matrix Vectorization [HS81]

Vectorization of an $m \times n$ matrix converts it into a $mn \times 1$ vector given by:

$$vec(\mathbf{X}) = [x_{11}, \ldots, x_{n1}, x_{12}, \ldots, x_{n2}, \ldots, x_{1n}, \ldots, x_{nn}]^T$$

where x_{ij} denotes the element in the i^{th} row and j^{th} column of matrix $\mathbf{X}$.

Lemma 3.1: Roth's Column Lemma [HS81]

For any three matrices $\mathbf{X}, \mathbf{Y}$ and $\mathbf{Z}$,

$$vec(\mathbf{XYZ}) \quad = \quad (\mathbf{Z}^T \otimes \mathbf{X})\, vec(\mathbf{Y}) \tag{3.13}$$

where $\otimes$ is the Kronecker product [HS81].

Theorem 3.2: Closed Form Solution

For matrices $\mathbf{A}$, $\mathbf{D}$, $\mathbf{M}$ and vectors $vec(\breve{\mathbf{B}})$ and $vec(\breve{\mathbf{E}})$ described as above, the closed form solution for D-beliefs is

$$vec(\breve{\mathbf{B}}) = (\mathbf{I} - (\mathbf{M}\hat{\mathbf{M}})^T \otimes \mathbf{A} + (\mathbf{M}^2\hat{\mathbf{M}})^T \otimes \mathbf{D})^{-1} vec(\breve{\mathbf{E}}) \tag{3.14}$$

where $\hat{\mathbf{M}} = (\mathbf{I} - \mathbf{M}^2)^{-1}$.

Proof. The theorem can be proved by vectorizing Equation (3.10) and applying Roth's column lemma. ■

Theorem 3.3: Fixed Point and Convergence

The iterative updates in Equation (3.12) converge to a unique fixed point, for arbitrary initialization of the D-belief matrix, if and only if the spectral norm of $(\mathbf{M}\hat{\mathbf{M}})^T \otimes \mathbf{A} + (\mathbf{M}^2\hat{\mathbf{M}})^T \otimes \mathbf{D}$ is less than 1.

$$\text{NetConf converges} \Leftrightarrow$$
$$\rho\left((\mathbf{M}\hat{\mathbf{M}})^T \otimes \mathbf{A} + (\mathbf{M}^2\hat{\mathbf{M}})^T \otimes \mathbf{D})\right) < 1 \tag{3.15}$$

Here, $\hat{\mathbf{M}} = (\mathbf{I} - \mathbf{M}^2)^{-1}$.

Proof. The Jacobi method of solving a system of linear equations [Saa03] states that a linear equation system of form $\mathbf{x} = (\mathbf{I} - \mathbf{P})\mathbf{y}$ converges if and only if $\rho(\mathbf{P}) < 1$.

Rewriting the update rule in Equation (3.12) in terms of vectorized D-priors and D-beliefs and applying the above result proves the theorem. ■

In practice, convergence may be ensured by setting $\mathbf{M}$ as $c\mathbf{M}$, where $c > 0$ is an appropriately chosen constant according to Theorem 3.3. Here, c can be interpreted as the *modulation decay factor* for message propagation.

3.5 Experiments

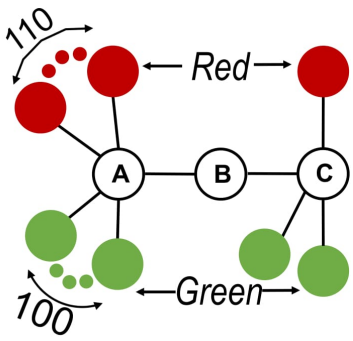
In this section, we (1) present a case study to demonstrate how the top competitors, unlike NetConf, violate our axioms and (2) experimentally verify the scalability and effectiveness of NetConf.

3.5.1 Synthetic Data

We present a case study (Figure 3.6(a)) to illustrate how major competitors disobey our axioms. Here, A, B and C are the core nodes (unlabeled). Given the labels for the remaining peripheral nodes (red/green) and homophily network effects, we investigate the belief/leaning scores assigned by NetConf, BP and CAMLP.

In experiments, we use $[0.1, 0.9]$ and $[0.9, 0.1]$ as prior for the red (top) and green (bottom) with nodes. The core nodes are given uniform prior $[0.5, 0.5]$. Compatibility matrix from Eq. 3.16 with $\epsilon = 0.4$ is used, with CAMLP's β set to the recommended default of 0.1. The belief/leaning returned by the three methods are tabulated in Figure 3.6(b).

$$\mathbf{H} = \begin{pmatrix} 0.5+\epsilon & 0.5-\epsilon \\ 0.5-\epsilon & 0.5+\epsilon \end{pmatrix} \tag{3.16}$$



Method	A	B	C
BP	**6.20e-9**	**0.2561**	0.5^+
CAMLP	**0.4860**	0.5005	0.5210
NetConf	**0.4833**	**0.4967**	0.5118

Figure 3.6: Case Study: (a) graph with $k = 2$ classes (b) BP vs CAMLP vs NetConf: final belief/leaning for class green (bold = red; 0.5^+ is slightly above 0.5)

All three methods label A correctly as red. CAMLP and NetConf result in a belief value which is close to 0.5 as is desirable. However, BP yields a red belief (≈ 1) despite the comparable number of red and green neighbors, which is counter-intuitive.

The classification of node B illustrates the importance of certainty well. B has two neighbors – the red A and the green C. CAMLP, which does not store/propagate certainty, compute B's belief from those of A and C, resulting in a misclassification (violation of Axiom 3.2). However, NetConf recognizes the high certainty of A ($\approx 60\times$ neighbors) and by giving it higher weight, correctly classifies B.

Similar results were obtained for $\epsilon \in (0, 0.5)$ and $\beta \in (0, 1)$. In sum, NetConf obeys axioms and results in intuitive classification unlike major competitors.

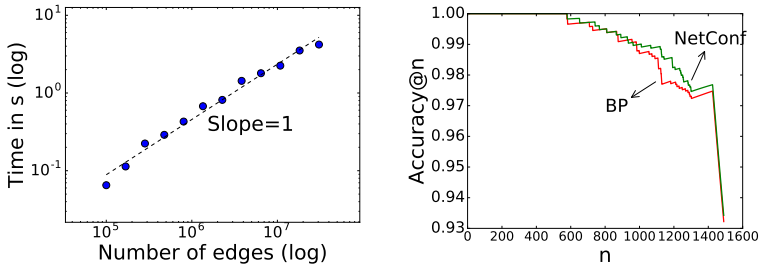
3.5.2 Real-World Data

Our experiments use three diverse publicly available real-world datasets (Table 3.4). We implemented NetConf (iterative version from Equation (3.12)) in MATLAB, as it is well-optimized to handle sparse matrix operations. The modulation decay factor was chosen according to Theorem 3.3. Due to lack of prior work which incorporates certainty in a scalable manner, we resorted to the widely used BP as baseline. All experiments were conducted on 2.7 GHz Intel Core i5 with 16 GB main memory. Our experimental findings can be summarized under the following three categories.

3.5.2.1 Q1. Scalability: How fast and scalable is NetConf with #edges?

We uniformly sampled 150K-30M edges from Pokec network and timed NetConf and BP for 5 iterations (computations only) to allow comparability. In each case, we seeded 20% nodes and used $\mathbf{H}$ from Equation (3.16) with $\epsilon = -0.4$ (heterophily). Figure 3.7(a) plots running time (in seconds; averaged over 10 trials) with the network size in log-log scale.

The plot shows our algorithm scales linearly with the graph size. It was also found to be $\sim 600\times$ faster than a MATLAB implementation of BP by avoiding loops and heavy-weight



(a) Scalability **(b) Accuracy@n**

Figure 3.7: NetConf is (a) scalable (b) outperforms the baseline, achieving better accuracy and precision

operations (similar to [GGKF15]), processing upto $\sim 30M$ edges in a few seconds. This suggests that NetConf is fast and is expected to scale well to large graph applications.

3.5.2.2 Q2. Effectiveness: How accurate is NetConf?

We compare overall accuracy and accuracy@n curve of NetConf against that of BP on three datasets. In all cases, we seeded 30% nodes with their true labels and prior certainty of 1 (due to lack of richer ground truth). Unlabeled nodes were initialized to $[\frac{1}{k}, \ldots, \frac{1}{k}]$ where $k = \#$classes. The compatibility matrix $\mathbf{H}$ from Equation (3.16) with $\epsilon = 0.4$ and -0.4 were used for homophily (Polblogs/Coauthor) and heterophily (Pokec) respectively.

(a) Overall Accuracy: The class with the highest belief/D-belief (BP/NetConf) was assigned as the class for a node, breaking any ties arbitrarily. The accuracy results (Table 3.5) show that NetConf consistently matches or outperforms BP and the differences are statistically significant.

(b) Accuracy@n: We compute the accuracy on top n nodes in a ranking based on the confidence of classification and plotted it as a function of n. The difference in top two beliefs was used as the ranking mechanism for BP; for NetConf, difference in top two D-beliefs was used as it incorporates certainty as well. NetConf emerged as the clear winner on Polblogs dataset, as is evident from Figure 3.7(b). Similar trends were observed in other datasets. These results suggest that NetConf is ideal for *precision-critical* applications, e.g., fraud detection [GGF14b, HSB+16a].

3.5.2.3 Q3. Certainty Scores: Do they make sense?

On the Coauthor network, we rank the authors on their score for class DB (*databases*) and list the top-5 by NetConf (Figure 3.1(b)) and by BP (Figure 3.1(c)). Authors in the former list, with high D-belief for 'DB', have several DB publications and coauthors, a high H-index and several DB-related distinctions. In contrast, BP ignores certainty and produces perfect scores for many authors, as long as they have exclusively DB co-authors and publications, no matter how many or how few. Thus, they all tie in first place; we broke ties arbitrarily and only Prof. Jiawei Han is in both lists.

Table 3.5: Accuracy of BP vs NetConf (averaged over 5 runs): Underlined numbers indicate significant differences $p \leq 0.05$ according to a two-sided sign test.

Accuracy	Polblogs	Coauthor	Pokec
BP (Baseline)	91.38	76.26	73.78
NetConf	**92.40**	**81.89**	**75.02**

In summary, our empirical studies show that NetConf (i) obeys axioms and leads to intuitive classification (ii) is faster than BP and has linear scalability; (ii) never loses to BP and usually outperforms it; (iii) produces certainty scores that reflect our expectations.

3.6 Related Work

Table 3.6 gives an overview of the differences between the methods. In summary, our proposed NetConf is the first method that (i) handles arbitrary network effects, (ii) satisfies all axioms, and (iii) gives a closed-form solution for beliefs and certainties.

Transductive inference, a special case of semi-supervised learning, has attracted a lot of interest [CSZ06a, Zhu05]. Belief Propagation [YFW03] is closely related, and we have described it in Section 3.2. BP has replaced label propagation [Zhu05] and it has been successful on node classification problems, due to its ability to handle both homophily and heterophily. However, its convergence can be guaranteed for some special graphs only [MK07]. Approximations to BP were able to prove convergence, for the 2-class case [KKK+11], the multi-class case [GGKF15], and heterogeneous graphs [EGF+17b]. However, none of the methods can model uncertainty.

Efforts to incorporate uncertainty or confidence are recent [B+08, FHC12, OC12, TC09, YFK15, YFK16]. Except CAMLP [YFK16], all are restricted to homophily effects only. Adsorption [B+08] and its extension MADDL [TC09], which propagate labels by performing a controlled random walk on the graph can only handle homophily. Dirichlet-based Graph Regularization (DRG) [FHC12] assumes every node has a Dirichlet prior and propagates it along edges. However, it is slow, as it needs to solve an optimization problem numerically at every iteration. Transduction Algorithm with Confidence (TACO) [OC12] computes both belief and $k \times k$ uncertainty matrix for all nodes alternatively at every iteration. But, it penalizes high degree nodes for small differences in the beliefs of neighbors even if the neighbors indicate the same class. None of the above methods handles arbitrary network effects. SOCNL [YFK15] and CAMLP both introduce uncertainty, but they both fail Axiom 3.1 (no network effects).

In summary, NetConf is the only method that satisfies all the specifications in Table 3.6.

3.7 Conclusion

We presented NetConf, a method to perform belief propagation along with uncertainties. The main idea was to model beliefs as Dirichlet distributions and messages as multinomial counts. Unlike existing works, NetConf follows proposed axioms, generalizes to arbitrary network

	LP [Zhu05]	SOCNL [YFK15]	BP [YFW03]	Adsorption [B+08]	MADDL [TC09]	DGR [FHC12]	TACO [OC12]	LinBP [GGKF15]	CAMLP [YFK16]	NetConf
Obeys axioms						✓				✓
Homo-/hetero-phily			✓					✓	✓	✓
Scalability	✓	✓	✓	✓	✓		✓	✓	✓	✓
Closed-form	✓	✓						✓	✓	✓

Table 3.6: NetConf has all desirable properties

effects and is highly scalable. NetConf has a closed-form solution and strong convergence guarantees. Our empirical analysis indicated the strong potential of using uncertainty in node classification tasks.

Chapter 4

HOLS: Leveraging Higher-Order Structures

Chapter based on work that appeared at TheWebConf 2020 [EKF20] [PDF].

Do higher-order network structures aid graph semi-supervised learning? Given a graph and a few labeled vertices, labeling the remaining vertices is a high-impact problem with applications in several tasks, such as recommender systems, fraud detection and protein identification. However, traditional methods rely on edges for spreading labels, which is limited by the fact that all edges are not equal. Vertices with stronger connections participate in higher-order structures in graphs, which calls for methods that can leverage these structures in the semi-supervised learning tasks.

Our contributions are three-fold. *First*, we create an information-theoretic metric to quantify the homogeneity of labels in higher-order structures in graphs. We show that across four diverse real-world networks, higher-order structures exhibit more homogeneity of labels compared to edges. *Second*, we create an algorithm, HOLS, for label spreading using higher-order structures. HOLS has strong theoretical guarantees and reduces to standard label spreading [ZBL+03] in the base case. *Third*, we conduct extensive experiments to compare HOLS to several traditional and recent state-of-the-art methods. We show that higher-order label spreading using triangles in addition to edges (HOLS-3) is up to 4.7% better than label spreading using edges alone. Compared to the baselines, HOLS-3 leads to statistically significantly higher accuracy in all-but-one cases. HOLS-3 is also fast and scalable to large graphs, running under 2 minutes in graphs with over 21 million edges.

4.1 Introduction

Given an undirected unweighted graph and some labeled vertices, the graph transductive learning or semi-supervised learning problem (SSL) aims to infer the labels for the remaining unlabeled vertices [ZGL03, ZBL+03, TC09, BMN04, YFW03, KW17, YCS16, APK+19]. Graph SSL finds applications in various use cases: in a social network, we can infer a particular character-

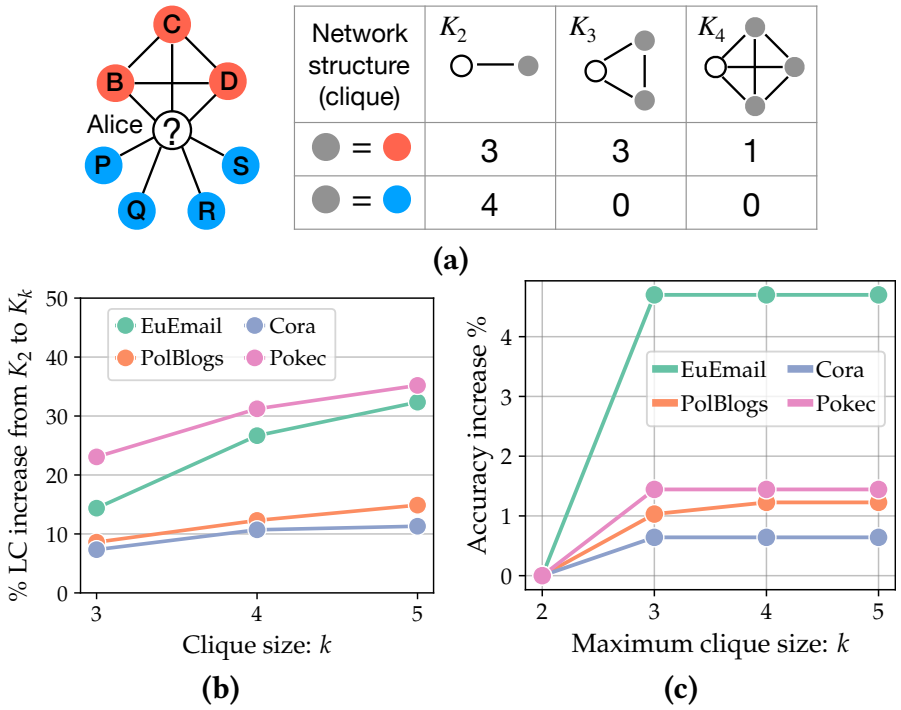


Figure 4.1: ***(a) Motivation*: Existing graph semi-supervised learning approaches propagate labels via edges only and incorrectly classify the unlabeled central vertex 'Alice' as blue. We propose HOLS to leverage the higher-order network structures to propagate vertex labels. HOLS correctly labels the central vertex as red.** ***(b) Higher-Order Label Consistency*: We show that triangles are 7-23% more consistent in vertex labels than edges across four real-world networks. The percentage increase diminishes with clique size k.** ***(c) Higher-Order Label Spreading*: HOLS consistently outperforms the edge-level label spreading algorithms in vertex labeling tasks. Using triangles helps the most. Further higher-order cliques produce little-to-no added benefit.**

istic (e.g., political leaning) of a user based on the information of her friends to produce tailored recommendations; in a user-product bipartite rating network, using a few manually identified fraudulent user accounts, SSL is useful to identify other fraudulent accounts [ACF13, EGF+17b]; from networks of their physical interaction of proteins, SSL can identifying protein functions using a few labels [VFMV03].

Traditional graph SSL algorithms leverage a key property of real-world networks: *the homophily of vertices* [AB02, MSLC01], i.e., the nearby vertices in a graph are likely to have the same label. However, these methods tend to be limited by the fact that all the neighbors of a vertex are not equal. Consider your own friendship network where you have many acquaintances, but only a few close friends. In fact, prior research has shown that vertices with a strong connection participate in several higher-order structures, such as dense subgraphs and cliques [Jac10, JRBT12, SGB17, HR05]. Thus, leveraging the higher-order structure between vertices is crucial to accurately label the vertices.

Let us elaborate this using a small friendship network example, shown in Figure 4.1(a). The central vertex, Alice, participates in a close-knit community with three close friends B, C, and D, all of whom know each other. In addition, she has four acquaintances P, Q, R, and S from different walks of life. Let the vertices be labeled by their political beliefs—vertices B, C, and D have the same blue label; and the rest of the vertices have the red label. Even though Alice has more red connections than blue, the connection between Alice, B, C, and D is stronger as Alice participates in three 3-cliques and one 4-clique with them. In contrast, Alice has no 3- and 4-cliques with P, Q, R, and S. Owing to the stronger connection with the red nodes, Alice should be labeled red as well. However, traditional graph SSL techniques that rely on edges alone label Alice as blue [ZGL03, ZBL+03]. This calls for methods that look beyond edges and leverage strong higher-order structures to label vertices in graph SSL.

In this chapter, we quantify the usefulness of higher-order structures in graph semi-supervised learning. We answer the following key research questions:

- **[RQ1]** Are higher-order network structures more homophilic in labels compared to edges?
- **[RQ2]** How can we leverage higher-order network structures for graph SSL in a principled manner?
- **[RQ3]** Do higher-order structures help improve graph SSL?

We make the following three contributions to answer the research questions.

First contribution: we create a novel metric to measure the homogeneity of labels within higher-order structures in a network. If the homogeneity of a higher-order structure is high, then one can expect label spreading methods using the higher-order structure to aid in vertex labeling tasks. On the contrary, low homogeneity can hurt.

However, how can one compare the homogeneity across different higher-order structures and graphs? For example, how does the homogeneity of triangles compare to the homogeneity of edges in a graph? An appropriate metric should be invariant to the size of the higher-order structure, the number of instances of the structure in the graph, and the marginal distribution of the labels in the graph. Thus, we propose an information-theoretic measure, called the 'Higher-Order Label Consistency', which satisfies these desired properties. HOLC is an entropy score normalized by the baseline distribution of expected homogeneity of labels in similar graphs.

Using the proposed label consistency metric, we measure the higher-order label consistency across four real-world networks. We show the key finding with cliques in Figure 4.1(b). We find that cliques are more homogeneous than edges across all four networks, indicating that the use of cliques can help in efficiently labeling of vertices. Additionally, we observe that the consistency increases with the order of the clique, although with diminishing returns.

Second contribution: we create an algorithm, HOLS, for label spreading using higher-order structures. Spreading labels with higher-order structures is fundamentally different from that of the traditional edge-based label spreading [ZBL+03]. Thus, we develop a general algorithm, HOLS, for graph SSL approaches that leverage higher-order structures. HOLS works for any user-inputted higher-order structure and in the base case, is equivalent to the standard edge-based label spreading. Furthermore, using the equivalence between HOLS and LS on a modified graph, we show that HOLS has a closed-form matrix solution and strong theoretical guarantees.

Third contribution: we conduct extensive experiments to show the effectiveness and speed of HOLS. We use four real-world datasets for our experiments and show the key results in Fig-

ure 4.1(c). Here we show that label spreading via higher-order structures strictly outperforms label spreading via edges. Compared to edges, the gain is the most when using 3-cliques. The use of further higher-order cliques has little additional performance gain. In our experiments shown later in Section 4.5, HOLS outperforms recent state-of-the-art methods by a statistically significant margin across most datasets. Digging deeper into the performance reveals that HOLS has the highest gain when the networks have high label consistency and high clustering coefficient.

Notably, HOLS is practically useful on large networks, with a total run time of under 2 minutes on networks with over 21 million edges. HOLS is over 15 times faster than the recent deep learning baselines and has similar run-time compared to the standard label propagation methods.

4.2 Related Work

Background on Graph SSL: Given a graph and labels on a few vertices, traditional graph semi-supervised learning methods typically infer the labels for all vertices by optimizing a loss function of the form $\mathcal{L} = (1-\eta)\mathcal{L}_s + \eta\mathcal{L}_g$. Here, the first term is the *supervised loss* which imposes a penalty when the inferred values on the labeled vertices differ from their given values and the second term is the *graph loss* which penalizes inferred values that are not *smooth* over the graph structure. A parameter $\eta \in (0, 1)$ trades off the two factors. Various graph SSL methods define their loss functions as variants of the above.

Traditional Graph SSL Approaches: By far, the most widely adopted graph SSL techniques are label propagation [ZGL03] and label spreading [ZBL+03]. Label propagation (LP) clamps labeled vertices to their provided values and uses a graph Laplacian regularization, while label spreading (LS) uses a squared Euclidean penalty as supervised loss and *normalized* graph Laplacian regularization which is known to be better-behaved and more robust to noise [VLBB08]. Both these techniques permit closed-form solution and are extremely fast in practice, scaling well to billion-scale graphs. Consequently, a number of techniques build on top of these approaches, for example, to allow inductive generalization [BMN04, WRC08], to incorporate certainty [TC09], and so on. When the graph is viewed as pairwise Markov random field, belief propagation (BP) [YFW03] may be used to recover the exact marginals on the vertices. BP can handle network effects beyond just homophily; however, it has well-known convergence problems from a practitioner's point of view [SNB+08]. While traditional techniques, in general, show many desirable theoretical properties such as closed-form solution, convergence guarantees, connections to spectral graph theory [ZGL03] and statistical physics [YFW03], as such, they do not account for higher-order network structures.

Recent Graph SSL Approaches differ from traditional SSL methods in training embeddings of vertices to jointly predict labels as well as the neighborhood context in the graph. Specifically, Planetoid [YCS16] uses skipgrams, while GCN [KW17] uses approximate spectral convolutions to incorporate neighborhood information. MixHop [APK+19] can learn a general class of neighborhood mixing functions for graph SSL. As such, these do not incorporate *specific*

higher-order structures provided by the user. Further, their performance in practice tends to be limited by the availability of 'good' vertex features for initializing the optimization procedure.

Hybrid Approaches for Graph SSL: Another way to tackle the graph SSL problem is a hybrid approach to first extract vertex embeddings using an unsupervised approach such as node2vec [GL16], DeepWalk [PAS14] or LINE [TQW+15] and then use the available labels to learn a transductive classifier such as an SVM [Joa99]. Such methods, however, neither have well-understood theoretical properties nor do they optimize for a single objective in an end-to-end manner.

Higher-Order Network Structures: Recent work has shown that graphs from diverse domains have many striking higher-order network structures [BGL16] which can be leveraged to improve graph clustering [YBL18], link prediction [BAS+18, AMA19] and ranking [RRK+19]. Significant recent algorithmic advancements made in counting [JS17] and enumeration [DBS18] of higher-order network structures (esp., cliques) enables and supports the aforementioned applications. From a graph SSL point of view, the explicit use of higher-order network structures has remained limited to belief propagation over 2×2 image cliques to improve image denoising, segmentation and rendering in computer vision [LRHB06, PL08]. Till date, the importance of higher-order network structures in SSL atop explicit graph data has largely remained unexplored and our work aims to address this gap.

Regularization Framework: Recent line of research [NSZ09] has shown that Laplacian regularization method for semi-supervised learning at the regime of a fixed number of labeled points but a large number of unlabeled points is actually not well-posed, and as the number of unlabeled points increases the solution degenerates to a non-informative function. To address this issue, techniques based on ℓ_p-Laplacian regularization [ACR+16, ST17] have been proposed, where p depends the dimensionality of the random graph model. The method we consider in this chapter can be viewed as an alternative to p-Laplacian, based on the symmetric Laplacian of a modified graph constructed based on higher-order motif counts. Establishing the precise nature of connection between these two forms of regularization, however, is beyond the scope of this chapter and is left to future work.

Comparison: We compare the best performing HOLS algorithm (HOLS-3 which uses triangles in addition to edges) *qualitatively* to prominent SSL approaches in Table 4.1 and *quantitatively* via experiments to representative methods from the above categories: LP and LS (traditional), GCN (recent) and node2vec + TSVM (hybrid).

4.3 Higher-Order Label Consistency

The motivation for our work comes from the hypothesis that when real-world networks exhibit *homophily*, higher-order motifs (e.g., triangles) tend to be more homogeneous in labels than edges. In this section, we develop an information-theoretic measure called *label consistency* to

Table 4.1: Qualitative comparison of HOLS-3 (using edges and triangles) with traditional and recent graph SSL approaches.

Desiderata	LP [ZGL03]	LS [ZBL+03]	BP [YFW03]	Planetoid [YCS16]	GCN [KW17]	MixHop [APK+19]	**HOLS-3**
Higher-order structures				✓	✓	✓	✓
Theoretical guarantees	✓	✓	?				✓
Fast algorithm	✓	✓	✓				✓

quantify the extent of label homogeneity across occurrences of any given motif within a large graph and use it to verify our hypothesis on real-world networks.

4.3.1 Notation

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. Edges can be directed and/or weighted; the weight of an edge from vertex i to vertex j is given by w_{ij}. Let $\ell(i) \in \{1, 2, \ldots, C\}$ denote the label of a vertex $i \in \mathcal{V}$. Each vertex has a unique label and there are C possible labels.

For a given higher-order network structure (or motif) κ, let $|\kappa|$ denote its *size* which is its number of participating vertices. For example, when κ is a triangle, $|\kappa| = 3$. Let Q_κ be the set of all occurrences of motif κ in graph $\mathcal{G}$. Let a subgraph $q \in Q_\kappa$ be a specific instantiation of κ. $\mathbb{1}[i \in q]$ denotes that a vertex i is part of the subgraph q and we say that κ is incident on vertex i. $|q| = |\kappa| \; \forall \; q \in Q_\kappa$. Let w_q denote the weight of subgraph q computed from the weights of its incident edges, e.g., via a product.

Following standard practice, we use $H(\cdot)$ to denote entropy and $\mathbb{1}[\cdot]$ to denote the indicator function, which evaluates to 1 when the enclosed expression is true.

4.3.2 Quantifying Label Consistency

When can we say that a given subgraph over a set of vertices is consistent in labels? Intuitively, highest label consistency is achieved when all its participating vertices have the same label and lowest label consistency is achieved when all vertices have different labels. Thus, the homogeneity of labels within a subgraph can be naturally quantified via Shannon's entropy [Sha48] of label distribution over the participating vertices:

Definition 4.1: Label Entropy of a Subgraph

Let q be a subgraph and and let $\hat{\mathbf{p}}_q(c) = \sum_{i \in q} \mathbb{1}\left[\ell(i) = c\right] / |q|$ denote the probability that a randomly chosen vertex from q has a label c. The label entropy of q is the Shannon's entropy of the distribution $\hat{\mathbf{p}}_q$:

$$H(q) = -\sum_{c=1}^{C} \hat{\mathbf{p}}_q(c) \ln_2 \hat{\mathbf{p}}_q(c) \tag{4.1}$$

where $0 \ln_2 0 = 0$ by convention.

Note that $0 \leq H(q) \leq \ln_2 C$ and a lower value signifies a greater homogeneity in labels within subgraph q. Given a motif κ, we may compute the vertex label entropy for a vertex by aggregating the label entropy across its occurrences.

Definition 4.2: Vertex Label Entropy of a Motif

The vertex label entropy of a motif κ incident on a vertex i in a graph $\mathcal{G}$ is the average label entropy over all the instantiations of κ incident on i.

$$H_i(\kappa; \mathcal{G}) = \frac{\sum_{q \in Q_\kappa} \mathbb{1}\left[i \in q\right] w_q H(q)}{\sum_{q \in Q_\kappa} \mathbb{1}\left[i \in q\right] w_q} \tag{4.2}$$

A low value of $H_i(\kappa; \mathcal{G})$ suggests that labels within any randomly chosen occurrence of κ in the neighborhood of vertex i are likely to be homogeneous and hence κ might be a useful network structure to leverage for SSL of vertex i. Note that vertex label entropy is not defined for a motif that does not appear in the neighborhood of interest.

It is crucial to observe that the label entropy values can vary significantly across motifs and graphs depending on (a) the size of the motif $|\kappa|$, (b) the number of possible labels C and (c) the marginal distribution of the labels. To see this, note that the maximum possible entropy itself scales as $\ln_2 \min(|\kappa|, C)$. Thus, we in order to derive a credible measure of label consistency that can be fairly compared across different motifs and graphs, we propose to compute the vertex label entropy of a motif *relative to a common baseline or null model.* Let $\mathcal{F}$ be the distribution of vertex-labeled graphs according to some null model. Then, the vertex label consistency of a motif is defined as follows.

Definition 4.3: Vertex Label Consistency of a Motif

The vertex label consistency of a motif κ incident on a vertex i in a graph $\mathcal{G}$ is defined as the vertex label entropy of κ on i relative to the expected value in the null model $\mathcal{F}$. Mathematically,

$$\lambda_i(\kappa) = 1 - \frac{H_i(\kappa; \mathcal{G})}{\mathbb{E}_{\mathcal{G}' \sim \mathcal{F}}[H_i(\kappa; \mathcal{G}')]} \tag{4.3}$$

Here, $\mathcal{G}'$ is a random graph drawn from the null model $\mathcal{F}$ and $H(\kappa; \mathcal{G}')$ is the vertex label entropy of κ in $\mathcal{G}$ from Definition 4.2. Several such random graphs are drawn and the expected value over all such draws is the normalization denominator. Note that vertex label consistency, similar to vertex label entropy, is only defined for motifs that appear in the neighborhood of a vertex.

A value of $\lambda_i(\kappa) = 1$ indicates perfect vertex label consistency of motif κ: vertices participating in every occurrence of κ in the neighborhood of i have the same label $\ell(i)$, resulting in $H_i(\kappa; \mathcal{G}) = 0$. A value of $\lambda_i(\kappa) = 0$ indicates that the labels of vertices within occurrences of motif are no more consistent than they would be in the null model. Finally, negative values of $\lambda_i(\kappa)$ are also possible. This occurs when the vertex i preferentially attaches to those having a diverse set of labels. In this case, the observed vertex label entropy is higher than is expected in the null model.

What is the right null model to use? One can consider two possibilities: *(i) Label permutation null model* where the vertex-label assignments are shuffled while fixing the graph structure (edges) and the marginal label distribution. *(ii) Edge permutation null model* where the edges are shuffled uniformly at random as in an Erdős-Rényi random graph model, while fixing the vertex-label assignments and hence the marginal distribution of labels. Surprisingly, both null models lead to same value of vertex label consistency as the probability of observing a motif with a particular label configuration according to either null model depends only on the marginal label distribution.

We average the vertex label consistency across all vertices to assign a single label consistency score to a motif. We can then compare this score across different motifs.

Definition 4.4: Label Consistency of a Motif

The label consistency of a motif κ is the average vertex label consistency over all vertices $\mathcal{V}_\kappa = \bigcup_{q \in Q_\kappa} q$ incident with at least one occurrence of κ.

$$\bar{\lambda}(\kappa) = \frac{\sum_{i \in \mathcal{V}_\kappa} \lambda_i(\kappa)}{|\mathcal{V}_\kappa|} \tag{4.4}$$

Using this, we say that a motif κ_1 is more label-consistent than a motif κ_2 if and only if $\bar{\lambda}(\kappa_1) > \bar{\lambda}(\kappa_2)$.

Table 4.2: Statistics of datasets used

Dataset	Domain	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	C
EuEmail [LKF07]	Email communication	1005	16.0K	42
PolBlogs [AG05]	Blog hyperlinks	1224	16.7K	2
Cora [SB13]	Article citations	23.1K	89.1K	10
Pokec [TZ12]	Friendship	1.6M	22.3M	10

Finally, we compare label consistency to two popular graph theoretical measures in the literature. **(i) Clustering Coefficient [WS98, YBL18]:** Despite any apparent similarities between label consistency and clustering coefficient, they capture two very different quantities. Higher-order clustering coefficient deals with the number of observed cliques given all possible cliques that could have existed, and as such, does not use vertex labels. Label consistency, on the other hand, measures the entropy of labels within *any arbitrary motif* across all its *observed* occurrences. **(ii) Assortativity [New03]:** Label consistency is closely related to assortativity coefficient for *discrete* vertex characteristics. However, while assortativity quantifies mixing of vertices only at the level of edges, our proposed metric can handle any general higher-order motifs.

4.3.3 Label Consistency in Real-World Networks

How label-consistent are higher-order motifs in real-world networks? Here, we will answer this question with four diverse real-world datasets from several domains. We will use these datasets for our empirical analysis and experiments throughout the chapter.

4.3.3.1 Dataset

We investigate the patterns of higher-order label consistencies in the following networks. A summary of dataset statistics is provided in Table 4.2.

- **EuEmail [LKF07]** is an e-mail communication network from a large European research institution. Vertices indicate members of the institution and an edge between a pair of members indicates that they exchanged at least one email. Vertex labels indicate membership to one of the 42 departments.
- **PolBlogs [AG05]** is a network of hyperlinks between blogs about US politics during the period preceding the 2004 presidential election. Blogs are labeled as right-learning or left-leaning.
- **Cora [SB13]** is a citation network among papers published at computer science conferences. Vertex labels indicate one of 10 areas that the paper belongs to based on its venue of publication, e.g., Artificial Intelligence, Databases, Networking.
- **Pokec [TZ12]** is the most popular online social network in Slovakia. Vertices indicate users and edges indicate friendships. From the furnished user profile information, we extract the locality or 'kraj' that users belong to and use them as labels.

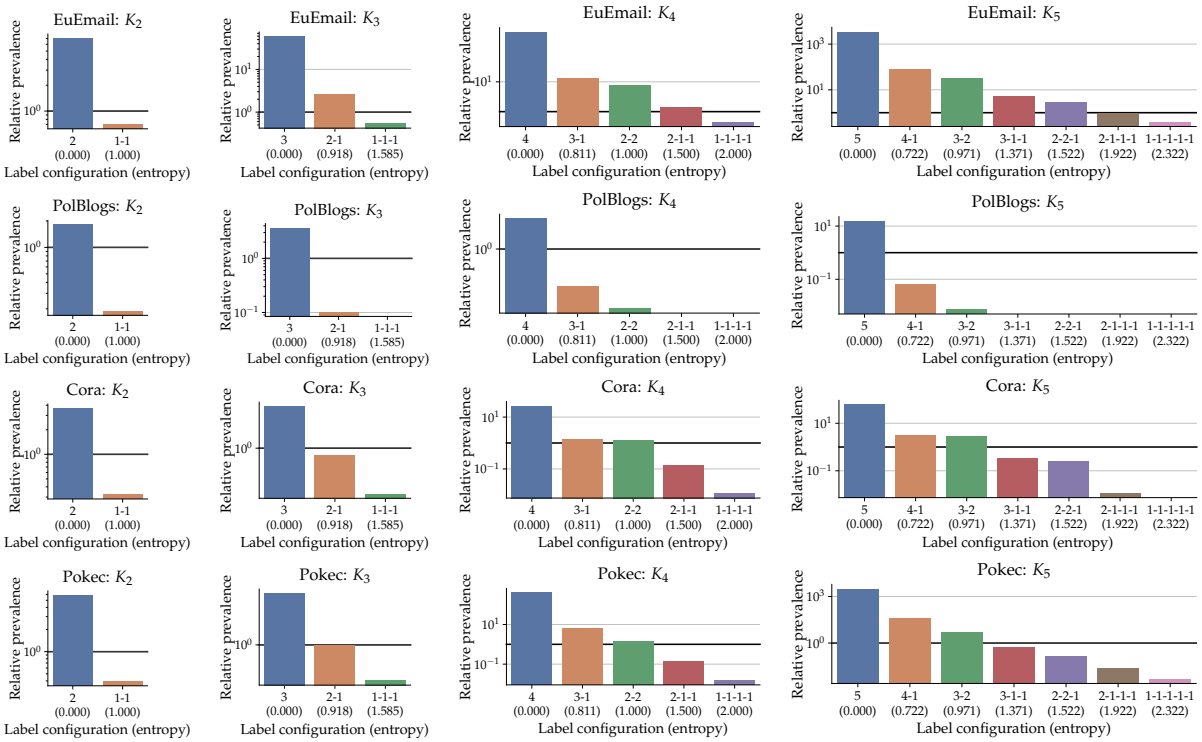


Figure 4.2: ***Most motifs in the real-world networks have low label entropy:*** **Relative prevalence of a label configuration is its observed fraction in the dataset over its probability of occurrence under the null model. Motifs with label configurations having low label entropy are strikingly more prevalent (relative prevalence $\gg 1$) than expected across all datasets, whereas those with high label entropy are unusually rare (relative prevalence $\ll 1$).**

These datasets exhibit homophily [AB02, MSLC01]: people typically e-mail others within the same department; blogs tend to link to others having the same political leaning; papers mostly cite those from the same area; people belonging to the same locality are more likely to meet and become friends. In all cases, we omit self-loops and take the edges as undirected and unweighted.

4.3.3.2 k-Clique Label Consistencies

In this section, we examine the label consistency of k-cliques in real-world networks. We focus on cliques for two reasons: (i) cliques are quintessential dense subgraphs which are important in network analysis and are the essential building blocks of many networks [Jac10, JRBT12, SGB17, HR05]; moreover, (ii) recent advancements enable fast and efficient enumeration of k-cliques [DBS18]. We focus on $k \in \{2, 3, 4, 5\}$ for computational reasons.

Label configuration. We introduce the term label configuration to capture a function of vertex-label assignments that is invariant under the permutation of vertices and labels. For

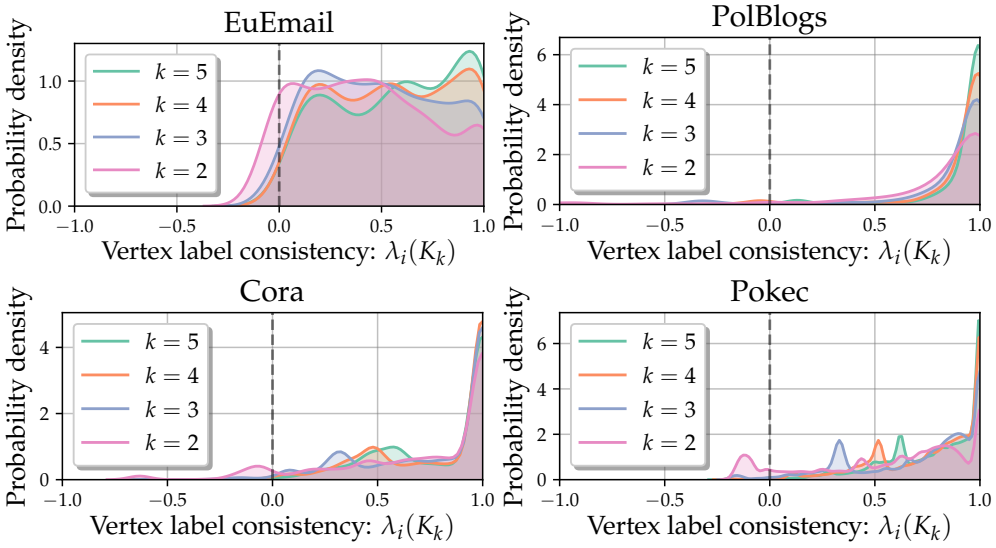


Figure 4.3: ***Real-world networks have high vertex label consistency:*** **Vertices typically have a high positive value of k-clique vertex label consistency, which peak near the highest possible value of 1. Vertices which have no incident k-cliques are ignored.**

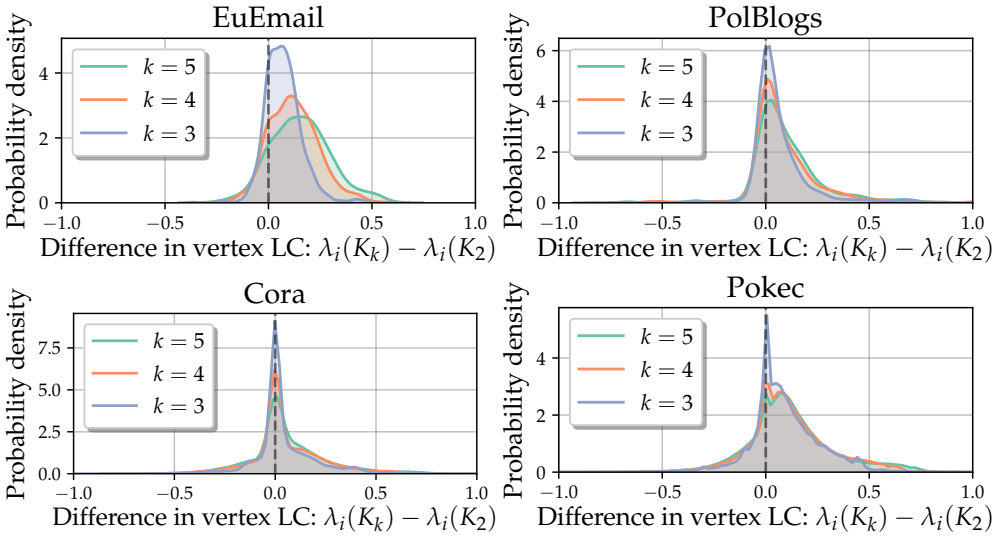


Figure 4.4: ***Larger cliques have higher vertex label consistency:*** **Vertex label consistency of cliques grows with their size k. Thus, the distribution of differences of k-clique and 2-clique vertex label consistencies per vertex lies slightly right of zero, and shifts rightward as k increases. Vertices which have no incident k-cliques are ignored.**

example, a 2-clique has two label configurations: '2' where both incident vertices have the same label (label entropy: 0) and '1-1' where they have different labels (label entropy: 1). A 3-clique has three label configurations: '3' where all three vertices have the same label, '2-1' where two of them share the same label and third vertex has a different label and '1-1-1' where

each vertex has a different label. Similarly, a 4-clique has 5 label configurations (4, 3-1, 2-2, 2-1-1, 1-1-1-1) and a 5-clique has 7 label configurations (5, 4-1, 3-2, 3-1-1, 2-2-1, 2-1-1-1, 1-1-1-1-1). Note that not all label configurations may be possible (e.g., 1-1-1 is impossible for a triangle in a 2-class problem) and still fewer may actually occur in practice. Thus, as a first step toward analyzing label consistency, we study the distribution over k-clique label configurations. Then, we turn to patterns in vertex label consistencies and label consistencies of k-cliques.

Our analysis reveals the following key observations.

Observation 1: Most motifs in real-world networks have low label entropy. Figure 4.2 plots the relative prevalence of k-clique label configurations, i.e., the ratio of observed fraction of k-cliques having a given label configuration to the probability of observing such a configuration under the null model. For example, if the probability of label configuration '1-1' for K_2 is 0.5 (observed) and 0.25 (from null model), its relative prevalence is 2. Label configurations on x-axis are sorted by increasing label entropy (indicated in brackets). We observe that low entropy label configurations are significantly more prevalent in the real-world than expected, while high entropy label configurations are unusually rare than expected. This confirms that real-world k-cliques are indeed more homogeneous in labels than can be explained by random chance.

Observation 2: Real-world networks have high vertex label consistency. A plot of the kernel density distributions of the k-clique vertex label consistencies of vertices is shown in Figure 4.3. We note that scores for most vertices are positive; moreover the distributions are skewed towards the highest possible value of one. Overall, this indicates that k-cliques are highly label-consistent in the neighborhoods of most vertices across all datasets.

Observation 3: Larger cliques have higher vertex label consistency. Are vertices more label-consistent in higher-order cliques than in edges? To answer this, we compute the difference in vertex label consistency wrt. k-cliques and 2-cliques per vertex, for vertices having at least one incident k-clique. The resulting distributions are shown in Figure 4.4. We observe that all the distributions lies towards the right of zero; with positive values of difference being more likely across all values of k and all datasets. Moreover, the distributions shift further towards the right for increasing k (this is most apparent for EuEmail dataset) showing that the vertex label consistency improves with increasing k.

Observation 4: Larger cliques are more label-consistent than edges. Finally, we examine the overall label consistency of k-cliques shown in Figure 7.1(b). The figure reveals that the label consistency increases monotonically with k. Specifically, triangles ($k = 3$) are 7-23% more label-consistent than edges, 4-cliques are 10-31% more label-consistent than edges and 5-cliques are 11-35% more label-consistent than edges. Thus, while triangles provide significant improvements in label consistency over edges, higher-order cliques for $k \geq 4$ provide diminishing returns in label consistency. The absolute values of label consistency for all considered cliques were found to lie in $[0.45, 0.6]$ for EuEmail (lowest), $[0.8, 0.95]$ for PolBlogs (highest), $[0.7, 0.8]$ for Cora and $[0.55, 0.8]$ for Pokec.

Altogether, in this section, we showed that real-world networks have highly label-consistent higher-order cliques. This forms the basis of our higher-order label spreading algorithm that we describe in the next section.

4.4 Higher-Order Label Spreading

In the section, we derive the proposed higher-order label spreading algorithm and shows its desirable theoretical properties.

4.4.1 Generalized Loss Function

Let $\mathcal{K}$ be the set of network structures or motifs (e.g., edges, triangles, diamonds) that we want to leverage for graph semi-supervised learning. Recall that Q_κ is the set of occurrences of a motif $\kappa \in \mathcal{K}$ and each such occurrence $q \in Q_\kappa$ has a weight w_q. Let $\mathbf{y}_i \in \{0,1\}^C$ be the provided label for a labeled vertex i such that $y_{ic} = 1$ if vertex i has a label c and is zero otherwise. We propose to leverage (higher-order) network structures in $\mathcal{K}$ by minimizing:

$$\mathcal{L} = (1-\eta)\mathcal{L}_s + \eta\mathcal{L}_g = (1-\eta)\mathcal{L}_s + \eta \sum_{\kappa \in \mathcal{K}} \alpha_\kappa \mathcal{L}_{g,\kappa} \tag{4.5}$$

where $\mathcal{L}_s$ is the supervised loss and $\mathcal{L}_{g,\kappa}$ is the graph loss with respect to motif κ. A parameter $\eta \in (0,1)$ trades off supervised and graph losses, while $\alpha_\kappa \in (0,1)$ captures the importance weight of κ in semi-supervised learning. Note $\sum_{\kappa \in \mathcal{K}} \alpha_\kappa = 1$.

How do we pick $\mathcal{L}_{g,\kappa}$? Intuitively, the graph loss with respect to a given motif should ensure that the *inferred labels are smooth over all occurrences of the motif*. Thus, for each occurrence of the motif, we propose to penalize the difference in inferred labels of every pair of incident vertices via the squared Euclidean loss as follows:

$$\mathcal{L}_{g,\kappa} = \frac{1}{2} \sum_{q \in Q\kappa} w_q \sum_{i,j \in q} ||\mathbf{x}_i - \mathbf{x}_j||^2 \tag{4.6}$$

Observe the following:

Proposition 4.1: Generalized LP Graph Loss

The graph loss of label propagation [ZGL03] is a special case of the graph loss from Equation (4.5) when we use only 2-clique motifs, i.e., $\mathcal{K} = \{K_2\}$.

Proof. Follows from observing that the set of 2-cliques Q_{K_2} is simply the set of edges and for each $q = \{i,j\} \in Q_{K_2}$, its subgraph weight is $w_q = w_{ij}$ where $\mathbf{W} = [w_{ij}]$ is the graph adjacency. ■

For supervised loss, we employ squared Euclidean penalty: $\mathcal{L}_s = \frac{1}{2}\sum_i ||\mathbf{x}_i - \mathbf{y}_i||^2$.

Next, we show how the graph loss in Equation (4.5) can be viewed as the graph loss for LP on a modified graph where edges have been re-weighted based on motif counts. Define κ-participation matrix as $\mathbf{W}^{(\kappa)} = [w_{ij}^{(\kappa)}]$ where each entry $w_{ij}^{(\kappa)}$ denotes the total weight of κ-motifs that vertices i and j participate in. If $\mathbb{1}[\cdot]$ denotes the indicator function, we have:

$$w_{ij}^{(\kappa)} = \sum_{q \in Q_\kappa} w_q \cdot \mathbb{1}[i \in q \wedge j \in q] \tag{4.7}$$

Observe that each pairwise loss term $||\mathbf{x}_i - \mathbf{x}_j||^2$ in Equation (4.5) appears with a total weight w'_{ij} given by $w'_{ij} = \sum_{\kappa \in \mathcal{K}} \alpha_\kappa w_{ij}^{(\kappa)}$ using which we may simplify the graph loss as:

$$\mathcal{L}_g = \frac{\eta}{2} \sum_{i,j} w'_{ij} ||\mathbf{x}_i - \mathbf{x}_j||^2 \tag{4.8}$$

Thus, Equation (4.8) establishes that the graph loss from Equation (4.5) is equivalent to that of LP on a modified graph with adjacency matrix $\mathbf{W}' = \sum_{\kappa \in \mathcal{K}} \alpha_\kappa \mathbf{W}^{(\kappa)}$ where each edge of the original graph has been re-weighted according to the total weight of κ-motifs it participates in, scaled by the corresponding motif importance α_κ, and finally summed over all such motifs $\kappa \in \mathcal{K}$ of interest. We will use this connection to derive a closed-form solution to HOLS.

Analysis of the running example: Let us return to our running example of the octopus graph, from Figure 4.1(a), to understand why the generalized loss function works well in practice. Let the inferred label for Alice be $\mathbf{x}^*$. By symmetry, all red vertices will have the same inferred label $\mathbf{x}_r$ and all blue vertices have the same inferred label $\mathbf{x}_b$. Let us examine classification results using (i) edges and (ii) edges and triangles with $\alpha_{K_2} = \alpha_{K_3} = 0.5$.

In the first case, the terms in the loss function for $\mathbf{x}^*$ are $3\eta||\mathbf{x}^* - \mathbf{x}_r||^2 + 4\eta||\mathbf{x}^* - \mathbf{x}_b||^2$. The optimal solution is $\mathbf{x}^* = (3\mathbf{x}_r + 4\mathbf{x}_b)/7$, which is closer to the inferred label for the blue vertices. We can show that for any value of η, the inferred label for blue labeled vertices is blue. Thus, Alice is incorrectly assigned a blue label.

In the second case, the terms in the loss function for $\mathbf{x}^*$ are $3 \times 0.5\eta||\mathbf{x}^* - \mathbf{x}_r|| + 4 \times 0.5\eta||\mathbf{x}^* - \mathbf{x}_b||^2 + 3 \times 0.5 \times 2\eta||\mathbf{x}^* - \mathbf{x}_r||^2$. These terms correspond to regularization for the 3 red edges, 4 blue edges and 3 red triangles, respectively (the factor of 2 in the last term appears because every triangle has two loss terms containing $\mathbf{x}^*$). In total, we get $4.5\eta||\mathbf{x}^* - \mathbf{x}_r||^2 + 2\eta||\mathbf{x}^* - \mathbf{x}_b||^2$. The resulting optimal solution is $\mathbf{x}^* = (9\mathbf{x}_r + 4\mathbf{x}_b)/13$. Thus, the unlabeled vertex is always assigned the correct red label for all values of η.

Here we showed that the use of triangles helps to explicitly incorporate complex higher-order structures in label spreading and vertex labeling.

4.4.2 Closed-Form and Iterative Solutions

Let $\mathbf{Y} = [\mathbf{y}_1 \dots \mathbf{y}_N]^T$ and $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_N]^T$ be the $N \times C$ matrices of prior and inferred labels where N is the total number of vertices. Let $\mathbf{D}' = [d'_{ij}]$ be the diagonal degree matrix for the modified graph adjacency $\mathbf{W}' = [w'_{ij}]$. Thus, $d'_{ii} = \sum_j w'_{ij}$ and $d'_{ij} = 0$ if $i \neq j$. Let

Algorithm 1 Higher-Order Label Spreading (HOLS)

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, number of classes C, set of labeled vertices $\mathcal{V}_l \subset \mathcal{V}$ and their labels $\ell : \mathcal{V}_l \to \{1, \ldots, C\}$ (at least one labeled vertex per class)
Parameters: motif set $\mathcal{K}$, motif weights $\alpha_\kappa \in (0,1)$ such that $\sum_{\kappa \in \mathcal{K}} \alpha_\kappa = 1$, weight $\eta \in (0,1)$ for supervised loss
Output: final label assignments $\ell^*(i)$ for all vertices $i \in \mathcal{V}$

procedure HIGHERORDERLABELSPREADING($\mathcal{G}, \mathcal{V}_l, \ell, \mathcal{K}, \alpha, \eta$)
▷ *Construct higher-order normalized graph Laplacian for regularization*
  for $\kappa \in \mathcal{K}$ **do**
    Construct κ-participation matrix $\mathbf{W}^{(\kappa)} = [w_{ij}^{(\kappa)}]$
      ▷ $w_{ij}^{(\kappa)}$: *total weight of κ-motifs where i and j appear together*
  $\mathbf{W}' \leftarrow \sum_{\kappa \in \mathcal{K}} \alpha_\kappa \mathbf{W}^{(\kappa)}$
  $\mathbf{D}' \leftarrow diag(d'_{ii})$ where $d'_{ii} = \sum_j w'_{ij}$
  $\tilde{\mathbf{L}}' \leftarrow \mathbf{D}'^{-1/2} \mathbf{W}' \mathbf{D}'^{-1/2}$
▷ *Construct label matrices $\mathbf{Y} = [y_{ic}]$ (prior) and $\mathbf{X} = [x_{ic}]$ (inferred)*
  $\mathbf{Y} \leftarrow \mathbf{0}_{|\mathcal{V}| \times C}$
  $y_{i\ell(i)} \leftarrow 1 \;\; \forall\, i \in \mathcal{V}_l$
  $\mathbf{X} \leftarrow \mathbf{Y}$
▷ *Label inference using HOLS*
  while not converged **do**
    $\mathbf{X} \leftarrow \eta(\mathbf{I} - \tilde{\mathbf{L}}')\mathbf{X} + (1-\eta)\mathbf{Y}$ ▷ *Equation (4.12)*
  $\ell^*(i) \leftarrow \text{argmax}_c \, x_{ic} \;\; \forall\, i \in \mathcal{V}$
  return ℓ^*

$\mathbf{L}' = \mathbf{D}' - \mathbf{W}'$ be the Laplacian matrix for the modified graph. Equation (4.8) can be re-written using matrix representation as:

$$\mathcal{L} = \frac{1-\eta}{2} ||\mathbf{X} - \mathbf{Y}||_F^2 + \frac{\eta}{2} \mathbf{X}^T \mathbf{L}' \mathbf{X} \tag{4.9}$$

We also consider a version of the loss function which uses the *normalized* Laplacian $\tilde{\mathbf{L}}' = \mathbf{D}'^{-1/2} \mathbf{L}' \mathbf{D}'^{-1/2}$ for regularization:

$$\tilde{\mathcal{L}} = \frac{1-\eta}{2} ||\mathbf{X} - \mathbf{Y}||_F^2 + \frac{\eta}{2} \mathbf{X}^T \tilde{\mathbf{L}}' \mathbf{X} \tag{4.10}$$

Using $\tilde{\mathbf{L}}'$ in place of $\mathbf{L}'$ performs as well if not better in practice; and moreover provides certain theoretical guarantees (see Proposition 4.3, and also [VLBB08]). Therefore, we will use Equation (4.10) as the loss function for our higher-order label spreading and refer to it as $\mathcal{L}_{\text{HOLS}}$. The closed-form solution for HOLS can now be obtained by differentiating $\mathcal{L}_{\text{HOLS}}$ with respect to $\mathbf{X}$ and setting it to zero. Thus, we derive:

$$\mathbf{X} = (1-\eta) \left(\mathbf{I} - \eta(\mathbf{I} - \tilde{\mathbf{L}}') \right)^{-1} \mathbf{Y} \tag{4.11}$$

Thus, using Equation (4.11), we are able to compute the optimal solution to HOLS, as long as the inverse of $\mathbf{I} - \eta(\mathbf{I} - \tilde{\mathbf{L}}')$ exists.

Due to the use of normalized Laplacian regularization, the following holds:

Proposition 4.2: Generalized Label Spreading

The proposed HOLS algorithm reduces to traditional label spreading [ZBL+03] for the base case of using only edge motifs, i.e., $\mathcal{K} = \{K_2\}$.

Proof. When $\mathcal{K} = \{K_2\}$, the modified adjacency $\mathbf{W}'$ is the same as the original adjacency $\mathbf{W}$. ■

This generalization grants HOLS its name.

In practice, matrix inversion is computationally intensive and tends to be numerically unstable. Hence, we propose to use an iterative approach to solve Equation (4.11) by first initializing $\mathbf{X}$ to an arbitrary value and then repeatedly applying the following update:

$$\mathbf{X} \leftarrow \eta(\mathbf{I} - \tilde{\mathbf{L}}')\mathbf{X} + (1 - \eta)\mathbf{Y} \tag{4.12}$$

Proposition 4.3 describes the theoretical properties of this approach.

Proposition 4.3: Convergence Guarantee for HOLS

The iterative update in Equation (4.12) always converges to the unique fixed point given in Equation (4.11) for any choice of initial $\mathbf{X}$.

Proof. From the theory of sparse linear iterative systems [Saa03], we know that Equation (4.12) converges if and only if $\rho(\eta(\mathbf{I} - \tilde{\mathbf{L}}')) < 1$ where $\rho(\cdot)$ is the spectral norm or the maximum absolute eigenvalue of the enclosed matrix. As the eigenvalues of normalized Laplacian $\tilde{\mathbf{L}}'$ are bounded in $[0, 2]$ [CG97], the eigenvalues of $\eta(\mathbf{I} - \tilde{\mathbf{L}}')$ lie within $[-\eta, \eta]$. Now, since $\eta \in (0, 1)$, we have $\rho(\eta(\mathbf{I} - \tilde{\mathbf{L}}')) \leq \eta < 1$. Thus, Equation (4.12) always converges. At convergence, the following holds: $\mathbf{X} = \eta(\mathbf{I} - \tilde{\mathbf{L}}')\mathbf{X} + (1 - \eta)\mathbf{Y}$. This leads to the fixed point from Equation (4.11), which is independent of initialization, as desired. ■

The overall algorithm of HOLS is summarized in Algorithm 1. For each motif $\kappa \in \mathcal{K}$, construct its κ-participation matrix by an exhaustive enumeration of all its occurrences. Note that the enumerated occurrences are processed one by one on the fly to update the participation matrix and discarded (no need for storage). Moreover, the enumeration for different motifs can be done in parallel. The participation matrices are combined into a single modified graph adjacency $\mathbf{W}'$; applying the iterative updates from Equation (4.12) finally results in labels for the unlabeled vertices. In practice, the iterative updates are applied until entries in $\mathbf{X}$ do not change up to a precision ϵ or until a maximum number of iterations T is reached.

Remark: It can be shown that higher-order label spreading on a graph $\mathcal{G}$ is equivalent to label spreading on a *hypergraph* $\mathcal{H}$ (recall that a hypergraph is a generalization of a graph in which a hyperedge can join any number of vertices.) constructed as follows: (i) Vertices in the hypergraph $\mathcal{H}$ are vertices in $\mathcal{G}$. (ii) For each instance of a motif of interest (e.g. edge, triangle) in $\mathcal{G}$, add to $\mathcal{H}$ an hyperedge connecting all participating vertices with weight equal to the importance of this motif. This observation connects the proposed HOLS algorithm to existing spectral semi-supervised classification techniques on hypergraphs [ZHS06]. However, instead of materializing such hyperedges–which requires prohibitively expensive space complexity even when reasonably sized motifs are used–HOLS provides a way to conduct higher-order label spreading in a space-efficient manner (see Proposition 4.4). In addition, this way of viewing hyperedges as representing motifs also paves the way to use deep hypergraph semi-supervised learning approaches [YNY+19, FYZ+19] to exploit higher-order network structures.

4.4.3 Time and Space Complexity

When only cliques are used as motifs $\mathcal{K}$ for semi-supervised learning, the following space and time complexity bounds hold:

Proposition 4.4: Space Complexity of HOLS

The space complexity of HOLS for a graph with N vertices, M edges and C classes is $\mathcal{O}(M + NC)$ independent of motif size and number of motifs used, provided all motifs are cliques.

The proof is sketched as follows. $\mathcal{O}(NC)$ is needed to store $\mathbf{X}$ matrix with the inferred labels. Note that two vertices participate in a clique only if they share an edge. Thus, the modified adjacency $\mathbf{W}'$ is at least as sparse as $\mathbf{W}$, having at most $\mathcal{O}(M)$ non-zero entries. Moreover, $\mathbf{W}'$ can be constructed by enumerating occurrences of all motifs in $\mathcal{K}$ and updating the relevant entries of $\mathbf{W}'$ on the fly.

Proposition 4.5: Time Complexity of HOLS

The time complexity of HOLS over a graph with M edges, C classes and a degeneracy (core number) of $k_{\max}$ using $\mathcal{K} = \{K_2, \ldots, K_n\}$ is given by $\mathcal{O}\left(M \sum_{k=2}^{n} k \left(\frac{k_{\max}}{2}\right)^{k-2}\right)$ for the construction of K_k-participation matrices plus $\mathcal{O}(MC)$ per iterative update using Equation (4.12).

The proof follows from Theorem 5.7 of [DBS18].

Practical Consideration: Despite the exponential complexity in k, we are able to enumerate cliques quickly using the sequential kClist algorithm [DBS18]. For example, our largest Pokec dataset has 21M edges, 32M triangles, 43M 4-cliques and 53M 5-cliques; and the enumeration of each took a maximum of 20 seconds on a stock laptop. Thus, HOLS remains fast

Method	EuEmail	PolBlogs	Cora	Pokec
Label Propagation (LP) [ZGL03]	0.2905	0.5814	0.2765	0.1994
Label Spreading (LS) [ZBL+03]	0.5228	0.9361	0.4921	0.5514
node2vec+TSVM [GL16, Joa99]	0.4563	**0.9481**	0.4233	T.L.E.
Graph Convolution Networks (GCN) [KW17]	0.5251	0.9470	0.4673	0.5290
HOLS (proposed)	**0.5473***	0.9476	**0.4953***	**0.5593***

Table 4.3: Accuracy of all methods averaged over five runs. In each column, the best value is bold and underlined, and the second best is underlined. Asterisk (*) denotes statistically significant difference ($p < 0.05$) compared to the closest second baseline. The results show that HOLS outperforms the state-of-the-art methods in terms of accuracy with a statistically significant margin in three out of four graphs.

Method	EuEmail	PolBlogs	Cora	Pokec
Label Propagation (LP) [ZGL03]	0.11	0.070	2.1	1320
Label Spreading (LS) [ZBL+03]	**0.040***	**0.036***	**0.21***	**93***
node2vec+TSVM [GL16, Joa99]	46	29	3060	>1 day
Graph Convolution Networks (GCN) [KW17]	1.8	1.3	6.4	2880
HOLS (proposed)	0.089	0.083	0.41	117

Table 4.4: Running time of all methods averaged over five runs. In each column, the best value is bold and underlined, and the second best is underlined. Asterisk (*) denotes statistically significant difference ($p < 0.05$) compared to the closest second baseline. The results show that HOLS has comparable running time to the standard label spreading algorithm and HOLS is at least 15 times faster than deep learning baselines.

and scalable when reasonably small cliques are used. Moreover, as we show in experiments, typically using triangles (3-cliques) in addition to edges suffices to achieve the best classification performance across a wide range of datasets.

4.5 Experiments

We empirically evaluate HOLS on the four diverse real-world networks: EuEmail (an e-mail network), PolBlogs (a blog hyperlink network), Cora (an article citation network), and Pokec (a friendship social network). Please refer to Section 4.3.3.1 for the network details.

4.5.1 Experimental Setup

We have implemented higher-order label spreading (HOLS) in MATLAB and have run the experiments on MacOS with 2.7 GHz Intel Core i5 processor and 16 GB main memory.

Baselines: We compare HOLS to the following baselines: *(1) Label Propagation (LP)* [ZGL03] which uses Laplacian regularization. *(2) Label Spreading (LS)* [ZBL+03] which uses normalized Laplacian regularization. *(3) node2vec+TSVM* which generates unsupervised vertex embeddings using node2vec [GL16] and learns decision boundaries in the embedding space using a transductive SVMs [Joa99]. *(4) Graph Convolutional Network (GCN)* [KW17] which is an end-to-end semi-supervised learner using neural networks. We implement LP and LS in MATLAB, and use open-sourced code for the rest.

Parameters: By default, we use a weight of $\eta = 0.5$ for supervised loss and $\mathcal{K} = \{K_2, K_3\}$ motifs (edges and triangles) for HOLS. The importance weight for triangles α_3 is tuned in $\{0.1, 0.2, \ldots, 0.9\}$ for each dataset and results are reported on the best performing value. We use the $\eta = 0.5$ for LS as well. LP, LS and HOLS are run until labels converge to a precision of ϵ or until T iterations are completed, whichever occurs sooner. We set $\epsilon = 10^{-6}$ and $T = 500$. We use the default hyperparameters for GCN, node2vec and TSVM. We supply 100, 20, 100 and 1000 labels for EuEmail, PolBlogs, Cora and Pokec datasets, where the vertices to label are chosen by stratified sampling based on class. These correspond to label fractions of 5%, 1.6%, 0.4% and 0.06% and on an average, 1, 10, 10 and 100 labeled vertices per class respectively.

Evaluation Metrics: Evaluating only over unlabeled vertices of degree at least one, we quantify success using: *(i) accuracy:* fraction of vertices correctly classified, *(ii) precision:* fraction of vertices classified as class c which actually belong to class c, *(iii) recall:* fraction of vertices which belong to a class c that are correctly classified, *(iv) F1-score:* harmonic mean of per-class precision and recall. For all these metrics, a higher value is more desirable.

4.5.2 Q1. Accuracy Comparison of HOLS

The accuracy of HOLS and all the baselines is summarized in Table 4.3. All values are averaged over five runs, each run differing in the set of vertices for which labels are supplied. The values for node2vec+TSVM on Pokec dataset are missing as the method did not terminate within 24 hours ('T.L.E.').

First, we observe in Table 4.3 that HOLS consistently leads to (statistically significant) improvements over LS, showing that using higher-order structures for label propagation helps.

We show that HOLS outperforms *all* baselines in three out of four datasets. The improvements over the best baseline are statistically significant ($p = 0.05$) according to a two-sided micro-sign test [YL99] in at least three out of five runs. Interestingly, for the smaller datasets (EuEmail and PolBlogs), while GCN outperforms LS, GCN loses to HOLS when triangles are used. node2vec+TSVM performs slightly better than HOLS on PolBlogs, however, the increase over HOLS is not statistically significant. For the larger datasets with extremely low labeling fraction ($< 0.5\%$ labeled vertices), HOLS performs the best and LS follows closely.

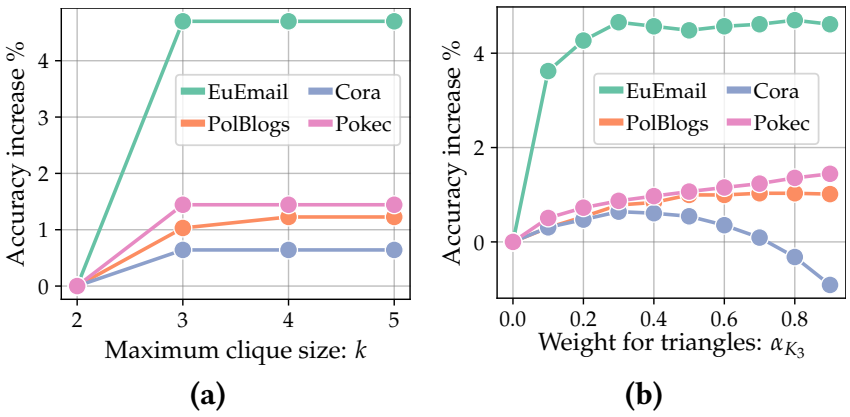


Figure 4.5: (a) Higher-order structures in HOLS improve the performance of vertex labeling. The use of 3-cliques gives the most boost. Larger cliques larger give minor improvements. (b) 3-cliques improve the performance for a large range of motif weights α across all datsets.

4.5.3 Q2. Variation of Accuracy with Higher-Order Structures

In this experiment, we study the effect of adding higher-order structures to HOLS. The relevant figures are displayed in Figure 4.5; all reported accuracies are averaged across five runs.

Fixing the motif set as $\mathcal{K} = \{K_2, K_3, \ldots, K_k\}$, we varying $k = 2, 3, 4, 5$ to study the marginal benefit of including higher-order cliques in graph SSL. The motif weights are tuned in $\alpha_j \in \{0, 0.1, \ldots,$
$0.9\}$, ensuring that edges are given a weight $\alpha_2 \geq 0.1$ for a connected graph, and further, all motif weights sum to 1.

The best performing motif weights were used to generate Figure 4.5(a), which plots the relative improvement in accuracy over LS that uses edges only. First, we observe that label propagation via higher-order structures strictly outperforms label propagation via edges. Compared to edges, the gain is the most when using 3-cliques. Second, we note that the use of further higher-order cliques has little marginal performance gain. This can be explained by the observation that the value of label consistency tapers off for higher-order cliques (see Figure 7.1(b)), suggesting that they add little value to graph SSL.

Next, we investigate the effect of the importance given to triangles in comparison to edges in HOLS. Fixing the motif set to $\mathcal{K} = \{K_2, K_3\}$, we vary the triangle weight α_3 of HOLS in $\{0, 0.1, \ldots, 0.9\}$. Figure 4.5(b) shows that the accuracy gain of HOLS over LS increases with an increase in triangle weight for most graphs. The only exception is Cora, where the accuracy gain grows until $\alpha_3 = 0.4$ before decreasing and eventually turning negative. Overall, triangles consistently help over a large range of motif weights.

4.5.4 Q3. Runtime Performance of HOLS

The running time of HOLS and all the baselines is summarized in Table 4.4. Notably, we see that HOLS runs in less than 2 minutes for graphs with over 21 million edges (the POKEC graph), showing the real-world practical scalability of thep proposed method.

We observe that LS is the fastest of all methods and HOLS comes a close second for three out of four datasets. The small difference in running time predominantly stems from the construction of triangle participation matrix. Notably, HOLS is over $15\times$ faster than the recent GCN and node2vec+TSVM baselines, *for comparable and often better values of accuracy.*

4.5.5 Q4. Variation of Accuracy with Label Consistency

Figure 4.6 and Figure 4.1(b) compare the dependency of classification performance (accuracy and F1 score) on the label consistency of triangles and edges. We observe these at the level of classes within a dataset and across datasets.

Variation Across Classes: Figure 4.6 plots the F1 score of each class using HOLS versus their average label consistency (LC) with respect to edges and triangles, i.e., $(\bar{\lambda}_c(K_2) + \bar{\lambda}_c(K_3))/2$ where the class label consistency $\bar{\lambda}_c(\kappa)$ of a class c with respect to a motif κ is the average vertex label consistency of all vertices which belong to the class. This plot shows a strong positive correlation between the two—the greater the label consistency for a class, the easier it is to classify the vertices belonging to that class. The strength of linear relationship between F1 and LC has high correlation coefficient scores between 0.83 and 1.0 across all datasets.

Variation Across Datasets: Figure 4.5(a) plots all four graphs based on the prevalence of triangles in the graph and the relative improvement in label consistency of triangles compared to edges. The prevalence of triangles is measured by the average clustering coefficient [WS98]—higher implies more triangles. The bubble sizes indicate the relative improvement in accuracy of HOLS (using triangles and edges) over LS (using edges only). We make an interesting observation that the large improvement from using triangles occurs when (i) triangles are more label consistent than edges, and (ii) there are many triangles in the graph. Thus, the EUEMAIL graph has the highest clustering coefficient among the datasets and the high LC results in most benefit from using triangles. The POLBLOGS and CORA graphs have lower values of clustering coefficient and LC scores; thus the improvements are lower. In POKEC, the lower clustering coefficient is compensated by the larger value label consistency, resulting in similar performance improvement to CORA.

4.5.6 Q5. Variation of Accuracy with Vertex Label Consistency

In this experiment, we investigate the properties of the vertices that are helped (and hurt) by incorporating higher-order structures. Here, a vertex is 'helped' if HOLS with higher-order structure correctly labels the vertex, while LS with only edges mislabels the vertex. Conversely, a vertex is 'hurt' if HOLS mislabels it while LS labels it correctly. In this analysis, we do not consider the vertices that are correctly or incorrectly classified by both methods.

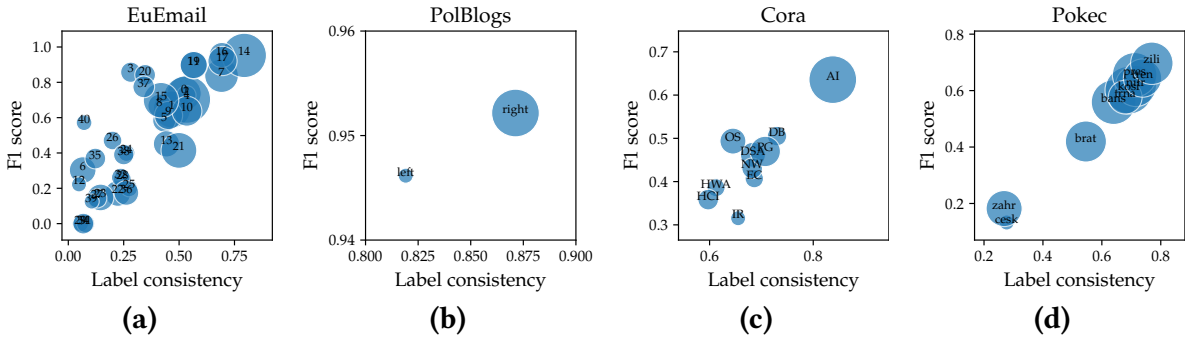


Figure 4.6: Across all datasets, the F1 scores of classes for vertex classification are strongly positively correlated to the class label consistencies (correlation between 0.82 to 1.00). Bubble size indicates the number of vertices in the class.

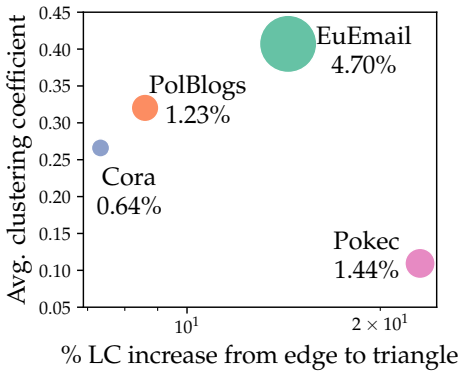
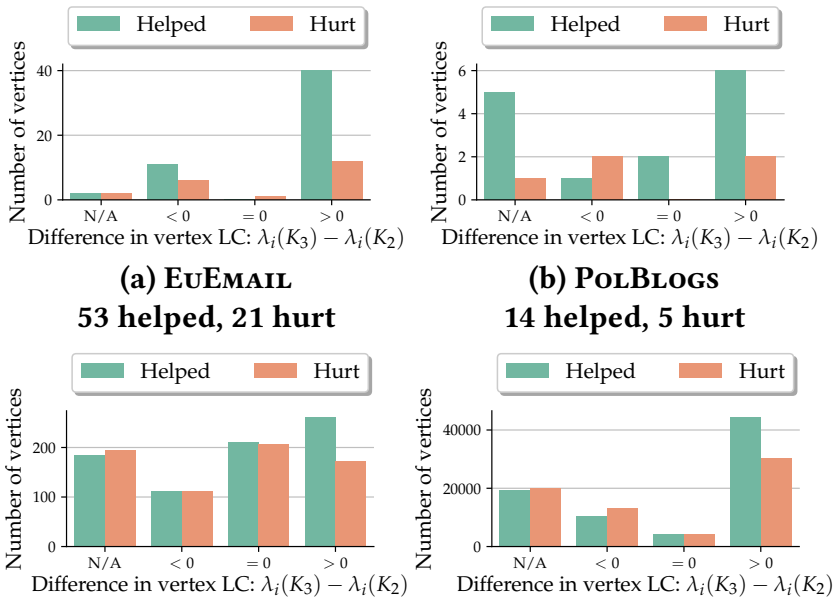


Figure 4.7: HOLS leads to larger accuracy gains over LS when the average clustering coefficient is high due to the presence of many triangles and triangles are far more label-consistent than edges.

We find a striking relation between the vertex label consistency scores and its helped/hurt probability. For each vertex, we calculate its relative vertex label consistency $\Delta\lambda_i(K_3)$, which is the difference between the triangle minus edge vertex label consistency for the vertex. The vertices are categorized into four bins using $\Delta\lambda_i(K_3)$: (a) 'N/A', if the vertex is not a part of any triangle, (b) < 0, where triangles have lower LC, (c) $= 0$ where the vertex has equal LC for both triangles and edges, and finally, (d) > 0 where triangles have a higher LC compared to edge LC. Figure 4.8 presents aggregate statistics for all the graphs.

First, we note that the total number of vertices helped by HOLS is higher than the number of vertices hurt. Next, HOLS largely corrects the mistakes made by LS for vertices with $\Delta\lambda_i(K_3) > 0$. The overall benefit of HOLS stems from the fact that most vertices have $\Delta\lambda_i(K_3) > 0$. In some datasets, the vertices $\Delta\lambda_i(K_3) < 0$ are hurt in HOLS as they have lower vertex label consistency in triangles, which lead to misclassification. Notably, many vertices that do not participate in a triangle, i.e., $\Delta\lambda_i(K_3) = N/A$, are helped by HOLS because of correct classification of its neighbors.



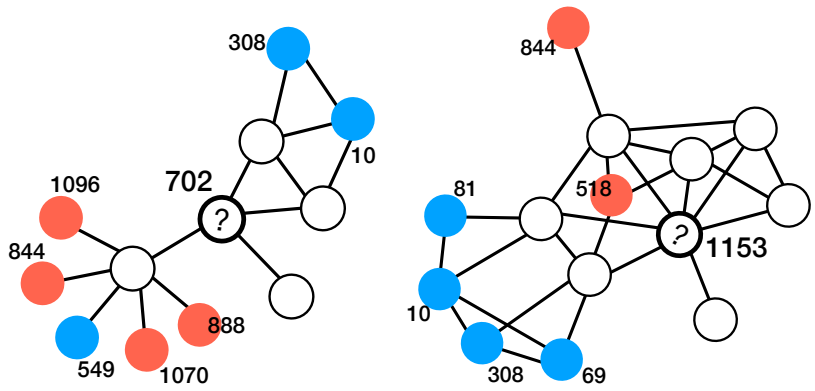
(a) EuEmail
53 helped, 21 hurt

(b) PolBlogs
14 helped, 5 hurt

(c) Cora
770 helped, 685 hurt

(d) Pokec
79K helped, 68K hurt

Figure 4.8: Vertices having a positive difference of local LC wrt. triangles and edges are generally helped by HOLS, while those having a negative difference tend to be hurt. Vertices having zero difference or no incident triangles show mixed trends.



(a) v702 **(b) v1153**

Figure 4.9: Case studies from PolBlogs dataset showing extended ego-networks of vertices (a) v702 and (b) v1153 which are both incorrectly classified by LS but correctly classified when triangles are taken into account using HOLS.

Case Studies: Here we look at real examples from the PolBlogs dataset to dig deep into when HOLS improves over LS. Figure 4.9 shows our analysis. Question mark denotes the central vertices v702 and v1153 of interest with ground truth labels 'blue' and 'red' respectively. The direct neighbors of the both v702 and v1153 are unlabeled and a few second hop neighbors are labeled with one of two labels: 'blue' or 'red'.

In both cases, LS correctly labels all vertices except the central vertex. This is because LS utilizes only edge-level information – v702 has more second-hop red neighbors than blue whereas v1153 has more second-hop blue neighbors than red. On the other hand, HOLS leverages the fact that v702 participates in one triangle with (inferred) blue vertices and that v1153 participates in several triangles with (inferred) red vertices and thus produces the correct labels.

In summary, these case studies provide concrete examples from real-world data where label spreading using edges alone leads to over-prediction of the locally popular class (e.g. blue for v1153 in Figure 4.9 (b)). In such cases, the use of higher-order network information has the benefit of correcting the over-prediction problem and leading to the correct labels *even when they are not locally popular.*

4.6 Conclusion

In this chapter, we created a framework to incorporate the signal present in higher-order structures in a graph. This work paves the way towards systematic study of the effect of higher-order structures in graph semi-supervised learning. Our proposed higher-order label consistency metric revealed that real-world graphs exhibit homophily in higher-order structures. We created a higher-order label spreading algorithm and experimentally showed that incorporating cliques aids in vertex labeling tasks across four large real-world datasets.

This work opens the avenue for several exciting future research directions. We outline four directions below.

First, in Figure 4.8, we found that while most vertices are correctly labeled by using higher-order structures, some vertices are mislabeled compared to when higher-ordered structures are not used. A promising research direction is to learn a vertex-level decision function that selects the label generated by edges or the higher-order structures. This will help in creating methods that can leverage the best of all worlds in vertex labeling tasks.

Second, having seen the advantage of higher-order structure in real-world graphs, it becomes fundamental to understand the benefits in the theoretical networks. For instance, what is the expected lift in using higher-order structures in Erdos-Renyi graphs? In stochastic block models, how does the behavior vary with intra-group and inter-group edge probabilities?

Third, having seen the benefits in undirected, unweighted graphs for vertex labeling, it is important to explore the benefits of higher-order structures in other types of graphs, such as heterogeneous and dynamic graphs, and in other application tasks, such as graph classification. This will be crucial to create the next-generation of robust and efficient algorithms.

Finally, it is promising to bring together recent local counting based techniques for efficient clique counts (e.g. [JS20]) with label spreading to improve the computational complexity of higher-order label spreading approaches.

Part III

Dynamic Graphs

Overview: Dynamic Graphs

Given time-evolving graph (explicit or implicit), how can we detect anomalies or events in near real-time immediately after they have occurred, or perhaps even early warn when their occurrence is anticipated?

In this part, we consider mining anomalies from data in which the connectivity structure evolves over time. In many settings, especially those relating to security and health care, the value of a newfound or anticipated anomaly lies in the moment, and not later. Thus, detection in near real-time, and early warning becomes critical. Our algorithms can detect anomalous graph footprints such as sudden appearance or disappearance of dense subgraphs (SpotLight, Chapter 5) and bridge edges (SedanSpot, Chapter 6) in near real-time, by only storing a small synopsis of the graph seen so far and requiring no supervision. We also develop SmokeAlarm (Chapter 7) to infer state-transition graph from time series data in an online manner and use that to early warn against user-labeled anomalies such as adverse medical conditions.

Chapter 5

SpotLight: Anomalous Dense-Subgraph Detection

Chapter based on work that appeared at KDD 2018 [EFGM18] [PDF].

How do we spot interesting events from e-mail or transportation logs? How can we detect port scan or denial of service attacks from IP-IP communication data? In general, given a sequence of weighted, directed or bipartite graphs, each summarizing a snapshot of activity in a time window, how can we spot anomalous graphs containing the sudden appearance or disappearance of large dense subgraphs (e.g., near bicliques) in near real-time using sublinear memory? To this end, we propose a randomized sketching-based approach called SpotLight, which guarantees that an anomalous graph is mapped 'far' away from 'normal' instances in the sketch space with high probability for appropriate choice of parameters. Extensive experiments on real-world datasets show that SpotLight (a) improves accuracy by at least 8.4% compared to prior approaches, (b) is fast and can process millions of edges within a few minutes, (c) scales linearly with the number of edges and sketching dimensions and (d) leads to interesting discoveries in practice.

5.1 Introduction

Time-evolving (or dynamic) weighted directed/bipartite graphs, where both nodes and edges are continuously added over time, are artifacts generated in many real-world contexts. Examples include transportation logs (w cabs travel from location s to location d), network communication logs (w packets sent by IP address s to IP address d), instant-messaging, phone call, e-mail logs (w messages/calls/emails from user s to user d), collaborative editing logs (w edits made by user s to page d) and so on.

We consider the problem of near real-time anomaly detection in such settings. Due to the fluid nature of what is considered 'normal', prior works typically focus on detecting specific anomalous changes to the graph, e.g., bridge edges [SD14, RHSS16], hotspot nodes [YAMW13], changes to community structure [STF06, SFPY07], graph metrics [HEF+10, FNG14], etc. In this

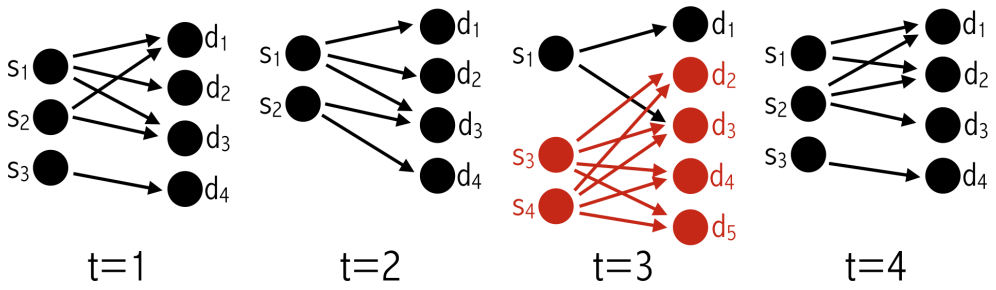


Figure 5.1: Sudden appearance of a dense subgraph at t=3.

work, we focus on detecting anomalies involving the *sudden appearance or disappearance of a large dense directed subgraphs* (near bicliques), which is useful in numerous applications: detecting attacks (port scan, denial of service) in network communication logs, interesting/fraudulent behavior creating spikes of activity in user-user communication logs (scammers who operate fast and in bulk), important events (holidays, large delays) creating abnormal traffic in/out flow to certain locations, etc. We are able to discover several of the above phenomena in real-world data (e.g., Figure 5.12).

We highlight two important aspects of the above definition. The (dis)appearance of a large dense subgraph is anomalous only if it is *sudden*, i.e., it has not been observed before or is not part of a slow evolution (e.g., steadily growing communities). Similarly, the sudden (dis)appearance of a large number of edges is anomalous only if the edges form a *dense* subgraph (the so-called *lockstep* behavior indicating fraud [BXG+13]). Figure 5.1 illustrates this. In the evolution of a bipartite graph, e.g., user edits page, an anomalous dense directed subgraph appears at t=3, indicating a possible edit-war between users s_3 and s_4 w.r.t. pages d_2, d_3, d_4, d_5. In contrast, the appearance of subgraph $\{s_1, s_2\} \rightarrow \{d_1, d_2, d_3\}$ at t=4 is not anomalous, since it has already been (partially) observed at t=1, 2.

The temporal aspect, i.e., near real-time detection, is crucial for our problem. The value of a newfound surge of ridership requests or network attack lies in the moment, not one week later. Moreover, given that nodes and edges are added over time, we seek solutions that can operate in sublinear memory, without storing a counter for each edge/node. The problem we set out to solve is:

Informal Problem 5.1

Given a stream of weighted, directed/ bipartite graphs, $\{\mathcal{G}_1, \mathcal{G}_2, \ldots\}$, **detect in near real-time** whether $\mathcal{G}_t$ contains a *sudden* (dis)appearance of a *large dense* directed subgraph **using sublinear memory**.

The technical challenge in detecting the sudden (dis)appearance of a large dense directed subgraph is computational. New edges and nodes are continuously arriving and we have limited time and space to process the changes. The approach that we take is to design a short summary

or sketch of the graph that both reveals newly found anomalies and can be quickly updated and maintained on a high-speed moving data stream.

Concretely, our contributions are: **(a) Algorithm (Section 5.4):** We propose SpotLight, a simple randomized sketching-based approach to solve Problem 5.1. **(b) Guarantees (Section 5.5):** We prove that SpotLight is *focus-aware* in expectation, i.e., flags focused addition or deletion of edges as more anomalous than dispersed changes of the same magnitude (Theorem 5.1) and maps anomalous graphs 'far' away from 'normal' instances in the sketch space *with high probability* for appropriate choice of parameters (Theorem 5.2). **(c) Effectiveness (Section 5.6):** Extensive experiments on real-world data show that SpotLight outperforms prior approaches in terms of precision and recall, is fast and scalable and leads to interesting discoveries.

5.2 Related Work

Anomaly Detection in Static Graphs: is well-studied (for survey, see [ATK15]). Unsupervised methods rely on node-level features [AMF10], spectral decomposition [PSS+10], finding dense subgraphs signifying fraud [BXG+13, HSB+16b], etc. In the presence of limited supervision, belief propagation is known to work well [EGF+17b].

Anomaly Detection in Time-Evolving Graphs: can be reviewed under the following categories (for survey, see [RSK+15]).

(i) Approaches comparing consecutive snapshots [KSV+16, SD14]: The traditional approach is to compare adjacent graphs $(\mathcal{G}_t, \mathcal{G}_{t+1})$ via a similarity function based on, e.g., belief propagation [KSV+16], random walks [SD14], etc., They do not consider evolutionary/periodic trends.

(ii) Dense subgraph detection based approaches [JBC+15, SHF16]: These techniques model dynamic graphs as node×node×time tensors and aim to approximately identify the top-k densest subblocks, e.g., persistent dense subgraphs. In contrast, we aim to detect only the *sudden* appearance of dense subgraphs in *near real-time*.

(iii) Graph decomposition/partitioning based approaches [STF06, SFPY07]: These methods store a summary of the graph structure based on tensor decomposition [STF06] or minimum description language [SFPY07] and identify change points as anomalies. Their primary focus is on the computationally hard problem of graph modeling.

(iv) Anomalous edge detection approaches [AZY11, RHSS16, MMA16]: The first two methods score the likelihood of an edge based on the community structure [AZY11], prior occurrence preferential attachment and homophily information [RHSS16]. By scoring edges independent of each other, these methods miss complex structural (e.g., dense subgraph) anomalies. They also cannot detect edges which are expected but do not occur. [MMA16] is closely related, but is applicable when only multiple heterogeneous graphs are evolving simultaneously.

(v) Others: [HEF+10] offers a suite graph metrics to perform anomaly detection at multiple temporal and spatial granularities. [IK04] detects anomalous nodes using their activity vectors from principle component analysis (PCA). [YAMW13] also uses PCA, but to detect anomalous nodes (hotspots). [FNG14] proposes density-consistent statistics to compare graphs having significantly different edge counts.

Property	[KSV+16], [SD14]	[STF06],[SFPY07]	[YAMW13]	[IK04]	[HEF+10]	[RHSS16]	This work
Directed/bipartite graphs		✓		?	?		✓
Weighted/multi edges	✓	✓	✓	✓	✓	✓	✓
Sublinear memory						✓	✓
Theoretical guarantee							✓

Table 5.1: Qualitative comparison with prior work on anomaly detection in streaming graphs.

A qualitative comparison is provided in Table 5.1.

Randomized Graph Streaming Algorithms: for testing connectivity and bipartiteness, constructing sparsifiers and spanners, approximating the densest subgraph etc. in the semi-streaming model (in $\mathcal{O}(n \text{ polylog } n)$ space where n is the number of nodes) are popular within the theory community [McG14, MTVV15]. However, they do not address graph *anomaly detection* using *sublinear memory*.

Randomized Algorithms for Anomaly Detection: Perhaps, the first known randomized anomaly detector is Isolation Forests [LTZ08] for static multi-dimensional data. Due to its empirical success, randomized algorithms for streaming multi-dimensional data streams are recently gaining traction [WZF+14, Pev16, GMRS16]. In this work, we investigate a randomized algorithm for the streaming *graph* setting.

5.3 Preliminaries

In this section, we introduce our streaming model and formalize how to detect the sudden (dis)appearance of large dense subgraphs.

Streaming Model: Let $\mathfrak{G} = \{\mathcal{G}_t\}_{t=1}^{\infty}$ be a graph stream. Each graph $\mathcal{G}_t$ is a tuple $(\mathcal{S}_t, \mathcal{D}_t, \mathcal{E}_t)$ where $\mathcal{S}_t$ and $\mathcal{D}_t$ are the possibly time-evolving sets of source and destination nodes respectively and each edge (s, d, w) in the edge set $\mathcal{E}_t$ originates from a *source* $s \in \mathcal{S}_t$, ends at a *destination* $d \in \mathcal{D}_t$ and carries a *weight* $w \in \mathbb{R}^+$ (w=0 is equivalent to the absence of an edge). We assume each node (source or destination) has a unique identifier that is fixed over time, i.e., the *node-correspondence* across graphs is known. Let $\mathbf{A}_t = [A_{t,sd}]$ be the adjacency of $\mathcal{G}_t$ where each $A_{t,sd}$ denotes the sum of weight of edges connecting a source s to a destination d in graph $\mathcal{G}_t$.

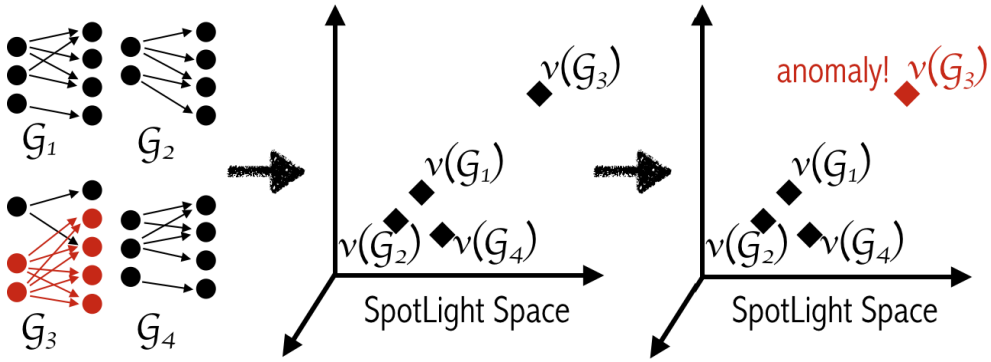


Figure 5.2: Overview of SpotLight

While there are other ways of aggregating weights, this is the most natural in the applications we consider (see Section 5.1) .

The above model allows us to represent a flexible range of graphs: (i) weighted or unweighted (by letting $A_{t,sd} = 1 \; \forall \; s, d$), (ii) bipartite or unipartite (by allowing $\mathcal{S}_t$ and $\mathcal{D}_t$ to overlap) and (iii) directed or undirected (by constraining $A_{t,sd} = A_{t,ds}$) when $s \neq d$).

Problem Description: Given a graph $\mathcal{G}$ with adjacency $\mathbf{A}$, let $\mathcal{G}(\mathcal{S}', \mathcal{D}')$ denote the directed subgraph induced by the source set $\mathcal{S}'$ and the destination set $\mathcal{D}'$. Its density $\rho(\mathcal{G}(\mathcal{S}', \mathcal{D}'))$ can be defined in several ways, e.g., $\sum_{s \in \mathcal{S}', d \in \mathcal{D}'} A_{sd} / \left|\mathcal{S}'\right| \left|\mathcal{D}'\right|$ – the higher the total weight of edges in it, the greater its density [Die12].

In a nutshell, a graph $\mathcal{G}_t$ is said to be anomalous – i.e., contain a sudden appearance or disappearance of a dense directed subgraph – if there is a *large* directed subgraph which shows a *significant* change in density compared to the past graphs $\{\mathcal{G}_{t-1}, \mathcal{G}_{t-2}, \ldots\}$. For example, in Figure 5.1, letting $\mathcal{S}' = \{s_3, s_4\}$ and $\mathcal{D}' = \{d_2, d_3, d_4, d_5\}$, the subgraph $\mathcal{G}_3(\mathcal{S}', \mathcal{D}')$ has high density (=1) but $\mathcal{G}_1(\mathcal{S}', \mathcal{D}')$ and $\mathcal{G}_2(\mathcal{S}', \mathcal{D}')$ have low densities, 0.125 and 0 respectively. Hence $\mathcal{G}_3$ is an anomaly. The next section presents the proposed method to identify such anomalies.

5.4 Proposed Method

The proposed method, called SpotLight, works in two main steps as shown in Algorithm 2. First, it extracts a K-dimensional (we show how to choose K in Section 5.5) SpotLight sketch $\mathbf{v}(\mathcal{G})$ for every $\mathcal{G}$, such that graphs containing the sudden (dis)appearance of large dense subgraphs are 'far' from 'normal' graphs in the sketch space (line 4). Second, it exploits the distance gap in the sketch space to detect graphs yielding anomalous sketches as anomalous graphs (line 5). A schematic is given in Figure 5.2. We next elaborate on these two steps in greater detail.

5.4.1 SpotLight Graph Sketching

A natural way to sketch a graph is by enumerating the total edge weight of each directed subgraph $\mathcal{G}(\mathcal{S}', \mathcal{D}')$ for sufficiently large source and destination sets $\mathcal{S}', \mathcal{D}'$. However, this sketch

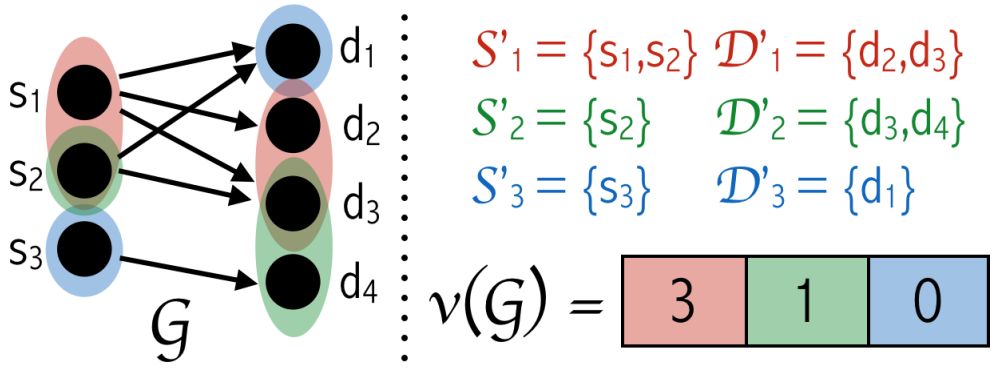


Figure 5.3: A $(K{=}3, p{=}0.5, q{=}0.33)$-SpotLight sketch $\mathbf{v}(\mathcal{G})$ of a graph $\mathcal{G}$ with unit-weight edges. Each sketch dimension $v_k(\mathcal{G})$ is the total weight of edges going from a random set of sources $\mathcal{S}'_k$ and to a random set of destinations $\mathcal{D}'_k$.

has exponential number of dimensions and is infeasible to compute or store. Hence, we propose to compose a sketch containing total edge weights of K specific directed subgraphs (called *query subgraphs* henceforth) chosen independently and uniformly at random, according to node sampling probabilities, p for sources and q for destinations. This leads to (K, p, q)-SpotLight graph sketching.

Conceptually, SpotLight sketching first chooses K query subgraphs $\{(\mathcal{S}'_k, \mathcal{D}'_k)\}_{k=1}^{K}$ by sampling each source (or destination) into each $\mathcal{S}'_k$ (resp. $\mathcal{D}'_k$) with probability p (resp. q). This choice is made only once per source or destination (the first time it is seen) and is fixed throughout the graph stream. Next, for every graph $\mathcal{G}$, its sketch $\mathbf{v}(\mathcal{G}) \in \mathbb{R}^K$ is computed as $v_k(\mathcal{G}) = \sum_{s\in\mathcal{S}'_k, d\in\mathcal{D}'_k} A_{sd} = \textit{total\_edge\_weight}(\mathcal{G}(\mathcal{S}'_k, \mathcal{D}'_k))$. For example, in Figure 5.3 showing a graph $\mathcal{G}$ with unit-weight edges, there are three edges belonging to the first query subgraph (red), one to the second (green) and none to the third (blue). Hence, its sketch is $\mathbf{v}(\mathcal{G}) = (3, 1, 0)$.

An efficient implementation of SpotLight sketching using hashing is given in Algorithm 2. The hash functions ensure that the node to query subgraph mapping remains fixed over time without explicitly storing it. The choice of the first hash bucket in line 13 is arbitrary; one can pick any value within the suitable range. Observe how this algorithm is able to seamlessly process old and new nodes alike.

SpotLight sketching can be thought of in two alternative ways. First, it can be regarded as a memory-limited and non-deterministic generalization of two common used graph features – nodal degree ($K{=}|\mathcal{S}|, p{=}1/|\mathcal{S}|, q{=}1$ or $K{=}|\mathcal{D}|, p{=}1, q{=}1/|\mathcal{D}|$) and total edge weight ($K{=}p{=}q{=}1$). Second, and more interestingly, each sketch dimension can be considered as a spotlight which illuminates and allows for monitoring a region of the graph (i.e., its query subgraph). The central idea is that *the (dis)appearance of a large and dense subgraph would be brought to light by at least one of these spotlights, provided there are enough of them and each one is fine-grained, illuminating a small enough region of the graph.* In Section 5.5, we prove high probability guarantees of exactly this nature.

Algorithm 2 SpotLight graph stream anomaly detection

Input: a stream $\mathfrak{G}$ of weighted directed/bipartite graphs
Parameters: sketch dimensionality K, source sampling probability p, destination sampling probability q
Output: a stream of anomaly scores

1: **procedure** SpotLight($\mathfrak{G}, K, p, q$)
2: Initialize(K, p, q)
3: **for** graph $\mathcal{G} \in \mathfrak{G}$ **do**
4: $\mathbf{v} \leftarrow$ Sketch($\mathcal{G}$)
5: **yield** AnomalyScore($\mathbf{v}$)
6: **procedure** Initialize(K, p, q)
7: **for** $k = 1, \ldots, K$ **do**
8: Pick source hash $h_k : \mathcal{S} \rightarrow \{1, \ldots, \lfloor 1/p \rfloor\}$ and destination hash $h'_k : \mathcal{D} \rightarrow \{1, \ldots, \lfloor 1/q \rfloor\}$ independently at random.
9: **procedure** Sketch($\mathcal{G}$)
10: $\mathbf{v} \leftarrow \mathbf{0}_K$
11: **for** edge $e = (s, d, w)$ in graph $\mathcal{G}$ **do**
12: **for** $k = 1, \ldots, K$ **do**
13: **if** $h_k(s) == 1$ and $h'_k(d) == 1$ **then**
14: $v_k \leftarrow v_k + w$
15: **return** $\mathbf{v}$

5.4.2 Anomaly Detection in the SpotLight Space

Exploiting the distance gap between the anomalous graphs containing the sudden (dis)appearance of large dense subgraphs and 'normal' instances in the SpotLight (sketch) space, we may now employ any off-the-shelf data stream anomaly detector (e.g., [GMRS16, Pev16, WZF+14]) to carry out AnomalyScore procedure call (line 5 of Algorithm 2). These techniques require sublinear memory and output an anomaly score for every data point (i.e., SpotLight graph sketch) in the stream.

5.5 Theoretical Analysis

This section presents the distance guarantees offered by SpotLight sketch space and also analysis of running time and memory.

5.5.1 Guarantees for SpotLight Sketches

How do we theoretically analyze the distance between graphs in the SpotLight space, even though the sketching algorithm is randomized? What properties should this distance function obey? How do we choose the sketching parameters so that anomalous graphs lie 'far' from

'normal' instances with high probability in the SpotLight space? These are the questions we set out to answer.

In the rest of this section, $\mathcal{G}$ is always an arbitrary weighted directed/bipartite graph on N_s sources and N_s destinations. Adding unit-weight edges to $\mathcal{G}$ increments corresponding edge weights by one, even if these edges already existed. $\mathbf{v}(\cdot)$ represents the (K, p, q)-SpotLight sketch. For simplicity, we let $N_s{=}N_d{=}N$ and $p{=}q$. Also, without loss of generality, we consider only the appearance of dense subgraphs (disappearance can be argued in a similar way).

We begin by defining SL-distance (SL for SpotLight) between graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ in the SpotLight space as a *deterministic* function of $\mathcal{G}_1, \mathcal{G}_2$ and the sketching parameters K, p, q.

Definition 5.1: SL-Distance

The SL-distance between graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the expected squared Euclidean distance between their SpotLight sketches, i.e., $\bar{d}(\mathcal{G}_1, \mathcal{G}_2) = \mathbb{E}\left[||\mathbf{v}(\mathcal{G}_1) - \mathbf{v}(\mathcal{G}_2)||_2^2\right]$, where the expectation is taken over the random coin tosses of the sketching algorithm[a].

[a] $\bar{d}(\cdot, \cdot)$ is not a metric, but it obeys a relaxed triangle inequality.

We devote the rest of this section to show (i) that SL-distance is *focus-aware*, a desirable property for anomaly detection and (ii) how to set sketching parameters so that 'anomalous' graphs lie far from 'normal' ones according to SL-distance. All proofs are given in the appendix.

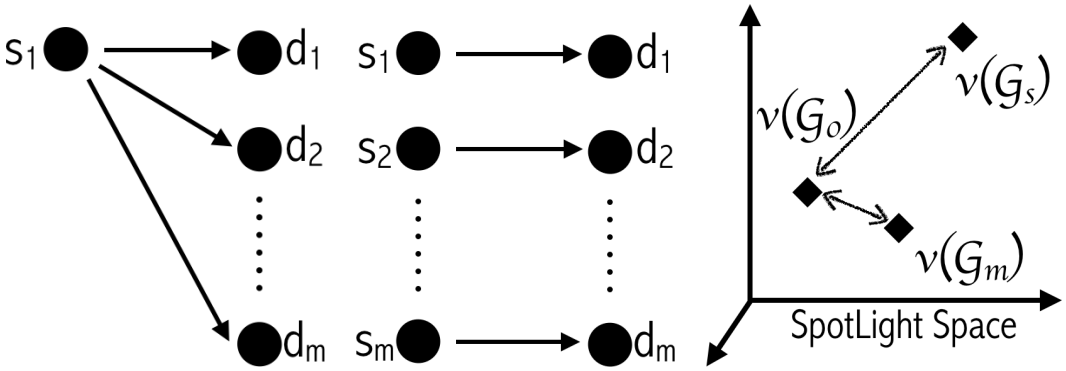
5.5.1.1 Focus-Awareness

Many highly dynamic settings, e.g., IP-IP communication logs, present bursty traffic leading to a high variance in the total edge weight. Thus, it becomes easy for a sudden appearance of dense subgraph, e.g., denial of service attack, to evade detection, unless the distance function used has the so-called *focus-awareness* property: 'random [dispersed] changes in graphs are less important [anomalous] than targeted [focused] changes of the same extent' [KSV+16]. In this section, we show that SL-distance has this desirable property. Consider,

Example 5.1: Star vs. Matching

Add an *out-star graph* (Figure 5.4(a)) of m unit-weight edges (focused change) to $\mathcal{G}$ to obtain $\mathcal{G}_S$. Add a *matching graph* (Figure 5.4(b)) of m edges (dispersed change) to $\mathcal{G}$ to create $\mathcal{G}_M$. Intuitively, the appearance of a dense star subgraph is more anomalous (e.g., potential port scan attack/ hotspot in road traffic) and accordingly, we desire $\bar{d}(\mathcal{G}, \mathcal{G}_S) > \bar{d}(\mathcal{G}, \mathcal{G}_M)$. See Figure 5.4(c).

We show that SL-distance not only satisfies the condition above, but even the distance gap increases with the number of edges m and sketch dimensionality K. That is, $\mathcal{G}_S$ is increasingly more anomalous than $\mathcal{G}_M$ as m grows. See Lemma 5.1.



(a) Star graph **(b) Matching graph** **(c) Ideal sketches**

Figure 5.4: Focus-awareness: Addition of dense star graph is more anomalous than that of the sparse matching graph.

Lemma 5.1: Star vs. Matching

Suppose $\mathcal{G}$, $\mathcal{G}_S$ and $\mathcal{G}_M$ are as defined in Example 5.1, with (K, p, q)-SpotLight sketches $\mathbf{v}(\cdot) \in \mathbb{R}^K$ and let $0 < p, q < 1$. Then, $\bar{d}(\mathcal{G}_S, \mathcal{G}) > \bar{d}(\mathcal{G}_M, \mathcal{G}) + \mathcal{O}\left(Km^2\right)$.

Proof. Let $1 \leq k \leq K$ be a sketch dimension with query subgraph $(\mathcal{S}'_k, \mathcal{D}'_k)$. Define binary random variables $r_{ks} = \mathbb{I}[s \in \mathcal{S}'_k]$ and $u_{kd} = \mathbb{I}[d \in \mathcal{D}'_k]$, where $\mathbb{I}[\cdot]$ is the identity function.

From Figure 5.4a, $v_k(\mathcal{G}_S) - v_k(\mathcal{G}) = \sum_{i=1}^{m} r_{ks_1} u_{kd_i}$. Thus, $\bar{d}(\mathcal{G}_S, \mathcal{G}) = \sum_{k=1}^{K} \mathbb{E}\left[r_{ks_1} (\sum_{i=1}^{m} u_{kd_i})^2\right] = Kmpq + Km(m-1)pq^2$. From Figure 5.4b, $v_k(\mathcal{G}_M) - v_k(\mathcal{G}) = \sum_{i=1}^{n} r_{ks_i} u_{kd_i}$ and so we have $\bar{d}(\mathcal{G}_M, \mathcal{G}) = \sum_{k=1}^{K} \mathbb{E}\left[(\sum_{i=1}^{m} r_{ks_i} u_{kd_i})^2\right] = Kmpq + Km(m-1)p^2q^2$. Thus, $\bar{d}(\mathcal{G}_S, \mathcal{G}) > \bar{d}(\mathcal{G}_M, \mathcal{G}) + \mathcal{O}\left(Km^2\right)$. ■

The edge addition process in Example 5.1 was deterministic, in the sense that the relative position of added edges was fixed. We now consider the more general case where m edges are added uniformly at random (i.e., non-deterministically) in regions of different sizes. Theorem 5.1 shows that the smaller the region in which edges are added, the farther away the final graph lies from the initial graph in the SpotLight space (in expectation). In other words, the more focused the edge addition, the more anomalous the final graph is expected to be in the SpotLight-space.

Theorem 5.1: Focus-Awareness

Consider the distribution of graphs $\mathcal{F}'$ obtained by adding m unit-weight edges (in expectation) to any $n' \times n'$ region of $\mathcal{G}$ by sampling each of the n'^2 possible edges with probability m/n'^2. Let $\mathcal{F}''$ be another distribution over graph obtained by adding edges in a similar manner to any $n'' \times n''$ region. Then,

$$n'' < n' \implies \mathbb{E}_{\mathcal{G}'' \sim \mathcal{F}''} \left[\bar{d}(\mathcal{G}, \mathcal{G}'') \right] > \mathbb{E}_{\mathcal{G}' \sim \mathcal{F}'} \left[\bar{d}(\mathcal{G}, \mathcal{G}') \right] \tag{5.1}$$

Proof. We now state and prove a lemma which we will then use to prove the above theorem.

Lemma 5.2

Let $\mathcal{G}$ be an arbitrary graph and let $\mathcal{G}'$ be obtained by adding $m (\leq n^2)$ unit-weight edges (in expectation) uniformly to any $n \times n$ region of $\mathcal{G}$ by sampling each of the n^2 possible edges independently with probability m/n^2. Assuming n is large and $p{=}q$,

$$\mathbb{E} \left[\bar{d}(\mathcal{G}, \mathcal{G}') \right] = K p^2 m \left(1 + \frac{2pm}{n} + p^2 m \right) \tag{5.2}$$

Further, if $n \gg m$, $\text{Var} \left[\bar{d}(\mathcal{G}, \mathcal{G}') \right] = \mathcal{O} \left(K p^4 m^2 \left(1 + 2p^2 m + p^4 m^2 \right) \right)$, where the expectation and variance have been taken over the random coin tosses of the edge addition process.

Proof. Let $\mathbf{A} = [A_{sd}]$ denote the adjacency of edges added to $\mathcal{G}$ to get $\mathcal{G}'$. Then, $\bar{d}(\mathcal{G}, \mathcal{G}') = \mathbb{E} \left[\left(\sum_{s,d} r_{ks} A_{sd} u_{kd} \right)^2 \right]$, where the expectation is taken over the coin tosses of the algorithm, i.e., $\{r_{ks}, u_{kd}\} \; \forall \; k, s, d$. Using $\mathbb{E}[r_{ks}] {=} \mathbb{E}[u_{kd}] {=} p$. This simplifies to $\bar{d}(\mathcal{G}, \mathcal{G}') = p^2 \sum_{s,d} A_{sd} + p^3 \sum_{s,d \neq d'} A_{sd} A_{sd'} + p^3 \sum_{s \neq s', d} A_{sd} A_{s'd} + p^4 \cdot \sum_{s \neq s', d \neq d'} A_{sd} A_{s'd'}$. To get $\mathbb{E} \left[\bar{d}(\mathcal{G}, \mathcal{G}') \right]$ where the expectation is now taken over the randomness of edge addition, i.e., A_{sd}, we substitute $\mathbb{E}[A_{sd}] = m/n^2$ for $1 \leq s \leq n, 1 \leq d \leq n$ (and otherwise zero) to derive Equation (5.2). Variance calculation, while similar and straight-forward, is omitted in the interest of space. ■

The theorem is now proven by observing Equation (5.2) is decreasing in n. ■

Theorem 5.1 guarantees a separation in the expected SL-distance, (the expectation is taken over the random coin tosses of the edge addition process), which is a *necessary* condition for anomaly detection to work. It is not sufficient, however: in order to detect $\mathcal{F}''$ as anomalies in the SpotLight space, a *large* distance gap with *high* probability is crucial. Section 5.5.1.2 addresses precisely this.

5.5.1.2 Criterion for Anomaly Detection

To show that anomalous graphs are mapped *far* from *normal* instances in the SpotLight space, we need formal definitions for (i) what 'far' means in the sketch space and (ii) what class of

'normal' graphs to use as a control group. These are provided in Definition 5.2 and Definition 5.3 respectively.

Definition 5.2: ϵ-SL-Farness

If $\bar{d}(\mathcal{G}_1, \mathcal{G}) > \bar{d}(\mathcal{G}_2, \mathcal{G}) + \epsilon$, we say that $\mathcal{G}_1$ is ϵ-SL-far from $\mathcal{G}$ compared to $\mathcal{G}_2$.

Definition 5.3: Erdős-Rényi Control Group

Let $\mathcal{G}$ be a graph on N sources and N destinations. An Erdős-Rényi (ER) control group $\mathcal{F}_{\text{ER}}(\mathcal{G}, m)$ is defined as a distribution of graphs, where each instance $\mathcal{G}_{\text{ER}}$ is obtained by adding m unit-weight edges (in expectation) uniformly throughout the graph by sampling each of the N^2 possible edges independently with probability m/N^2.

The choice of ER control group is motivated by focus-awareness: we wish to distinguish the addition of a dense subgraph of m edges in *any focused* part of the graph from a case where the same m edges are added uniformly at random throughout the graph. Theorem 5.2 asserts this is indeed the case: when sketching parameters are chosen appropriately, it is possible to achieve an ϵ-separation between the anomalous and normal graphs with high probability.

Theorem 5.2: Anomaly Detection Criterion

Add n^2 unit-weight edges in any $n \times n$ region to get $\mathcal{G}_{\text{BC}}$ (BC for BiClique). Let $1 \ll n^2 \ll N^2$ and $p = q < 0.5$. Then, $\mathcal{G}_{\text{BC}}$ is ϵ-SL-far from $\mathcal{G}$ compared to a $\mathcal{G}_{\text{ER}}$ drawn from $\mathcal{F}_{\text{ER}}(\mathcal{G}, n^2)$ with high probability $1-\delta$, i.e.,

$$\mathbf{Pr}_{\mathcal{G}_{\text{ER}} \sim \mathcal{F}_{\text{ER}}(\mathcal{G}, n^2)} \left[\bar{d}(\mathcal{G}, \mathcal{G}_{\text{BC}}) - \bar{d}(\mathcal{G}, \mathcal{G}_{\text{ER}}) \geq \epsilon \right] \geq 1 - \delta \tag{5.3}$$

where δ is the false positive rate on the ER control group, provided:

$$K > \frac{(1 + p^2 n^2)^2}{4 p^2 n^2 \delta} + \frac{\epsilon}{p^3 n^3} \tag{5.4}$$

Proof. Let $\mu = \mathbb{E}\left[\bar{d}(\mathcal{G}, \mathcal{G}_{\text{ER}})\right]$ and $\sigma^2 = \text{Var}\left[\bar{d}(\mathcal{G}, \mathcal{G}_{\text{ER}})\right]$. Invoking Chebyshev's inequality, we have with probability $1-\delta$: $|\bar{d}(\mathcal{G}, \mathcal{G}_{\text{ER}}) - \mu| \leq \sqrt{\sigma^2/\delta}$. Thus, if we flag a graph $\mathcal{G}'$ as anomalous if $|\bar{d}(\mathcal{G}, \mathcal{G}') - \mu| > \sqrt{\sigma^2/\delta}$, we erroneously flag δ fraction of the control group as anomalies (false positive rate). In order to detect $\mathcal{G}_{\text{BC}}$ as an anomaly at this threshold, we need $\bar{d}(\mathcal{G}, \mathcal{G}_{\text{BC}}) - \mu - \epsilon > \sqrt{\sigma^2/\delta}$. Under the stated assumptions, $\bar{d}(\mathcal{G}, \mathcal{G}_{\text{BC}}) - \mu \approx 2Kp^3n^3$ and $\sigma^2 \approx Kp^4n^4(1 + 2n^2p^2 + n^4p^4)$. Thus, we derive a quadratic inequality in K resulting in the following, which can then

be relaxed using $a^2 + b^2 \geq 2ab$ to obtain Equation (5.4).

$$K \geq \left(\frac{1 + n^2p^2}{4pn\sqrt{\delta}} + \sqrt{\left(\frac{1 + n^2p^2}{4pn\sqrt{\delta}} \right)^2 + \frac{\epsilon}{2p^3n^3}} \right)^2$$

■

Observe from Equation (5.4) that more sketch dimensions are required if ϵ is high or δ is low which is intuitive: the higher the separation needed between the anomaly and the control group or the lower the permitted false positive rate on the control group, the more dimensions we need. Another subtle point to note here is that Theorem 5.2 guarantees an isolation of anomalies in the sketch space, *without knowing a priori which $n \times n$ region contains the dense subgraph* – this is crucial because, in practice, anomalous dense subgraphs can appear (or disappear) in any region. Further, Theorem 5.2 also guides us in choosing parameters, as stated below.

Corollary 5.1: Optimal Sketching Parameters

From Equation (5.4), the optimal value of p requiring the least sketching dimensionality is obtained by solving $n^5p_*^5 - np_* = 6\epsilon\delta$. When ϵ=0, this reduces to p_*=$1/n$ i.e., sample exactly one *added* edge in expectation. Accordingly, we require $K_* > 1/\delta$.

Proof. Setting the first derivative of RHS of Equation (5.4) to zero, we get $n^5p_*^5 - np_* = 6\epsilon\delta$ (second derivative at $p_* \geq 0$). ■

For example, with K=$50, p$=q=0.2, we may detect the addition of n=5 biclique as an anomaly with ϵ=0 separation by incurring at most δ=2% false positive rate on the ER control group.

5.5.2 Time and Space Complexity

SpotLight obeys the sublinear memory and linear time constraints of Problem 5.1, as stated below.

Lemma 5.3: Linear Running Time

SpotLight takes $\mathcal{O}\left(|\mathcal{E}| \cdot K\right)$ time to process each $\mathcal{G} = (\mathcal{S}, \mathcal{D}, \mathcal{E})$ in the stream.

Lemma 5.4: Sublinear Memory Requirement

SpotLight takes $\mathcal{O}\left(\ln N_s + \ln N_d + K\right)$ to process each graph in a stream having N_s sources and N_d destinations.

SpotLight sketching runs in $\mathcal{O}(|\mathcal{E}| \cdot K)$ running time due to the loops in lines 11-12 (Algorithm 2), since the other steps require constant time. The $\mathcal{O}(\ln N_s + \ln N_d)$ space is a lower bound on memory requirements, since each edge (including source and destination identifiers) needs to be read (one by one). An additional $\mathcal{O}(K)$ space is needed to store the sketch. Anomaly detection in SpotLight space takes $\mathcal{O}(K)$ time and sublinear space, e.g., using [GMRS16].

5.6 Experiments

We empirically evaluate the proposed method on datasets where the anomalies are verifiable and interpretable. We begin with the details of datasets and experimental setup.

5.6.1 Datasets

We shortlist three real-world publicly available time-evolving graph datasets, where the anomalies can be verified by comparing to manual annotations or by correlating with real-world events:

Darpa dataset [LCF+99] contains $4.5M$ IP-IP communications taking place between 9484 source IPs and 23398 destination IPs over $87.7K$ time steps (minutes). Each communication is a directed edge *(srcIP, dstIP, 1, time)*. We obtain a stream of 1463 graphs by aggregating edges occurring in every hourly duration. The dataset contains 89 known network attacks – large or stealthy – e.g., `portsweep`, `ipsweep`, `mscan` and `snmpgetattack`. Most attacks were large (> 100 edges), but were targeted at and/or engineered from a few hosts and occurred in single/multiple bursts of time – thus, leading to the *sudden (dis)appearance of large dense subgraphs* that we aim to detect. Using the furnished ground truth (attack/not) for each edge, we label a graph as anomalous if it contains at least 50 attack edges.

Enron dataset [SA04] contains $\sim 50K$ emails exchanged among 151 employees of the energy company over a 3 year period surrounding the famous Enron scandal. Each email is a directed edge *(sender, receiver, 1, timestamp)*. We derive a stream of 1139 graphs by treating each day as its own graph. As ground truth is not directly available, we verify the detected anomalies by correlating with the major events of the scandal.

NycTaxi dataset [nyt18] contains taxi ridership data during a 3-month period (Nov 2015–Jan 2016) obtained from New York City (NYC) Taxi Commission. Each taxi trip is furnished with pick-up (PU)/drop-off (DO) times and (lon, lat) coordinates of PU/DO locations, which we process as follows. We manually click on the centers of 57 geographically or conceptually distinguishable NYC *zones* based on common knowledge – including parks, airports, stadiums, bridges, residential neighborhoods, islands – on a map and note their (lon, lat) coordinates. Every PU/DO location is then assigned to the nearest *zone*. Thus, a directed edge *(srcZone, dstZone, 1, timestamp)* is created for each taxi trip. These are further aggregated into 2208 graphs, each containing trips that took place in a given hourly duration. We verify the detected anomalies by

correlating with important occasions – holidays, events, unusual weather conditions – which affect the normal rhythm of road traffic.

5.6.2 Experimental Setup

We implement SpotLight (abbreviated as SL henceforth) in Python and run experiments on MacOS with 2.7 GHz Intel Core i5 processor and 16 GB main memory. By default, we use $K{=}50$ sketch dimensions and $p{=}q{=}0.2$ source/destination sampling probabilities. Mapping to Theorem 5.2, this corresponds to detecting a $n{=}5$ biclique (or more) as an anomaly w.r.t. the control group by incurring less than $\delta{=}2\%$ false positives. This also ensures all edges are covered twice in expectation. For the anomalous sketch detection step, we use the state-of-the-art Robust Random Cut Forests (RRCF) [GMRS16] with 50 trees and 256 samples (unless specified otherwise).

Baselines: We compare SpotLight to the following three baselines on the labeled Darpa dataset: **(a) EdgeWeight (EW):** We consider a vanilla version of SL by setting $K{=}p{=}q{=}1$, i.e., sketching each graph using a single coarse-grained feature, namely, its total weight of edges. Observe that EW tends to miss 'small' anomalies which do not alter the total edge weight significantly compared to usual. **(b) RHSS [RHSS16]**, abbreviated based on the last names of authors, processes each edge e in the stream individually, outputting a likelihood score $\ell(e)$. We compute the likelihood of a graph $\mathcal{G} = (\mathcal{S}, \mathcal{D}, \mathcal{E})$ as the geometric mean of the per-edge likelihoods (similar to [AZY11]): $\ell(\mathcal{G}) = (\prod_{e=(s,d,w)\in\mathcal{E}} \ell(e)^w)^{1/W}$ where W is the total edge weight. Finally, to reflect the intuition that a more likely graph is less anomalous, we use $\textit{anomaly\_score}(\mathcal{G}) = -\ln\ell(\mathcal{G})$. We implement RHSS in Python without using the sketching-based approximation[1]. **(c) STA [STF06]** scores the anomalousness of each graph as the error incurred in reconstructing it based on a streaming graph decomposition. We use 50 as the rank of decomposition.

Evaluation Metrics: Each method above outputs an anomaly score (higher is anomalous) per graph. Sorting these in descending order, we compute the number of anomalies caught $TP(k)$ (true positives) among the top k most anomalous graphs, for every k. If the overall number of anomalies is N, we compute *precision*@k $= TP(k)/k$ and *recall*@k $= TP(k)/N$. We also summarize the overall accuracy using the AUC (Area Under ROC Curve) score. Recall that *precision*@k, *recall*@k and AUC lie in $[0, 1]$ and a higher value is better. In addition, we note the running time of all methods, averaged over five runs.

Experimental Design: Our experiments are designed to answer the following questions: **[Q1] Accuracy**: How well is SpotLight able to spot anomalies compared to baselines? What is the trade-off with respect to running time? How does the performance vary with parameters? **[Q2] Scalability**: How does the running time scale with the number of edges in the stream and

[1]We also tried computing the anomaly score as the negative average of the per-edge likelihoods and obtained similar results.

Method	precision@ 100	200	300	400	recall@ 100	200	300	400
Ideal	*1.0*	*1.0*	*0.96*	*0.72*	*0.35*	*0.69*	*1.0*	*1.0*
SL	**0.96**	**0.79**	**0.64**	**0.57**	**0.34**	**0.55**	**0.67**	**0.80**
EW	0.86	0.54	0.47	0.46	0.30	0.38	0.49	0.65
RHSS	0.31	0.28	0.32	0.36	0.11	0.19	0.33	0.50
STA	0.23	0.16	0.19	0.24	0.08	0.11	0.20	0.34

Table 5.2: SpotLight (SL) achieves better precision and recall than baselines (EW, RHSS, STA). Bold indicates the highest value in each column (excluding ideal). Underline shows significant differences (p-value ≤ 0.01) w.r.t. baselines according to a two-sided micro-sign test [YL99].

sketch dimensions K? **[Q3] Discoveries**: Does SpotLight lead to interesting discoveries on real world data? We now present our findings.

5.6.3 Q1. Accuracy

Table 5.2, Figure 5.5 and Figure 5.6 compare the precision, recall, accuracy (AUC) and running time of SL with baselines on the labeled Darpa dataset. Figure 5.7 shows the variation of accuracy with parameters. As SL and EW are initialized based on the first 256 graphs, performance is reported on the subsequent $1463 - 256 = 1207$ graphs, containing 288 ground truth anomalies (23.8% of total).

Precision and Recall: Table 5.2 gives the precision and recall at cut-off ranks $k \in \{100, 200, 300, 400\}$. Ideal values are computed based on an oracle which scores the ground truth anomalies higher than all non-anomalies. We see that SL consistently outperforms all baselines achieving $11 - 46\%$ (statistically significant) improvements. Further, a plot of precision vs. recall for all methods, shown in Figure 5.5, reveals that SL's curve (blue) lies completely above those of all baselines, achieving higher precision for every recall value. Thus, the performance gain of SL generalizes to all cut-off ranks (k).

Accuracy vs. Running Time: Figure 5.6 plots the accuracy (AUC) of each method vs. its running time (in seconds). We see that SL achieves the highest accuracy (=0.91), 8.4% higher than EW (=0.83) and 30% higher than RHSS (=0.70). This gain comes at a cost of a mere $4\times$ slow down compared to EW and RHSS. STA, which computes graph decomposition, was considerably slower.

Accuracy w.r.t. Sampling Probabilities p, q: Figure 5.7(a) shows how the accuracy varies with source (p) and destination (q) sampling probabilities for K=10 dimensions, after tying p=q for simplicity. We see that the poor accuracy results from choosing very low (anomalous dense subgraphs are easily missed as very few nodes are sampled resulting in a sketch with

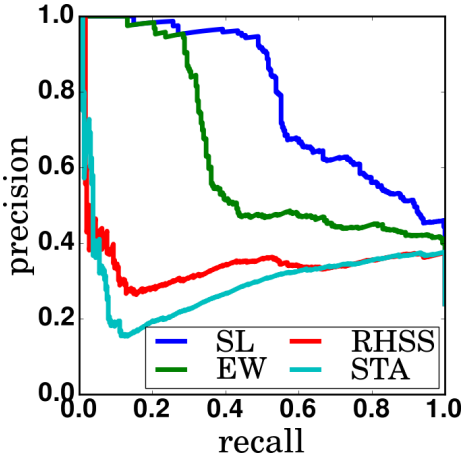


Figure 5.5: SL outperforms baselines in terms of precision and recall.

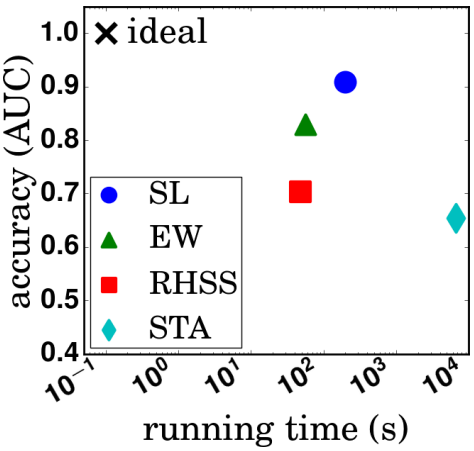


Figure 5.6: Accuracy-running time trade-off offered by SL.

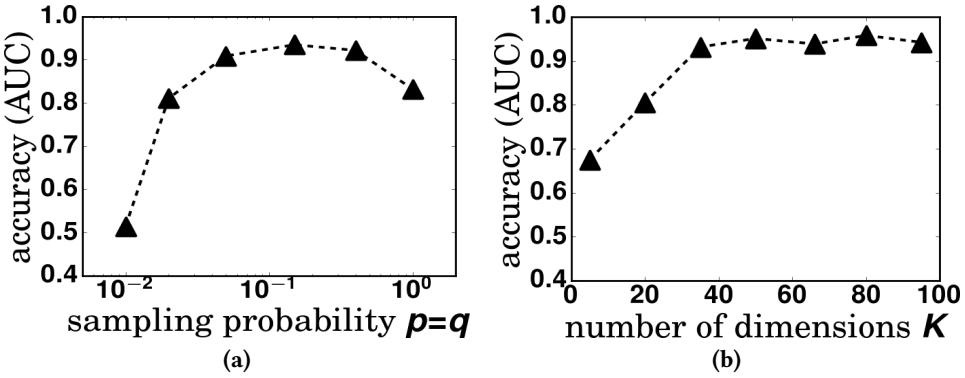


Figure 5.7: Variation of accuracy with (a) $p = q$ **for** $K = 10$ **and (b) with** K **for** $p = q = 0.1$**.**

mostly zeroes) and very high (sketch dimensions are coarse-grained, similar to EW, as almost all nodes are sampled) node sampling probabilities. The sweet spot lies in between. Over a large interval $[0.05, 0.4]$, the accuracy remained fairly robust (insensitive) to the exact value of p.

Accuracy w.r.t. #dimensions K: Figure 5.7(b) shows the variation of accuracy with the number of sketch dimensions $K \in \{5, 20, 35, 50, 65, 80, 95\}$ for $p = q = 0.1$. We see that accuracy increases rapidly from 0.67 to 0.95 as K is increased from 5 to 50, beyond which it stabilizes around 0.95. This is the classic 'diminishing returns' pattern we expect. When K is low, an added SpotLight sketch dimension likely 'illuminates' a new part of the graph and detects anomalies that were previously undetected, but once K crosses a threshold (here, 50) when most of the graph is already 'illuminated', a new sketch dimension gives little to no added benefit.

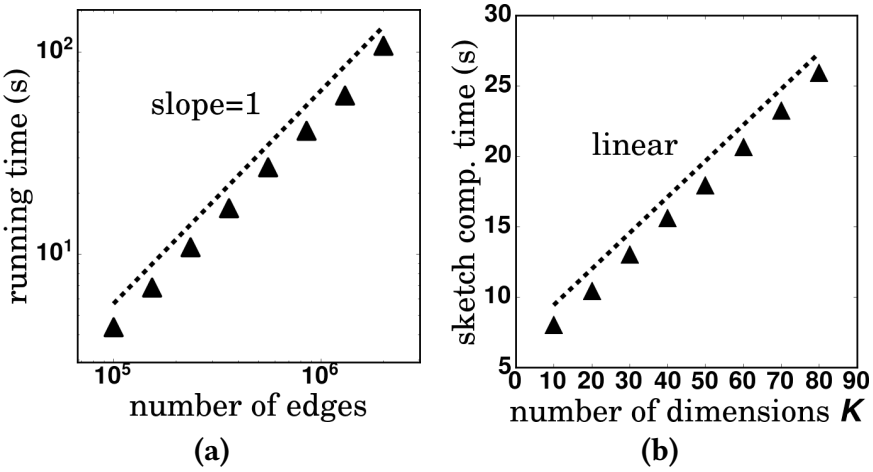


Figure 5.8: SL scales linearly with the number of (a) edges in the stream and (b) sketch dimensions K.

5.6.4 Q2. Scalability

Figure 5.8 shows the scalability of SL with the number of edges and sketch dimensions. We use RRCF with 10 trees and sample size 128.

With #edges: We uniformly sample $100K - 2M$ edges from the DARPA dataset in eight logarithmic steps and timed SL. Figure 5.8(a) plots the running time (in seconds) vs. the number of edges in log-log scales. We see that the points align with a line of slope 1, indicating SL scales linearly with input size (as is desirable). Note also that SL is fast and is able to process $2M$ edges in less than 2 minutes!

With #dimensions: We now vary the SPOTLIGHT sketch dimension $K \in \{10, 20, \ldots 70, 80\}$ and measure the time taken to compute sketches for $0.5M$ edges. Figure 5.8(b), plotting the running time (in seconds) with the number of dimensions, reveals that SL scales linearly with the dimensionality of SPOTLIGHT sketch.

These are consistent with our expectations based on Lemma 5.3.

5.6.5 Q3. Discoveries

We provide a complete analysis of SL and baselines on the labeled DARPA dataset; in the interest of space, we only summarize the discoveries due to SL on ENRON and NYCTAXI datasets, omitting baseline results.

5.6.5.1 DARPA

Leveraging ground truth, we now delve deeper into why the baselines perform poorly compared to SL on DARPA dataset. Figure 5.9 plots the anomaly scores (higher is anomalous) of all methods along with ground truth (spikes in the 'ideal' black curve). Our explanation will use Figure 5.10, which plots the number of attack (red) and non-attack (green) edges over time t. In these figures,

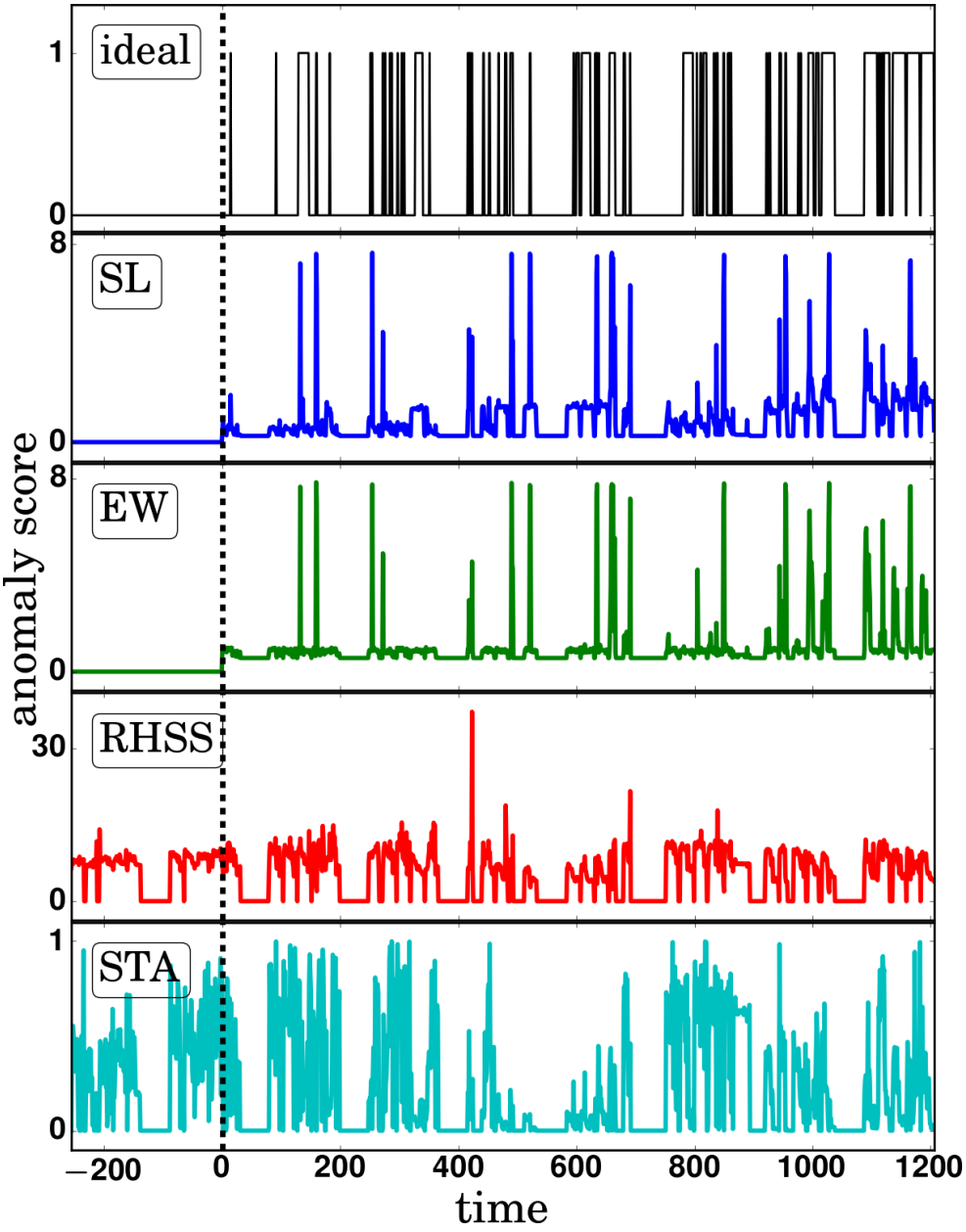


Figure 5.9: Anomaly detection on DARPA dataset. Spikes in the 'ideal' black curve indicate ground truth anomalies.

$t < 0$ corresponds to the initialization period for SL and EW, resulting in zero anomaly score. We now examine each baseline separately.

EW: Around $t{=}\{150, 450, 650, 850, 1000\}$, Figure 5.10 shows several spikes (of height $10^4{-}10^5$) in attack weight (red); these are significantly higher than the non-attack weight (green) which never exceeds 10^4. Hence, these 'large' anomalies are easily detected by tracking only the total edge weight (green spikes in Figure 5.9). However, EW fails to detect anomalous graphs in which the total weight of edges is comparable to that observed at many prior graphs – e.g.,

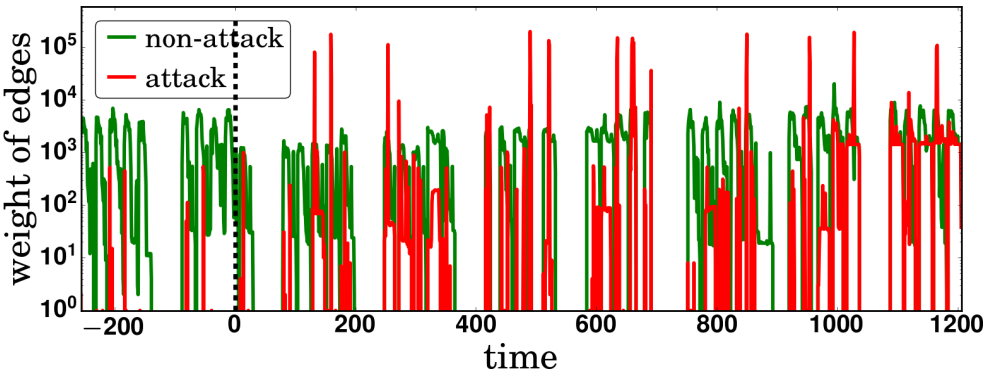


Figure 5.10: Understanding (un)detected anomalies in DARPA using the number of attack and non-attack edges over time.

anomalies around t={1, 300, 500}. On the other hand, SL keeps track of the total weight of edges *in several local regions within the graph*; since attack edges are concentrated in regions of the graph where non-attack edges typically do not occur, these are detected by SL, even if the weight of attack edges is small, e.g., at t=1.

RHSS: RHSS scores each graph based on the likelihood of its edges computed based on its prior occurrence, preferential attachment and homophily. Simply put, (graphs containing) edges which are seen before or which connect high degree nodes or nodes having many common neighbors are non-anomalous. However, we find that these assumptions are more suited to slowly-evolving social networks rather than highly dynamic settings. To see why, consider: (a) *Repeated attacks:* `neptune`[2] attack occurs at 33 different times, including $t = -204$, which is within the initialization period. Once RHSS has 'seen' all `neptune` attack edges, subsequent occurrences, however rare and dense, are not found anomalous. (b) *Repeatedly attacking (victimized) nodes:* Once a node has (been) attacked sufficiently many times, it attains a high degree; consequently, further attacks by (or on) it are 'likely' (due to preferential attachment) and non-anomalous.

STA: STA computes *a single* graph decomposition model to summarize the data seen so far – admittedly, a much harder problem than anomaly detection – and scores the anomalousness of each graph as the error incurred in reconstructing it from the model. The assumption of a single normal behavior does not apply to dynamic settings (such as this) – e.g., in Figure 5.10, it is as normal for the number of non-attack edges to be around 1000 as it is to be 0 – consequently, STA is very sensitive in practice and leads to numerous false alarms.

5.6.5.2 ENRON

Figure 5.11 plots the anomaly score vs. time for ENRON dataset, after initializing SL based on the first 256 days (05/12/99-01/22/00) with shingle length 7 (weekly periodicity). We examine the top 6 non-consecutive time durations having the highest anomaly scores. As we show

[2]A SYN flood denial of service attack to which every TCP/IP implementation is vulnerable to some extent. See www.ll.mit.edu/ideval/docs/attackDB.html.

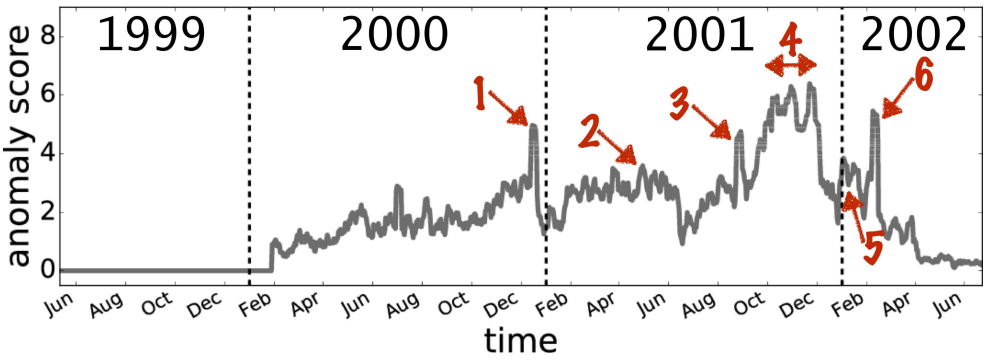


Figure 5.11: Anomaly detection on Enron dataset

below, these anomalies correspond to major events – either company-wide emails or public announcements triggering excitement or confusion – in the Enron time line[3].

2000: (1) Dec 13-14: Skilling announced as CEO. **2001:** (2) May 23: Enron completes its millionth transaction via Enron Online. (3) Sep 28: Lay to employees: 'Third quarter is looking great.' (4) Oct 7-Nov 22: Wall Street Journal article reveals Enron's precarious state. One ton Enron documents shredded. Fastow ousted. SEC launches formal investigation. Restructuring of $690M obligation is announced. **2002:** (5) Jan 23-30: Lay resigns as chairman and CEO. Baxter commits suicide. Cooper takes over as CEO. (6) Feb 7-8: Fastow, Kopper and Skilling testify before Congress.

5.6.5.3 NycTaxi

Figure 5.12 plots the anomaly score vs. time for NycTaxi dataset, after initializing SL based on the first 256 hours ($\sim$ 10 days) of Nov 2015 with shingle length 24 (daily periodicity). As before, we examine the top 6 non-consecutive time durations having the highest anomaly scores.

The most anomalous period (Jan 23-24) coincided with the January 2016 United States blizzard which produced a historic 3 feet of snow and rendered normal traffic operation impossible. The next three anomalies (around Nov 27, Dec 25, Jan 1) corresponded to festival periods – Thanksgiving, Christmas, New Year – presumably due to unusual traffic patterns around Manhattan (closed offices, Macy's Thanksgiving parade, New Year parties) and airports (people flying in/out of JFK and LaGuardia). The next two anomalies (Nov 14, Nov 29-30) are more interesting because they do not coincide with holidays or weather conditions, and as such, are not expected to be anomalous.

To further understand why Nov 14 and Nov 29-30 were flagged, we derive an anomaly score per sketch dimension from RRCF and *propagate the anomalousness* to NYC zones. Thus, the anomaly score of a zone is the sum of anomaly scores of all dimensions it participates in. The most anomalous zones during these dates turned out to be Bedford on Nov 14 and LaGuardia airport on Nov 29-30. Digging deeper, we discovered that these locations popped up in several archived new articles on these dates. At 12pm Nov 14, 'huge fire [ripped] through Bedford-

[3]verified using www.agsm.edu.au/bobm/teaching/BE/Enron/timeline.html

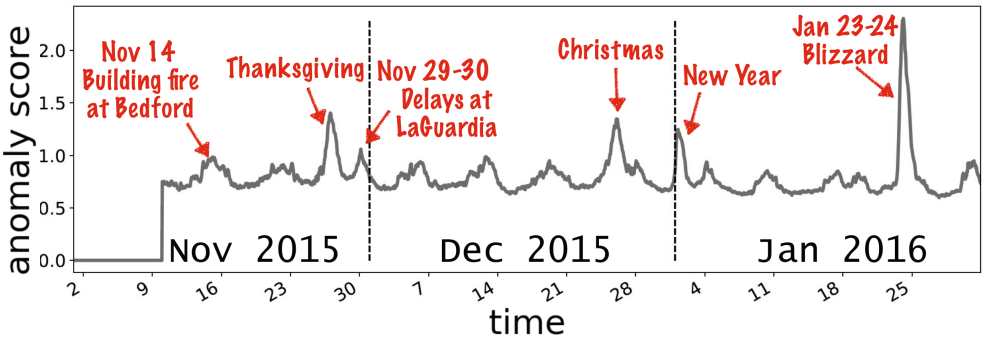


Figure 5.12: Anomaly detection on NycTaxi dataset

Stuyvesant building'[4] threatening its collapse and creating unusual traffic in/out of the area. On Nov 29-30 (Sunday after Thanksgiving), 'thousands [were] delayed at airport in an attempt to return home after Thanksgiving'[5] causing the usual morning rush hour traffic at LaGuardia to persist throughout the day with over an hour-long wait times for taxis.

Thus, the *sudden (dis)appearance of large dense subgraphs* detected by SL on real-world data have a practical significance, from network attacks in IP-IP communication logs to holidays, abnormal weather or local traffic conditions in transportation logs.

5.6.6 Discussion

Why do SL/EW perform better than STA/RHSS? STA and RHSS make strict modeling assumptions, e.g., stable community structure or homophily, restricting their scope to limited settings, e.g., slowly evolving graphs, friendship networks. In contrast, EW and SL use a less restrictive definition of anomaly which is applicable to a wider variety of highly dynamic settings. **Can the detected anomalies be attributed to few nodes?** Yes, by explicitly maintaining the node to sketch dimension mapping and following the 'anomalousness propagation' heuristic in Section 5.6.5.3.

5.7 Conclusion

We presented a simple, scalable, easy-to-code algorithm called SpotLight for sketching a graph. SpotLight sketches facilitate fast and reliable identification of anomalies, where an anomaly is the sudden appearance (or disappearance) of a large dense directed subgraph. Theoretical analysis provides concrete settings where there is a provable distance gap in the sketch of a graph where m edges are scattered at random throughout the graph (dispersed) vs. the sketch of a graph where m edges are added in a smaller subgraph (focused). The distance gap sets the stage for classic anomaly detection algorithms to spot the more distant graph. Experiments

[4] www.nydailynews.com/new-york/huge-fire-rips-bedford-stuyvesant-building-article-1.2435059

[5] pix11.com/2015/11/29/thousands-delayed-at-airport-in-an-attempt-to-return-home-after-thanksgiving/

on a variety of real-world datasets demonstrate that SpotLight outperforms prior approaches in terms of both precision and recall. Yet, many new opportunities remain. Adaptive data-driven sketches, while harder to analyze, may yield better results in practice. Interpretability and anomaly attribution are also important questions. Finally, the trajectory of an anomaly is vital to both understand and predict.

Chapter 6

SedanSpot: Anomalous Edge Detection

Chapter based on work that appeared at ICDM 2018 [EF18] [PDF].

Given time-evolving IP-IP network traffic, how can we identify malicious communications as soon as they occur? How can we quickly spot suspicious calls/messages/e-mails (possibly, scam) from communication logs? More generally, given a stream of edges from a time-evolving (un)weighted (un)directed graph, how can we detect anomalous edges *in near real-time using sublinear memory*? To this end, we propose SedanSpot, a principled randomized algorithm, which exploits two tell-tale signs of anomalous edges: they tend to (i) occur as bursts of activity and (ii) connect parts of graph which are sparsely connected. SedanSpot has the following desirable properties: *(a) Burst resistance:* It provably downsamples edges from bursty periods of network traffic, *(b) Holistic scoring:* It takes into account the whole (sampled) graph while scoring the anomalousness of an edge, giving diminishing importance to far-away neighbors, *(c) Efficiency:* SedanSpot supports fast updates and scoring and hence can be efficiently maintained over stream; further, it can detect anomalous edges in sublinear space and constant time per edge. Through experiments on real-world datasets, we demonstrate that SedanSpot is fast and accurate, outperforming the state-of-the-art by 270% in terms of AUC while taking $3\times$ less time.

6.1 Introduction

Time-evolving (un)weighted (un)directed graphs, where edges and vertices arrive continuously over time, are becoming increasingly ubiquitous. Examples include phone call networks (user u calls user v at time t, speaking for w seconds), instant-messaging/e-mail networks (user u sends user v a message/e-mail of size w at time t), IP-IP networks (machine u sends machine v a packet of size w at time t) and so on. In these settings, edges are generated in increasing order of their time-stamps, giving rise to a stream of edges, or *edge streams*.

We consider the problem of *near real-time anomaly detection* in such edge streams, where the goal is to detect whether an incoming edge is anomalous or not, as soon as it is received. While graph anomaly detection is a well-explored research area, most methods apply to offline settings, or online settings where the edges have been aggregated into graph snapshots (elaborated in Section 6.2). In contrast, we seek algorithms which directly process the edge stream to flag anomalies in near real-time, which is crucial to curtail the impact of malicious activities and kick-start recovery processes in a timely manner. Moreover, given that the number of vertices is not known a priori and can grow as the stream progresses, the algorithm should operate in memory sublinear in graph size. Informally, the problem we set out to solve is:

Informal Problem 6.1

Given an edge stream $\mathfrak{E}=\{e_1, e_2, \ldots\}$ from a/an (un)weighted (un)directed graph, detect whether e_i is anomalous, in near real-time using sublinear memory.

Due to the fluid nature of what is considered 'normal', prior works typically focus on detecting particular anomalous changes to the graph such as dense subgraphs [SHF18], hotspot vertices [YAMW13] and changes to community structure [SFPY07]. In this work, we focus on detecting edges *which connect sparsely-connected parts of graph* (e.g., bridge edges). Figure 6.1 illustrates this. In an edge stream from an unweighted directed graph, the edges received until time $t=0$ form two clusters of vertices $\{(a_1, \ldots, a_5), (b_1, b_2, b_3)\}$. Thus, edges $a_4 \rightarrow b_2$, $a_4 \rightarrow b_1$ and $a_4 \rightarrow b_3$ (occurring at $t=7$) which connect these otherwise disconnected clusters of vertices should be flagged anomalous (possibly an 'attack' by a_4 on the (b_1, b_2, b_3)-cluster).

The *simultaneous occurrence* of edges highlighted in red (at $t=7$) in Figure 6.1 is not coincidental. Prior work has shown that fraudulent or important events in many real-world applications indeed occur as *spikes or bursts of activity* (e.g., [BXG+13, GGF14a]). Examples include network security threats (port scan, denial of service), scams (malicious entities attacking many victims before they get exposed), occasions (festivals producing a burst of longer-than-usual phone calls) and so on. Anomaly detection approaches which do not account for this observation ([AZY11, RHSS16]) tend to miss several anomalies, e.g., $a_4 \rightarrow b_3$ being masked as normal by the recent occurrences of $a_4 \rightarrow \{b_1, b_2\}$.

While anomalous activity tends to occur as bursts, burstiness does not necessarily signify an anomaly: in dynamic situations like network traffic, normal activity can also be bursty. Thus, in order to reliably detect anomalies, we need to combine the *temporal* dynamics of the edges with the graph *structural* information. The proposed method, called SedanSpot (short for Streaming EDge ANomaly SPOTter) does precisely this: it detects unexpected edges which connect sparsely-connected regions *in the face of spikes of anomalous activity*. We note that there other kinds of anomalies (e.g., low temperature or periodic attacks), but they are not the focus of this work.

Given the running time and memory constraints of Problem 6.1, SedanSpot maintains an online sample of edges (using SedanSampler) which is then used to score the anomalousness of any new edge (via SedanScorer). SedanSpot has the following desirable properties. *(a)*

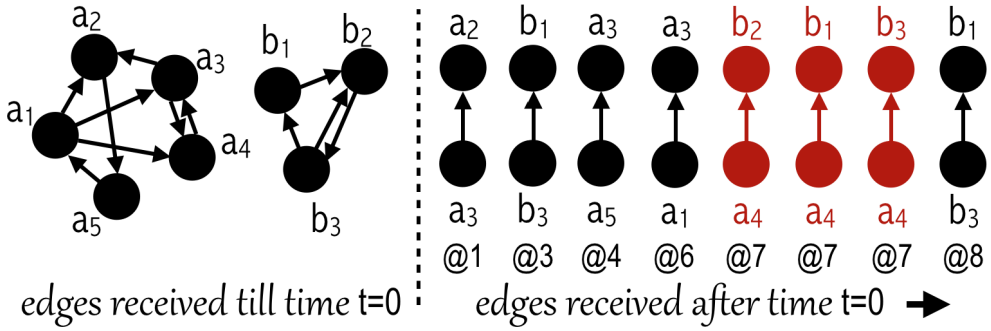


Figure 6.1: An edge stream showing a burst of three anomalous 'bridge' edges (highlighted in red).

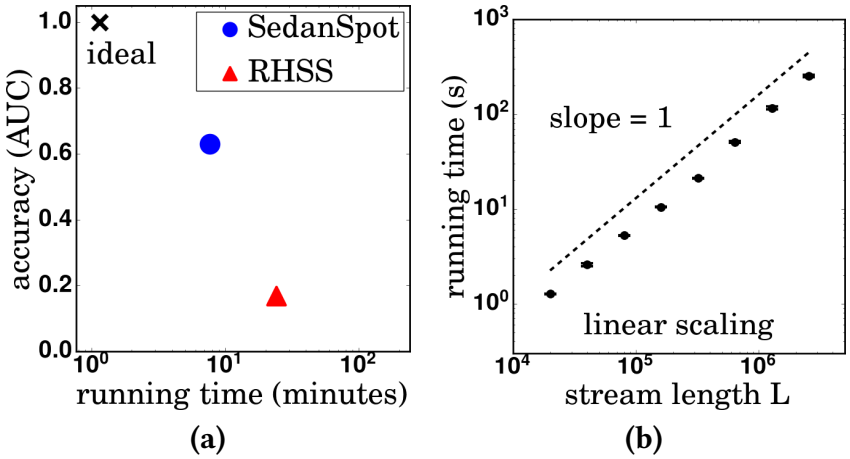


Figure 6.2: Overview of experimental results: SedanSpot (a) outperforms the state-of-the-art in terms of both accuracy and speed and (b) scales linearly with the number of edges in the input stream.

Burst resistance: SedanSampler provably downsamples edges from bursts of activity, *(b) Holistic scoring:* SedanScorer scores the anomalousness of edges by taking into account the whole (sampled) graph, giving diminishing importance to far-away neighbors, *(c) Efficiency:* SedanSpot supports fast updates and scoring and hence can be efficiently maintained over stream; further, it can detect anomalous edges in sublinear space and constant time per edge. Overall, SedanSpot is fast and accurate, outperforming the state-of-the-art by 270% in terms of AUC while taking $3\times$ less time (Figure 6.2(a)) and scaling linearly with the number of edges in the stream (Figure 6.2(b)).

Reproducibility: We use publicly-available datasets and open-source our code at `https://github.com/dhivyaeswaran/sedanspot`.

6.2 Background and Related Work

We review related work on graph anomaly detection and provide background on random walk with restart and sampling in streams, which the proposed SedanSpot is based on.

6.2.1 Anomaly Detection in Graphs

Offline anomaly detection of static or time-evolving graphs is a well-explored research area (for surveys, see [ATK15, RSK+15]). Unsupervised methods for static graphs rely on comparing node- or egonet-level features [AMF10], matrix factorization [TL11], graph partitioning [Cha04], node proximity measures such as Personalized Page Rank [Hav03], Katz measure [Kat53], etc. (see [LK03] for a comparative study of such measures). Any available labels can be leveraged by employing semi-supervised methods, e.g., belief propagation [EGF+17b]. Offline, unsupervised methods for dynamic graphs typically incorporate the timestamps on edges by modeling them as node/edge attributes [HSB+16a] or finding dynamic dense subgraphs occurring in short time intervals [RTG17] or spotting suspicious dense sub-tensors in *node* × *node* × *time* tensor via decomposition [KB09] or greedy strategies [SHF18].

Online (streaming) anomaly detection methods can be further divided into those which operate on (1) aggregated graph streams or (2) raw edge streams.

Graph Streams: Many methods assume that the raw edge stream has been processed into a stream of graph snapshots (each containing edges occurring in a given duration). [STF06] maintains a streaming tensor decomposition and uses it to detect and attribute events/changes via the reconstruction error of a new tensor. [SFPY07] uses graph partitioning and minimum description length to detect change points. [GGSH12] detects evolutionary community outliers, i.e., nodes which evolved differently compared to their communities. [RGNH13] uses non-negative matrix factorization to determine the belongingness of nodes to 'roles', which is then used to flag anomalies. [WFLW15] finds anomalous nodes using vector auto-regression on the features of nodes and their communities. [SD14] and [KSV+16] compare consecutive snapshots of graphs through similarity/distance functions related to random walks [KKK+11]. Specifically, [SD14] uses commute-time embedding, whereas [KSV+16] uses a fast variant of belief propagation. While applicable only to undirected graphs, these can also attribute anomalousness to nodes or edges.

Edge Streams: GOutlier [AZY11] scores the likelihood of each edge in the stream by maintaining a 'structural reservoir sample of edges' to induce node partitioning and tracking the probability of an edge connecting two of those partitions. [YAMW13] detects nodes having unusual levels of activity or structural changes by performing egonet-level Principal Component Analysis at multiple temporal granularities. [RHSS16] scores the anomalousness of each edge in the stream based on its prior occurrence, preferential attachment and mutual neighbors (homophily). [MMA16] is related, but applies only when *multiple* unweighted graphs with *typed* nodes and edges are evolving simultaneously.

Table 6.1: Qualitative comparison with closely-related prior work

Property	DC [KSV+16], CAD [SD14]	AH [YAMW13]	GOutlier [AZY11]	RHSS [RHSS16]	SedanSpot
Online (operate on edge streams)		✔	✔	✔	✔
Generality (weighted/directed)				✔	✔
Burst resistance	N/A				✔
Holistic scoring	✔				✔
Efficiency (sublinear memory)			?	✔	✔
Efficiency (constant time per edge)	N/A		✔	✔	✔

As such, none of the prior methods have all the desirable properties that SedanSpot exhibits, as shown in Table 6.1.

6.2.2 Random Walk with Restart (RWR)

Consider a random surfer starting at a vertex u of a given (un)directed graph. At each step, he returns to u with probability α or jumps to a random out-neighbor based on edge weight. The steady state probability that the surfer will finally remain at v is termed the RWR relevance score of v w.r.t. u and is an attractive measure of vertex proximity in many applications, e.g., search and retrieval [TFP06]. Concretely, let $\mathbf{A}$ be the adjacency matrix with A_{uv} denoting the (non-negative) total weight of edge from u to v. Define row-normalized adjacency $\bar{\mathbf{A}}$ as $\bar{A}_{uv} = A_{uv} / \sum_a A_{ua}$ when $\sum_a A_{ua} > 0$, or zero otherwise. If $\mathbf{q}_u$ is the n-dimensional binary vector where all but the u^{th} entry are zeros, the vector of RWR relevance scores $\mathbf{r}_u$ of all nodes w.r.t. u is given by:

$$\mathbf{r}_u = (1 - \alpha)\bar{\mathbf{A}}^T \mathbf{r}_u + \alpha \mathbf{q}_u \tag{6.1}$$

Equation (6.1) is typically solved by repeating the above update till convergence, which takes $\mathcal{O}\left(\text{NNZ}(\bar{\mathbf{A}})\right)$ time per iteration where $\text{NNZ}(\cdot)$ is the number of non-zero entries. Since this can be expensive, fast approximate variants of RWR is a growing research area. In particular, local random walks (which we use) are successfully applied to link prediction [LL10] and recommendation [EJL+18]. Existing work on RWR relevance score computation for edge streams either assume a single start vertex known ahead of time [YJK18] or maintain all-pair relevance scores [YM16]; thus they are are not applicable to our setting.

6.2.3 Sampling in Streams

Reservoir Sampling [Vit85] is classic algorithm to maintain a fixed-size uniform sample of elements in a stream. Weighted Reservoir Sampling [ES06] is used when elements are to be sampled with different weights. When the stream contains edges from a graph, several application-specific sampling mechanisms exist for counting triangles [SERU17], wedges [ADWR17], etc. A comprehensive treatment is given in [ANK14]. However, none of these techniques downsample edges from bursty periods, which is needed to reliably detect anomalies (e.g., in Figure 6.1).

6.3 Problem Framework

We begin with some notation. Let $\mathfrak{E} = \{e_i\}_{i=1}^{\infty} = \{e_1, e_2, \ldots, e_L, \ldots\}$ be the stream of edges from an underlying time-evolving graph $\mathfrak{G}$. Each element e_i in the stream is 4-tuple (u_i, v_i, w_i, t_i) of a source vertex $u_i \in \mathcal{V}$, a destination vertex $v_i \in \mathcal{V}$, edge weight w_i and time of occurrence t_i and represents the addition of this edge to the graph $\mathfrak{G}$. Here, $\mathcal{V}$ is the set of all vertices, which is not known a priori but changes as $\mathfrak{G}$ evolves, losing old vertices and gaining new ones. However, each vertex is assumed to have a unique identifier, e.g., user ID or IP address, that is fixed over time.

Note that $\mathfrak{G}$ represents a *multigraph*; hence, two vertices may be (and usually are) connected multiple times, each time with a possibly different weight. Any number of edges could arrive at the same time, thus $t_{i+1} \geq t_i$. Further, depending on the nature of $\mathfrak{G}$, the edges can be weighted ($w_i = 1 \; \forall \; e_i$, if unweighted) and/or have direction (assume a 'fake' (v_i, u_i, w_i, t_i) for every (u_i, v_i, w_i, t_i) when $u_i \neq v_i$, if undirected). We will also sometimes overload $t(e)$ and $w(e)$ to denote the timestamp and weight of edge e.

Next, we give the proposed framework to solve Problem 6.1.

6.3.1 Subproblems

Our goal is to detect anomalous edges by leveraging their temporal and spatial signals: they tend to (i) occur as bursts of activity and (ii) connect sparsely-connected parts of the graph. To do this quickly using bounded memory, we maintain a fixed-size sample of the edges seen thus far and use it to score the anomalousness of any new edge. Thus, Problem 6.1 can be subdivided into two subproblems, each incorporating one of the above signals of anomalousness, as follows:

Informal Problem 6.2: Edge Sampling

Given an edge stream $\mathfrak{E}$ and $S \in \mathbb{N}$, maintain an online sample $\mathcal{S}$ of S edges so as to downsample bursts of activity.

Algorithm 3 SedanSpot

Input: edge stream $\mathfrak{E} = \{e_i\}_{i=1}^{\infty}$
Output: stream of anomaly scores $\{y_i\}_{i=1}^{\infty}$
▷ *initializations*
SedanSampler.initialize()
SedanScorer.initialize()
for edges $\mathcal{E}_t$ received at time t from stream $\mathfrak{E}$ **do**
    for $e_i \in \mathcal{E}_t$ **do**
▷ *score this edge based on the current sample of edges*
        $y_i \leftarrow$ SedanScorer.anomaly_score(e_i)
▷ *update the current sample of edges using this edge*
        $e_{rem}, e_{add} \leftarrow$ SedanSampler.sample(e_i)
        SedanScorer.add(e_{add}) **if** e_{add} is not None
        SedanScorer.remove(e_{rem}) **if** e_{rem} is not None
        yield y_i

Informal Problem 6.3: Anomaly Scoring

Given a sample of edges $\mathcal{S}$ and a new edge e_i, design an anomaly scoring function $y_i = f(e_i; \mathcal{S})$ so as to give higher score to edges connecting parts of the graph which are sparsely connected.

The proposed SedanSpot consists of two components, each addressing one subproblem above – SedanSampler (Problem 6.2) and (ii) SedanScorer (Problem 6.3).

6.4 Proposed Method

A high-level pseudocode of SedanSpot using the sampling (Section 6.4.1) and scoring (Section 6.4.2) components is given in Algorithm 3. Every edge e_i in the stream is first compared to the current sample of edges via SedanScorer to determine its anomaly score. The sample is subsequently updated based on this edge using SedanSampler. We describe the algorithm below, assuming *directed edges* (extension to other settings is discussed in Section 6.4.3) and postpone analysis to Section 6.5.

6.4.1 Edge Sampling using SedanSampler

Given a sample size $S \in \mathbb{N}$, the main idea behind SedanSampler is to perform *rate-adjusted reservoir sampling* to maintain a *rate-adjusted* sample $\mathcal{S}$ containing S edges.

Definition 6.1: Rate-adjusted sample

$\mathcal{S}$ is said to be a rate-adjusted sample from a stream $\mathfrak{E}$ iff $\mathbf{Pr}\left[e \in \mathcal{S}\right] \propto 1/r(e)\ \forall\ e \in \mathfrak{E}$, where $r(e)$ is the *edge rate* at the time of occurrence of e.

Here, $r(\cdot)$ is a measure of *edge rate* such that a larger value signifies a more intense burst – shorter duration or higher count – of edges (see Equation (6.2)). Intuitively, rate-adjusted reservoir sampling ensures that, if a region R of an underlying graph $\mathfrak{G}$ is densely connected *solely* because of attack edges which occurred during bursts of activity, the corresponding region in the sampled graph induced by $\mathcal{S}$ still remains somewhat sparsely connected. This sets the stage to detect a subsequent edge belonging to same attack – and occurring in the same region R – as an anomaly compared to the sample.

6.4.1.1 Computing Edge Rate $r(\cdot)$

Let $t(e)$ denote the timestamp of edge e and let $\mathcal{E}_{t(e)}$ be the set of edges which arrive at time $t(e)$, including e. A natural choice for edge rate $r(e)$ during the occurrence of edge e is:

$$r(e) = \frac{\left|\mathcal{E}_{t(e)}\right|}{t(e) - t_{LSP}(e)} \tag{6.2}$$

where $t_{LSP}(e) = \max_{e' \text{ s.t. } t(e')<t(e)} t(e')$ denotes the timestamp of the *latest strictly previous* (LSP) edge corresponding to e[1]. Clearly, edges arriving together have identical values of $r(\cdot)$ and the larger the number of edges occurring at $t(e)$ or the smaller the time gap between e and its LSP edge, the more intense is the burst of edges and accordingly, the higher is the value of $r(e)$. While there are other possible characterizations of edge rate, we pick the form in Equation (6.2) as it leads to theoretical guarantees (see Theorem 6.1), depends only the recent history and can be computed in $\mathcal{O}\left(1\right)$ space by maintaining $t_{LSP}(e)$ and also works well in practice.

6.4.1.2 Dynamic Maintenance of Sample

A rate-adjusted sample of size S can be maintained using Weighted Reservoir Sampling (WRS) [ES06]. In WRS, a stream of weighted elements are given and the goal is to maintain a fixed-size reservoir (sample) of elements in a single pass over the stream, such that each element is sampled proportional to its weight. In our case, each edge e constitutes an element, with weight[2] equal to $1/r(e)$. The resulting algorithm, given in Algorithm 4, uses a MinHeap-PriorityQueue data structure for efficient $\mathcal{O}\left(\ln S\right)$ updates. In a nutshell, each edge e is assigned a priority, which is a random number that tends to be lower if $r(e)$ is high and at any point of time, the top S edges having the highest priorities constitute the sample $\mathcal{S}$.

[1]For the first set of edges in the stream, we can define $t_{LSP}(e) := 0$.
[2]This is different from the edge weight w of an edge $e = (u, v, w, t)$.

Algorithm 4 SEDANSAMPLER

Parameter(s): sample size S

procedure INITIALIZE
  $\mathcal{S} \leftarrow$ MinHeap-PriorityQueue(size S)
  ▷ *stores the top S edges with highest priorities; incurs $\mathcal{O}(lnS)$ addition, $\mathcal{O}(1)$ MPE (min. priority element) retrieval costs*
procedure SAMPLE(edge e)
  $x \sim$ Uniform$[0, 1]$
  $p \leftarrow x^{r(e)}$ ▷ *priority of e; $r(e)$ is defined in Equation (6.2)*
  if $\mathcal{S}$.is_full() **then**
    $e', p' \leftarrow H$.peek() ▷ *current MPE and its priority*
    if $p' < p$ **then**
      $\mathcal{S}$.pop() ▷ *remove current MPE, i.e., e'*
      $\mathcal{S}$.insert(edge e with priority p)
      return e', e
    else ▷ *leave sample unchanged*
      return None, None
  else ▷ *heap is not full, simply add e*
    $\mathcal{S}$.insert(edge e with priority p)
    return None, e

6.4.2 Anomaly Scoring via SEDANSCORER

How do we quantify the 'surprise' on seeing an edge e based on the sample $\mathcal{S}$? Intuitively, the more drastic the change in proximity (distance) between u and v after adding edge e to the sample, the more surprising it is. Accordingly, the main idea behind SEDANSCORER is to score the edge anomalousness as $f(e; \mathcal{S}) = \text{MPI}(e; \mathcal{S})$ where MPI$(\cdot)$ is the marginal proximity increase measure, as defined below.

Definition 6.2: Marginal Proximity Increase

The marginal proximity increase measure of edge e from source u to destination v w.r.t. a set of edges $\mathcal{S}$ is given by

$$MPI(e; \mathcal{S}) = s(v \mid u; \mathcal{S} \cup \{e\}) - s(v \mid u; \mathcal{S}) \tag{6.3}$$

Here, $s(v \mid u; \mathcal{S})$ is a measure of *directed vertex proximity* between source u and destination v based on edges $\mathcal{S}$, such that greater the number of shorter, heavily weighted paths from u to v in $\mathcal{S}$, the higher is its value. For reasons we give next, we use the relevance score produced by random walk with restart (Equation (6.1)) as the measure of directed vertex proximity.

Algorithm 5 SEDANSCORER

Parameter(s): restart probability α, number of walks N

procedure INITIALIZE
  $A \leftarrow$ Hash table mapping vertices to their LATs
procedure ADD(edge $e = (u, v, w, t)$)
  $A[u]$.increment(v, w)
procedure REMOVE(edge $e = (u, v, w, t)$)
  $A[u]$.decrement(v, w)
procedure SAMPLE_NEIGHBOR(vertex u_*, edge $e = (u, v, w, t)$)
▷ *samples neighbor of u_* from $\mathcal{S} \cup \{e\}$ based on edge weight*
  if e is None **or** $u_* \neq u$ **then**
    return $A[u_*]$.random_key()
  else
    $W \leftarrow w +$ out-weight of u_* in $\mathcal{S} \cup \{e\}$
    return v w.p. w/W **else** $A[u_*]$.random_key()
procedure VISIT_FRACTION(vertex u, vertex v, edge e)
▷ *outputs an estimator $\hat{s}(v \mid u; \mathcal{S} \cup \{e\})$ for relevance score*
  initialize *num_steps* $\leftarrow 0$, *num_visits* $\leftarrow 0$
  for $i = 1, \ldots, N$ **do**
    walk length $\ell \sim$ Geometric(α)
    num_steps $\leftarrow$ *num_steps* $+ \ell$
    current vertex $a \leftarrow u$
    for $j = 1, \ldots, \ell$ **do**
      num_visits $\leftarrow$ *num_visits* $+ \mathbb{I}(a == v)$
      $a \leftarrow$ SAMPLE_NEIGHBOR(a, e)
      break if a has no outgoing edges in $\mathcal{S} \cup \{e\}$
  return *num_visits*/*num_steps*
procedure ANOMALY_SCORE(edge $e = (u, v, w, t)$)
  visit_frac_before $\leftarrow$ VISIT_FRACTION(u, v, None)
  visit_frac_after $\leftarrow$ VISIT_FRACTION(u, v, e)
  return max(0, *visit_frac_after* $-$ *visit_frac_before*)

6.4.2.1 Why RWR Relevance Score

First, RWR relevance score is holistic as it incorporates direct as well as indirect (k-hop) paths between vertices while computing their proximity; hence, it is robust to noise and subsampling. Second, it is a probability and thus is bounded in $[0, 1]$ even though the edge weights may be arbitrarily large; thus, the resulting anomaly scores are also bounded. Third, relevance score is, in general, not symmetric and can capture situations where an edge $u \to v$ is expected while $v \to u$ is not. Finally, from a practical standpoint, relevance scores can also be estimated fast, using local random walks, which we describe next.

6.4.2.2 Fast Estimation through Local Random Walks

Computing RWR relevance score directly using Equation (6.1) takes $\mathcal{O}(S)$ time per iterative update, which scales linearly with the sample size. Instead, given an application-specific budget N on anomaly scoring time, SedanScorer performs N short, local, random walks starting from source u. The number of times v is visited in this process is used as the estimator $\hat{s}(v \mid u; \mathcal{S})$ for the RWR relevance score. This procedure helps decouple memory budget S from anomaly scoring time and thus, allows us to maintain a larger sample without compromising speed.

Algorithm 5 provides the pseudocode of SedanScorer based on local random walks. It uses the current sample of edges stored in a data structure A, which for now, can be considered as an adjacency list. Given parameters N and α, visit_fraction() outputs $\hat{s}(v \mid u; \mathcal{S} \cup \{e\})$ by performing N local random walks. Each time, a walk length ℓ is sampled based on the restart probability α (line 16). Then, ℓ steps (possibly less if there are dead ends) of random walk starting from source vertex u are taken, each time sampling a neighbor of the current vertex proportional to edge weight in $\mathcal{S} \cup \{e\}$ (line 17-22). The ratio of the number of times v is visited in this process to the total walk length is returned as the estimate $\hat{s}$.

6.4.2.3 Efficient Data Structure

In Algorithm 5, A is any data structure that supports edge addition/deletion and neighbor sampling. We could have naïvely used adjacency list implemented as nested hash tables and incurred $\mathcal{O}(1)$ cost for updates and $\mathcal{O}(d)$ for sampling where d is the vertex degree in the sample (increases with S). However, note that (i) the sampling routine is used significantly more often than updates, (ii) updates tend to become less frequent as the stream progresses[3]. We exploit these observations to speed up by using Alias method [Vos91].

Given an arbitrary discrete distribution with k outcomes, Alias method can produce a sample in $\mathcal{O}(1)$ time by incurring an $\mathcal{O}(k)$ preprocessing cost upfront. Since neighbor sampling is equivalent to sampling from a discrete distribution, we propose to use a *hash table of Lazy Alias Tables* (LATs), one LAT per vertex, as our data structure A. Assuming the LATs are up-to-date, sample_neighbor() takes only $\mathcal{O}(1)$ time. When an edge is added to or removed from the sample, only the LATs of affected vertices have to be updated. Moreover, even these updates can be done in a lazy fashion, i.e., only when we need to sample a neighbor of an affected vertex.

Note that, while computing $\hat{s}(v \mid u; \mathcal{S} \cup \{e\})$ in Equation (6.3), the edge e should *not* actually be inserted into the data structure – this would force unnecessary updates to LATs, incurring a large overhead. See line 12 of Algorithm 5 for how to avoid this. Also, due to randomness in the estimator $\hat{s}$, it is possible (although unlikely) to have *visit_frac_after* $<$ *visit_frac_before* in line 27, hence we return the maximum of this value or zero.

6.4.3 Extensions

We highlight possible extensions to SedanSpot, without detailing them in the interest of space. First, SedanSpot can be easily extended to undirected (by symmetrizing the MPI measure in Equation (6.3)) and bipartite settings (by allowing forward and backward jumps). Next, we can

[3]Unless we bias SedanSampler to sample recent edges; see Section 6.4.3.

easily bias the rate-adjusted sample towards recent edges by modifying line 5 of Algorithm 4 to incorporate edge recency. Finally, in principle, we can sample edges proportional to any monotonically decreasing function of their rate, $f(r(e))$, to downsample bursts. Using $f(x)=1/x$ leads to the guarantee in Theorem 6.1, but other variants may work well depending on the application.

6.5 Theoretical Analysis

Here, we analyze SedanSpot, proving desirable algorithmic (Section 6.5.1) and computational (Section 6.5.2) properties. Relevant proofs are given in the appendix.

6.5.1 Algorithmic Analysis

First, we show that Algorithm 4 is indeed correct: it samples each edge with probability inversely proportional to its edge rate:

Lemma 6.1: Correctness of Algorithm 4

Algorithm 4 maintains a rate-adjusted sample $\mathcal{S}$, as defined in Definition 6.1, from stream $\mathfrak{E}$.

Proof. Follows from Proposition 3 of [ES06]. ■

Importantly, SedanSampler ensures that the number of sampled edges belonging to a given time interval only depends on its duration and not on the number of edges occurring during it. In the following, a time tick τ is said to be anchored if some edge occurred at time τ, i.e., $\exists$ edge e s.t. $\tau=t(e)$.

Theorem 6.1: Burst Resistance

Consider time ticks $\tau_0=0$ and $\tau_1 \leq \tau_2 \ldots \leq \tau_K$ which are anchored. Let $\mathcal{H}_k$ be the set of edges arriving in time interval $I_k := (\tau_{k-1}, \tau_k]$ of duration $\ell_k = \tau_k - \tau_{k-1}$. If $\mathcal{S}$ is the rate-adjusted sample till time τ_K,

$$\mathbf{Pr}\left[e \in \mathcal{H}_k \mid e \in \mathcal{S}\right] = \ell_k / \sum_{k=1}^{K} \ell_k, \ \forall\, k \tag{6.4}$$

which is independent of $|\mathcal{H}_k|$.

Proof. A sampled edge e can belong to $\mathcal{H}_k$ in $|\mathcal{H}_k|$ mutually exclusive ways. Hence, we have $\mathbf{Pr}\left[e \in \mathcal{H}_k \mid e \in \mathcal{S}\right] = \sum_{e' \in \mathcal{H}_k} \mathbf{Pr}\left[e = e' \mid e \in \mathcal{S}\right]$, which can be simplified using Bayes' rule to

$\sum_{e' \in \mathcal{H}_k} \mathbf{Pr}\left[e = e' \wedge e \in \mathcal{S}\right] / \mathbf{Pr}\left[e \in \mathcal{S}\right] \propto \sum_{e' \in \mathcal{H}_k} \mathbf{Pr}\left[e' \in \mathcal{S}\right]$ since $\mathbf{Pr}\left[e \in \mathcal{S}\right]$ is independent of k.

Suppose $\gamma_1 < \gamma_2 \ldots < \gamma_{z_k} = \tau_k$ are the anchored time ticks during interval $I_k := (\tau_{k-1}, \tau_k]$. Let $\gamma_0 = \tau_{k-1}$ and $\mathcal{E}_{\gamma_i}$ be the set of edges occuring at γ_i. Then, $\sum_{e' \in \mathcal{E}_{\gamma_i}} \mathbf{Pr}\left[e' \in \mathcal{S}\right] \propto \sum_{e' \in \mathcal{E}_{\gamma_i}} 1/r(e') = \gamma_i - \gamma_{i-1}$. Thus, $\mathbf{Pr}\left[e \in \mathcal{H}_k \mid e \in \mathcal{S}\right] \propto \sum_{e' \in \mathcal{H}_k} \mathbf{Pr}\left[e' \in \mathcal{S}\right] = \sum_{i=1}^{z_k} \gamma_i - \gamma_{i-1} = \tau_k - \tau_{k-1} = \ell_k$. This proves the theorem. ■

This is advantageous given the tendency of anomalous edges to occur as bursts of activity: even though many edges occur in a small duration, rate-adjusted sampling ensures that only a few of them are stored in the sample, as illustrated below.

Example 6.1

Consider a 'normal' process generating $1M$ edges over 100 hours followed by an attacker producing $0.5M$ in 10 minutes. The expected number of anomalous edges in the sample is $10/(100 \times 60 + 10) < 0.2\%$.

In contrast, the sample $\mathcal{S}'$ produced by Uniform Reservoir sampler (UR-SAMPLER), which samples edges uniformly, i.e., $\mathbf{Pr}\left[e \in \mathcal{S}'\right] \propto 1 \, \forall e$, has $0.5M/1.5M = 33.3\%$ anomalous edges in expectation. The reduced fraction of anomalous edges in the rate-adjusted sample maintained by SEDANSAMPLER translates to a more accurate model of normal behavior, which sets the stage for better anomaly scoring via SEDANSCORER.

Observe from Equation (6.4) that, when the considered intervals are of the same duration, i.e., $\ell_i = \ell \; \forall \; i$, a sampled edge is equally likely to come from any of these intervals. In other words, rate-adjusted sampling ensures equal representation of two time durations of equal length in the sample (in expectation) irrespective of the common length of intervals ℓ, i.e., *simultaneously* at all temporal granularities.

Next, we show that Algorithm 5 returns a principled estimator of the random walk with restart relevance score (Equation (6.1)), in the sense that it is unbiased and consistent, as stated below.

Theorem 6.2

$\hat{s}\left(v \mid u; \mathcal{S}\right)$ from Algorithm 5 is an unbiased and consistent estimator of the RWR relevance score $s\left(v \mid u; \mathcal{S}\right)$.

Recall that RWR relevance score of v w.r.t. u is the steady state probability of being at (i.e., visiting) v while performing random walk with restart from u. Thus, it turns out to be the limiting value of the visit fraction output by Algorithm 5 (as $N \to \infty$). The proof is straightforward.

6.5.2 Time and Space Complexity

SedanSpot satisfies the sublinear memory and constant time per edge requirements of Problem 6.1 as stated below.

Lemma 6.2: Constant Scoring Time per Edge

SedanSpot takes at most $\mathcal{O}(N/\alpha)$ time in expectation (usually lesser in practice) to compute anomaly score of an edge (line 5, Algorithm 3).

Proof. $\mathcal{O}(N/\alpha)$ comes from the N local random walks, each of expected length $\mathbb{E}[\text{Geometric}(\alpha)] = 1/\alpha$ (lines 16-22 of Algorithm 5). Each step of random walk takes $\mathcal{O}(1)$ due to constant-time neighbor sampling via LAT. ■

Lemma 6.3: Sublinear Memory Requirement

SedanSpot takes $\mathcal{O}(S\ln|\mathcal{V}|)$ memory to process each edge.

Proof. The $\mathcal{O}(\ln|\mathcal{V}|)$ is a lower bound on the memory requirement, since each edge (including vertex IDs) needs to be read. Thus, $\mathcal{O}(S\ln|\mathcal{V}|)$ space is needed to store LAT data structures over the sample of S edges. ■

In addition, we can show that updates of SedanSpot are fast, especially amortized over the length of stream L, as the updates become less frequent as the stream progresses. In the following, let d_{avg} be an upper bound on the average vertex degree in the sample and let $H_n = \sum_{i=1}^{n} i^{-1}$ denote the sum of first n terms in the harmonic series. Further, let the edge rate be bounded in $[r_{\min}, r_{\max}]$. Then,

Lemma 6.4: Fast Amortized Updates per Edge

SedanSpot takes at most $\mathcal{O}\left(\frac{\ln S + d_{avg}}{L} \cdot \left(S + \frac{r_{\min}}{r_{\max}}(H_L - H_S)\right)\right)$ amortized time in expectation for updates (lines 6-8, Algorithm 3).

Proof. For $i > S$, the probability of sampling the i^{th} edge e_i in the stream is given by $p_i = r(e_i)^{-1} / \sum_{j=1}^{i} r(e_j)^{-1} \leq r_{\max}/(i \cdot r_{\min})$. The expected number of updates is $S + \sum_{S+1}^{L} p_i \leq S + r_{\max}/r_{\min} \sum_{S+1}^{L} 1/i$, each update costing $\mathcal{O}(\ln S)$ for sampling and $\mathcal{O}\left(d_{avg}\right)$ for scoring in expectation (assuming no correlation between edge rate and the degrees of incident vertices). Amortization completes the proof. ■

Table 6.2: Precision of SedanSpot and baseline (RHSS) at different cut-off ranks k. Bold signifies highest value in each column and underline shows significant differences (p-value ≤ 0.01) w.r.t. baseline according to a two-sided micro-sign test [YL99].

Method	precision@								
	200K	400K	600K	800K	1000K	1200K	1400K	1600K	1800K
SedanSpot	**1.00**	**0.97**	**0.93**	**0.89**	**0.85**	**0.83**	**0.81**	**0.80**	**0.79**
RHSS	0.49	0.36	0.29	0.29	0.32	0.35	0.36	0.36	0.33

As $H_n = \ln n + \gamma + \mathcal{O}\left(\frac{1}{n}\right)$ where γ is the Euler-Mascheroni constant, the amortized update time per edge remains small.

6.6 Experiments

We empirically evaluate the proposed SedanSpot on datasets where the anomalies are verifiable and/or interpretable. We first describe datasets and experimental setup.

6.6.1 Datasets

We shortlist three publicly-available real-world time-evolving graph datasets where the anomalies can be verified by comparing either against manually obtained ground truth or publicly-available information. These are:

Darpa [LCF+99] dataset consists of network traffic from 9484 source IPs to 23398 destination IPs over $87.7K$ minutes. There are $\sim 4.5M$ directed *<srcIP, dstIP, 1, time>* edges in total, of which 60% are manually annotated as anomalous. They correspond to 89 network attacks – such as denial of service or port scan – injected by domain experts. Despite this high proportion, the attacks themselves occurred infrequently (but as bursts of activity) and originated from a mix of IP addresses which either were solely dedicated to attacks, or more challengingly, attempted camouflage by participating in normal traffic. This makes Darpa dataset the perfect testbed for SedanSpot, which aims to detect precisely such anomalous bursts occurring in sparse regions of the graph.

Enron [SA04] dataset consists of e-mail communications among the 151 employees of Enron company from May 1999 to April 2002, a period of three years surrounding the famous Enron scandal. There are $\sim 50K$ directed *<sender, receiver, 1, date>* edges. Since ground truth is not directly available, we verify anomalies by correlating their time stamps with real-world events. We expect more edges to be flagged anomalous during periods of large internal (e.g., new CEO) or external (e.g., updates on lawsuit) changes which create (or result from) excitement or turbulence among the employees.

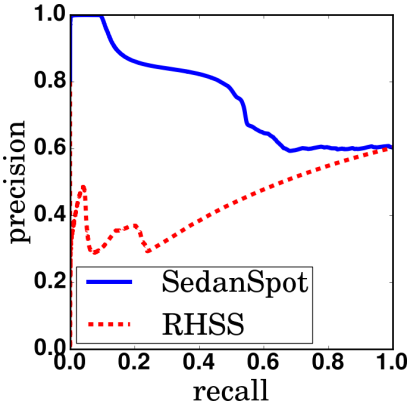


Figure 6.3: SedanSpot achieves better recall and precision on Darpa dataset for all cut-off ranks k.

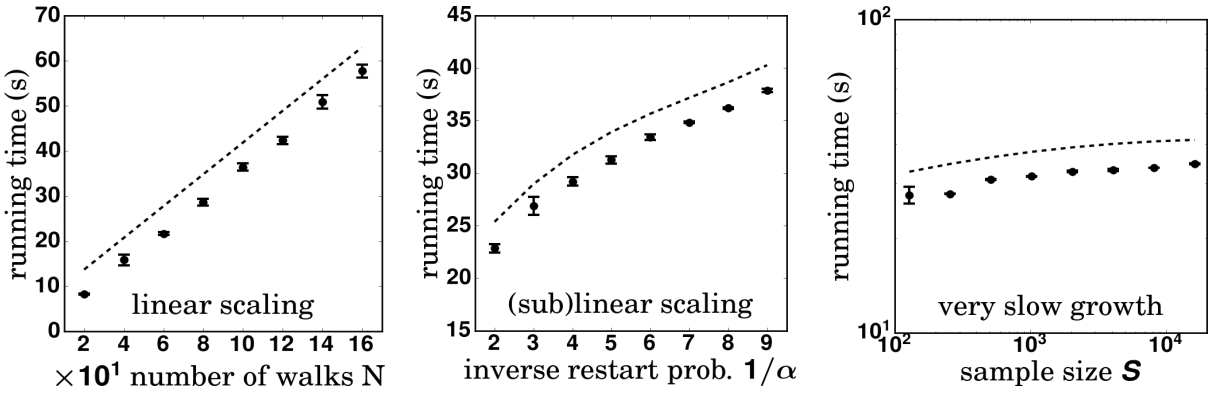


Figure 6.4: SedanSpot (a) scales linearly with the number of walks N, (b) scales sub-linearly with $1/\alpha$, the inverse of random walk restart probability and (c) grows very slowly with the sample size S.

Dblp [dbl14] is the collaboration network of authors of papers from DBLP computer science bibliography. Each undirected edge *<auth1, auth2, 1, pub_year>* between two authors represents a joint publication. For simplicity, we only consider papers published in 1991-2010 and retain nodes (authors) who have at least 50 edges, filtering out the remaining nodes (and corresponding edges). This resulted in a graph containing around $55.5K$ authors and $3.7M$ coauthorships. We expect anomalous edges to represent unlikely collaborations, e.g., authors from unrelated fields or different geographical regions. We verify anomalies using the public profiles of the authors.

6.6.2 Experimental Setup

We implement both SedanSpot and the baseline in C++ and run experiments on MacOS High Sierra with 2.7 GHz Intel Core i5 processor and 16 GB main memory.

Baseline: While there are two edge stream anomaly detection approaches in the literature – GOutlier [AZY11] and RHSS [RHSS16] – we use only RHSS, which extends to weighted and directed edges, as baseline. We extend the original algorithm (for undirected graphs; see Section 6.2) to directed setting by maintaining separate sketches for in- and out- neighborhoods of vertices. Using the recommended parameters $\delta{=}\epsilon{=}0.001$, we ended up with CountMin sketches of width $w{=}2719$ and depth $d{=}7$. We use the negative likelihood of edge occurrence probabilities as anomaly scores, giving equal importance to sample, preferential attachment and homophily scores.

Evaluation Metrics: All the methods output an anomaly score per edge (higher is more anomalous). Sorting the edges in descending order of their scores, we count the number of edges c_k flagged correctly as anomalous among the top k edges, for every cut-off rank $k \in \mathbb{N}$. If C is the total number of ground truth anomalies, we compute: *precision@k* $= c_k/k$ and *recall@k* $= c_k/C$. We also summarize the overall accuracy using the AUC (Area under ROC Curve) measure. All the above metrics lie in $[0, 1]$ and the higher, the better. We also note the running time (excluding IO) averaged over five runs.

Experimental Design: We design our experiments to answer: **[Q1] Accuracy:** How well does SedanSpot detect anomalies compared to baselines? What is the trade-off w.r.t. running time? How does the accuracy vary with parameters S, N and α? **[Q2] Scalability:** How does the running time of SedanSpot scale with input stream length L? What is the effect of parameters N, α and S? **[Q3] Discoveries:** Does SedanSpot lead to interesting discoveries in practice?

We now detail our experimental findings.

6.6.3 Q1. Accuracy

Table 6.2, Figure 6.2(a) and Figure 6.3 show precision, recall, accuracy and running time of all methods on the labeled Darpa dataset, with $N{=}100$ walks, $\alpha{=}0.15$ restart probability (the recommended value [LM04]) and $S{=}10K$ sample size.

Precision and Recall: Table 6.2 tabulates the precision@k of SedanSpot and RHSS at nine cut-off ranks $k \in [20K, 18K]$. Despite the use of randomization, the standard deviations in results (shown in brackets) were low ($\leq$0.02) indicating a fairly consistent performance across multiple runs. We see that SedanSpot outperforms RHSS on all considered k values, achieving $100{-}215\%$ (statistically significant) improvements in precision. Further, a plot of precision vs. recall for all cut-off ranks (Figure 6.3) shows that SedanSpot (solid blue) lies completely above the baseline (dashed red), indicating that the performance gains generalize to all cut-off ranks k.

Accuracy vs. Running Time: Figure 6.2(a) plots the average accuracy (AUC) of each method vs. its average running time (in minutes), over five runs. Error bars are omitted in this plot as the standard deviations turned out to be very low. We see that SedanSpot achieves a much higher

accuracy (=0.63) compared to the baseline (=0.17), while also running the faster (8 mins vs. 24 mins for RHSS). This corresponds to a 270% accuracy improvement in $3\times$ less processing time. We find that the main overhead of RHSS is due to computation using pairwise independent hash functions.

Accuracy vs. Parameters: We summarize our observations regarding the variation of accuracy with parameters on DARPA dataset below, omitting figures in the interest of space. First, the accuracy was robust (~ 0.635) to the restart probability $\alpha \in [0.5, 0.11]$. This is consistent with the trend observed for page rank (closely related to random walks), where α has little effect on the top ranking webpages based on their page ranks [LM04]. Next, accuracy exhibited a 'diminishing return' behavior as sample size S was varied in $[6K, 20K]$. As SEDANSPOT stores more edges, it better models normal behavior and thus accuracy increases. But the marginal increase itself decreases: once the normal behavior is captured sufficiently well in the sample, subsequent increase in S leads to little improvement. Finally, somewhat surprisingly, increasing the number of walks N did not necessarily lead to higher accuracy. In fact, the accuracy peaks around N=10 and then gradually starts decreasing. A similar pattern is observed for tasks such as link prediction, where estimates of RWR relevance scores based on a few walks often outperform their steady state values [LL10].

6.6.4 Q2. Scalability

Figure 6.2(b) and Figure 6.4 show how SEDANSPOT scales with the number of edges in the stream (or stream length) L, number of walks N, random walk restart probability α and sample size S. By default, we use L=$0.5M$, S=$10K$, N=100 and α=0.15. Running times are averaged over five runs and error bars indicate standard deviations.

With Stream Length L: We vary the number of edges L in the input stream in eight logarithmic steps from $20K$ to $2.56M$. Figure 6.2b, plotting the variation of running time in log-log scales, reveals a line of slope 1.0. This confirms the linear scalability of SEDANSPOT w.r.t. input stream length, thanks to its constant processing time per edge. Observe that SEDANSPOT processes $2.56M$ edges in around 4 minutes and thus is very fast (a speed of about $10.1K$ edges per second).

With #Random Walks N: Figure 6.4(a) plots the running time of SEDANSPOT against the number of random walks N, as it is varied in $[20, 160]$. The points align well on a straight line, indicating that SEDANSPOT scales linearly with N.

With Restart Probability α: Figure 6.4(b) plots the running time of SEDANSPOT against $1/\alpha$, where α, the restart probability, is varied in $[1/2, 1/9]$. The curve begins to flatten out for high values of $1/\alpha$, suggesting that the running time scales sublinearly with $1/\alpha$. This is natural given the finite sample of edges that SEDANSPOT maintains: even though the expected length of walks increases linearly with $1/\alpha$, many of these walks terminate early, resulting in a sublinear dependence.

Table 6.3: Contributions of SedanSampler and SedanScorer to AUC gains

Sampler	Scorer	AUC
No sampling	RHSS	0.17
UR-Sampler	RHSS	0.17
SedanSampler	RHSS	0.45
UR-Sampler	SedanScorer	0.57
SedanSampler	**SedanScorer**	**0.63**

With Sample Size S: Figure 6.4(c) shows how running time varies with sample size S in $[128, 16384]$ in log-log scales. The running time was small for low S due to premature termination of local random walks in a small sample of edges (many vertices did not have any outgoing edges). However, the curve begins to flatten out towards the end, indicating that the running time grows very slowly with sample size.

These findings are consistent with our analysis in Section 6.5.2.

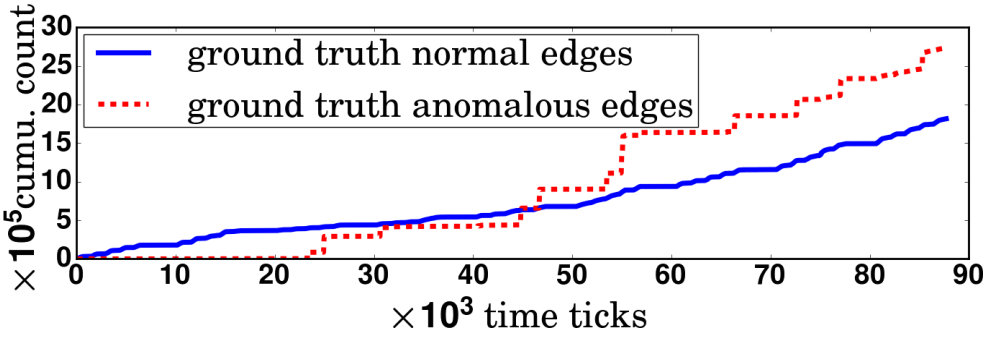
6.6.5 Q3. Discoveries

We provide a detailed analysis of SedanSpot and baseline on the labeled Darpa dataset; in the interest of space, we only focus on discoveries due to SedanSpot on the other datasets.

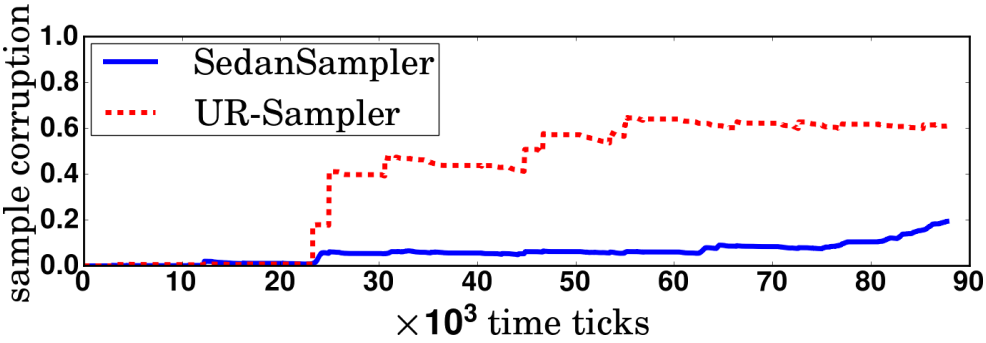
6.6.5.1 Darpa

Leveraging ground truth, we aim to understand why SedanSpot significantly outperforms RHSS on Darpa dataset. Recall that SedanSpot improves the state-of-the-art RHSS in two ways: (i) rate-adjusted sampling and (ii) holistic scoring. To understand their relative contributions to the overall accuracy gains (Figure 6.3), we consider the three intermediate versions of algorithms, combining UR-Sampler (based on Uniform Reservoir sampling) or SedanSampler with RHSS (which originally maintains counts over the *whole* stream, with 'no sampling'), and also using UR-Sampler with SedanScorer. Figure 6.5(a) summarizes the results.

We see that using RHSS as scorer gives the same AUC with or without Uniform Reservoir sampling. This is because UR-Sampler, selecting each edge with equal probability, 'preserves' the fraction of anomalous edges while sampling, thus leading to similar results. Switching to SedanSampler improves AUC by 165%; we explain this with the help of Figure 6.5(b) and Figure 6.5(c). Figure 6.5(b), plotting the cumulative count of normal (non-attack, solid blue) and anomalous (attack, dashed red) edges over time, contrasts the smooth increase of normal edges to the step-like behavior of red curve which results from the bursty nature of network attacks. SedanSampler exploits this by performing rate-adjusted sampling which downsamples edges from bursty periods of time and thus significantly decreases the fraction of anomalous edges ('corruption') in the sample, as shown in solid blue curve of Figure 6.5(c) (while the dashed red curve of UR-Sampler stabilizes around 0.6, which is exactly the fraction of anomalous edges in this dataset). The decreased 'sample corruption' finally paves the way to better anomaly scoring based on the sample.



(a) Steps in red curve show ground truth anomalous edges occur in bursts.



(b) SedanSampler achieves lower sample 'corruption' by rate-adjusting.

Figure 6.5: Anomaly detection in Darpa dataset

Table 6.4: SedanSpot detects network attacks with high precision

Name	% in top $0.2M$	Description
`smurf/ smurfttl`	82.45%	DoS attack using ICMP echo requests through intermediaries
`neptune/ nep- tunettl`	13.06%	SYN flood due to excess partially-open connections (overflow)
`satan`	2.36%	Gathers info on hosts by checking vulnerabilities like finger, ftp, nfs
`ipsweep/ portsweep`	0.74%	surveillance sweep to determine hosts/ports listening on a network
`teardrop`	0.45%	DoS attack due to improper handling of overlapping IP fragments
`apache2/ back`	0.27%	DoS attack against Apache web server where requests have many http headers or front slashes
others	0.58%	e.g., `warez`, `rootkit`, `nmap`
normal	0.09%	incorrectly flagged normal traffic

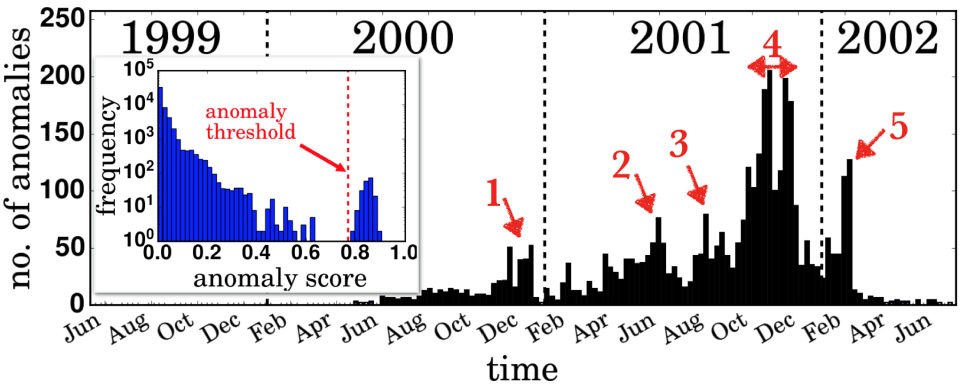


Figure 6.6: Anomaly detection in ENRON dataset

Using SEDANSCORER over the baseline RHSS scoring function always helps, regardless of the sampling algorithm employed, as seen from the improved accuracy values in Figure 6.5(a) ($0.17 \rightarrow 0.57$ with UR-SAMPLER, $0.45 \rightarrow 0.63$ with SEDANSAMPLER). We attribute this to the holistic edge anomaly scoring by SEDANSCORER based on the *whole* (sampled) graph, as opposed to RHSS which considers only the local neighborhood of the edge. Overall, using SEDANSCORER with SEDANSAMPLER (i.e., SEDANSPOT) performs the best.

Figure 6.5(d) lists the network attacks detected by SEDANSPOT among the top $0.2M$ anomalous edges. As shown, most flagged edges belonged to `smurf`, `neptune`, `satan` or `ipsweep` attacks, while less than 1% were false positives. In contrast, among the top $0.2M$ anomalous edges detected by RHSS, 38.6% were `smurf`, 4.5% were `ipsweep` and 3.6% belonged to other attacks. Notably, over 53.3% of flagged edges were false positives, which is unacceptable in many applications, especially network intrusion detection [SP10].

Somewhat surprisingly, RHSS did not detect *any* edge from `neptune` attack – the largest attack in DARPA dataset, consisting of $2.1M$ edges ($=46\%$ of total) and recurring multiple times – as an anomaly. However, this is easily explained: following the first occurrence of `neptune` attack, RHSS increments corresponding edge counts and vertex degrees; thus subsequent occurrences of `neptune` edges, which now have been observed before and connect high degree vertices, are flagged non-anomalous. On the other hand, the rate-adjusted sampling procedure of SEDANSPOT gives very low priority to `neptune` edges, which not only decreases their probability of being included in the sample, but also ensures that they are easily replaced even if they have been sampled, as stream progresses. Thus, SEDANSPOT successfully detects 13% `neptune` edges among its top $0.2M$.

6.6.5.2 ENRON

Figure 6.6 depicts the anomaly detection results on ENRON dataset using $N=50$ walks, $\alpha=0.15$ restart probability and $S=2K$ edges in sample. The inset, plotting the distribution of anomaly scores, showcases a large separation in scores for anomalies and non-anomalies, which is de-

sirable. Accordingly, we use the threshold marked in red (=0.76) to flag anomalies[4]. The rest of the figure plots the temporal distribution of flagged edges (aggregated weekly), which we verify by correlating with the publicly-available Enron time line[5].

The red arrows in Figure 6.6 mark the top five non-contiguous periods of time having the highest number of anomalous edges. As expected, these periods coincide well with notable events surrounding the Enron scandal, creating a flood of unusual e-mails from (even low-level) employees: (1) Dec 2000: Skilling announced as CEO. (2) Jun 2001: The California energy crisis ends. (3) Aug 2001: Skilling announces resignation. Lay named CEO. (4) Oct-Nov 2001: Fastow ousted. A formal investigation against Enron is launched. Stocks crumble. Enron files for bankruptcy. (5) Jan-Feb 2002: Cooper takes over as CEO after Lay resigns. Fastow, Kopper, Skilling and Watson (whistle-blower) testify before Congress.

6.6.5.3 Dblp

We run SedanSpot on Dblp with S=200K, α=0.15 and N=1000. As shown below with the help of anecdotal evidence, the top anomalous edges indeed represent unexpected or unlikely collaborations:

- *Alex Galis, Robert Szabo (2004):* This is due a joint invited paper at an IEEE MATA 2004 workshop, which marked the beginning of an unexpected collaboration between authors of different countries, namely, Galis from Univ. College London (UK) and Szabo from Budapest Univ. of Technology and Economics (Hungary).
- *Nikol Rummel, Nikolaos Avouris (2007):* This is the result of an interdisciplinary paper about Computer Supported Collaborative Learning, requiring collaboration between authors belonging to different fields (Psychology, ECE).
- *Ryan Thibodeau, Mark Carrington (2010):* This is the product of a rare massive collaboration effort among 44 authors across seven institutions, including Thibodeau from Univ. of Georgia, USA and Carrington from Univ. of Cambridge, UK. This marks their only joint publication.

In summary, our experiments demonstrate that SedanSpot outperforms state-of-the-art in terms of both speed and accuracy. The detected anomalies also carry practical significance, e.g., as security threats in network traffic.

6.7 Conclusion

We considered the problem of near real-time anomaly detection given a stream of edges from a/an (un)weighted (un)directed time-evolving graph, where anomalies are edges occurring as spikes or bursts of activity and connecting parts of graph which are otherwise disconnected. SedanSpot exploited these observations in sublinear memory by (i) performing rate-adjusted sampling which downsamples edges from bursty periods of time and (ii) using a holistic random

[4]In practice, one can use the median $\hat{\mu}$ and inter-quantile range $\hat{\sigma}$ of past scores to flag anomalies, *in an online manner*, when the score exceeds $\hat{\mu} + 3\hat{\sigma}$.

[5]http://www.agsm.edu.au/bobm/teaching/BE/Enron/timeline.html

walk based edge anomaly scoring function to compare an incoming edge with the whole (sampled) graph. Experiments on real-world datasets demonstrated the usefulness of our anomaly definition and efficacy of the proposed approach in several scenarios. Future work could explore detecting other kinds of anomalies, e.g., low temperature or periodic attacks, under the computational constraints of the streaming edge setting.

Chapter 7

SmokeAlarm: Early Warning of User-Input Anomalies

Chapter based on work that appeared at ICDM 2019 [EFMN19] [PDF].

How do can early warn against an impending student drop out or an adverse health condition in near real-time? How can we leverage recent interventions such as tutoring or medicines to early warn more accurately? More challengingly, how do we learn to early warn from data that is peppered with such interventions? Early warnings are pivotal for avoiding long-term problems in healthcare, education, mechanical failures, cloud and disaster management. To effectively aid human decision making in these high-stakes contexts, interpretability of the method producing warnings is a key concern.

We consider the problem of learning to interpretably early warn from labeled data tainted by interventions. Our contributions are: *(1) Principles:* We lay out three characteristics–*dominance*, *precedence* and *intervention-awareness*–of an ideal early warning system. *(2) Algorithm:* In line with these, we propose SmokeAlarm which learns from past labeled data containing interventions offline and can produce early warnings online. *(3) Interpretability:* SmokeAlarm learns state-based progression models in the presence and absence of interventions, which are "bi-inspectable" by the human decision maker. Extensive experiments on synthetic and real-world data demonstrate that SmokeAlarm outperforms baselines (by $16-38\%$ in terms of AUC, with an average lead time of 6.1 hours before the onset of septic shock), while scaling linearly with data size and also leading to intuitive, interesting discoveries in practice.

7.1 Introduction

Early warnings have serious implications in a variety of domains. In medicine, warning of a forthcoming epileptic seizure or septic shock event can be life-altering. In data centers, warning of a pending server crash can alert cloud system administrators of downstream problems. Interventions can come to the rescue – particularly if they are delivered at an opportune mo-

ment. Pulling over to the side of the road before a seizure can prevent a car accident; antibiotics can avert organ failure; counseling a student may prevent their drop out.

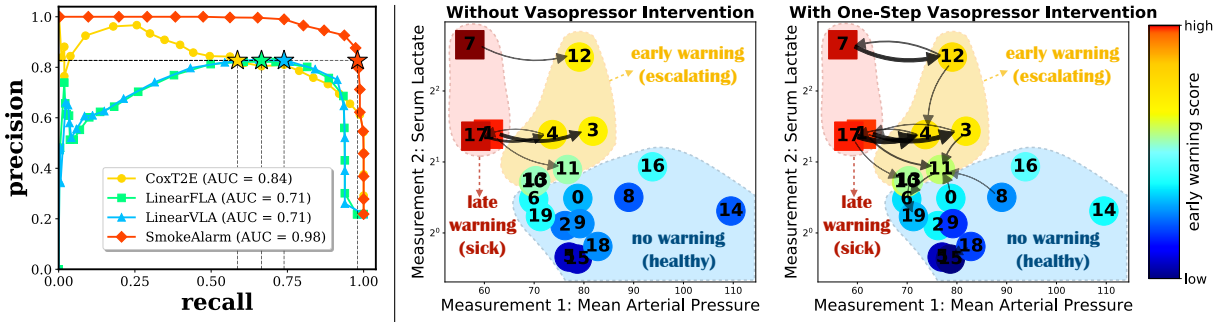
Consider how machine learning (ML) algorithms trained on historical data can automatically early warn. The classical solution is to train an algorithm with labels "fast-forwarded" in time. For instance, if the goal is to predict system failure k time steps in advance, then label a time step t with what happened at $t+k$. However, as noted by [PNMS13], if interventions administered in the intervening duration averted an event, the observed labels underestimate the true early warning score. If the intervention aids an event, labels may overestimate the score. Thus, if interventions are not handled correctly, counter-intuitive results may ensue, e.g., learning that asthma decreases the risk of pneumonia [CLG+15], when it was the aggressive care that asthma patients received that improved outcomes.

How can an ML algorithm account for interventions? [DS16] presents a compelling solution: physicians pairwise compare patient trajectories and label which time point has a higher severity score. A model trained to maximally agree with these pairwise labels then forms the basis for early warning. However, asking physicians to label trajectories ex post facto is laborious and will not scale. Moreover, doctors may themselves be tainted by interventions: a condition that can be treated effectively may be perceived less severe, and thus, may artificially deflate the early warning score. To cope with interventions, [DS16] clips off the suffix of a time series after the first event-related intervention is administered. However, following an intervention, there is still an abundance of data for training. In some datasets, e.g., from ICUs, the preponderance of data may be post-intervention. Further, even after an event-related intervention, early warnings continue to be beneficial since the first intervention may not be effective. We consider how past data, peppered with interventions, can be used to learn *intervention-aware* early warning scores.

To effectively aid human decision making in high-stakes domains such as healthcare, interpretability of the warnings is a key concern. It is important for validating the model pre-deployment and also for providing an evidence-based, human understandable explanation for its score. As pointed out by [Lip18], interpretability is not a monolithic concept; in this work, we seek an interpretable model which "can be readily presented to the user with visual or textual artifacts" [RSG16]. See Figure 7.1(b).

The approach we take in this chapter is to first learn a function that predicts the probability of a future event under various future intervention regimens and then suitably time-decay this function to produce an early warning score. Specifically, we take the *stochastic* and *prolonged* effect of interventions into account to explicitly model the evolution of trajectories in the presence and absence of their influence. Concretely, our contributions are:

1. **Principles:** We state three intuitive characteristics of an ideal early warning system: producing high early warning scores when a future event is more likely *(dominance)*, or is expected to occur sooner *(precedence)* and not presupposing that a specific future intervention will be administered *(intervention-awareness)*. See Section 7.3.
2. **Algorithm:** We propose SmokeAlarm for learning to early warn from past labeled data tainted by interventions. SmokeAlarm learns offline, produces early warning scores online by 'watching' an evolving trajectory, and provably obeys all three principles. See Section 7.4.



(a) Precision, Recall, AUC

(b) Inferred SmokeAlarm Model

Figure 7.1: SmokeAlarm is accurate and interpretable: (a) SmokeAlarm outperforms baselines on all metrics. (b) SmokeAlarm model with states (numbered vertices) and early warning scores (vertex colors) shows that the method alarms when the patient already has septic shock (late warning; red region), and does not warn when they are healthy (green region). Importantly, it warns during the escalation to septic shock (orange area), which is the opportune moment for intervention. SmokeAlarm also learns that *vasopressor* interventions tend to increase MAP as indicated by dark, thick rightward arrows in (b)-right; this results in decreased early warning scores of low-MAP states $\{1, 7, 17\}$.

3. **Interpretability:** A key novelty of SmokeAlarm is its "bi-inspectability", which is to say that the model can be visualized both in the presence and in the absence of an intervention. The probability of transitions between states can be compared depending on whether the intervention was administered. For example, in Figure 7.1b, we see that a vasopressor intervention is more likely to increase blood pressure (arrows are bolder pointing to the right). Also, the early warning score can be compared with and without an intervention: the deep red color of a sick state turns to a lighter red in the presence of an intervention.

A comprehensive battery of experiments is presented in Section 7.5. Synthetic settings are valuable because we can simulate what happens when sick patients are left untreated, without incurring the associated human cost. Experiments on real-world ICU data reveal that SmokeAlarm is better able to early warn of septic shock than past approaches, outperforming by $16 - 38\%$ in terms of AUC (see Figure 7.1(a)) with an average lead time of 6.1 hours before the onset of septic shock.

We primarily use examples from a healthcare setting to motivate and explain our ideas. However, we note that the ideas and algorithm presented in the chapter can be more broadly applied, e.g., early warning against student drop out with tutoring as interventions, early warning against mechanical failures with part replacements or repairs as interventions, etc.

Table 7.1: Comparison of Smoke Alarm to Prior Work on Early Warning.

Desiderata \ Method	Linear Violates All Principles [NHR+18, HHPS15]	Linear Violates Some Principles [DS16]	Non-Linear [FHH+17] [CSSS17] [?]	This work
P1. Dominance				✔
P2. Precedence				✔
P3. Intervention-Aware		✔		✔
Interpretable Model	✔	✔		✔

7.2 Related Work

Time series data has been well-studied in the data mining community for event detection [WH98], anomaly detection [KLFH06], similarity search [YZU+18], visualization [GLL+17] and more. [RLG+05] provides a comprehensive treatment. Here, we only survey work most relevant to intervention-aware and interpretable early warning.

Early Prediction: Prior work has considered early warning against various adverse events including sepsis [FHH+17, NHR+18], septic shock [HHPS15], heart failure [CSSS17], etc. In general, these approaches do not account for confounding interventions which "can mask the ground truth labels needed to train and evaluate a prediction system" [PNMS13], as we described earlier in Section 7.1.

Handling Interventions: To cope with the counter-intuitive results due to intervention confounding, [CLG+15] advocates for *intelligible* models amenable to repairing by domain experts, e.g., by deleting incorrect rules such as asthma reduces the risk of pneumonia. [DS16] proposes a human-in-the-loop solution by seeking expert labels for pairwise comparisons of time points. In contrast, we focus on solutions requiring minimal human labeling and post-processing efforts. More recently, [SS17] uses Counterfactual Gaussian Processes to forecast a single real-valued measurement in the presence of interventions. As such [SS17] does not address the early warning problem and scales poorly with input size due to the use of Gaussian Processes.

Progression Modeling: State-based methods can interpretably model the progression of trajectories. [WSW14] infers a continuous-time Markov model to stitch together partial disease trajectories into a global progression model for chronic obstructive pulmonary disease. [YML+14] learns a probabilistic model to estimate the stages of chronic kidney disease, among other applications. Both methods are unsupervised, ignore interventions and do not early warn. We also employ a state-based model, but explicitly account for interventions, illustrate bi-inspectability, and capitalize on labels to early warn.

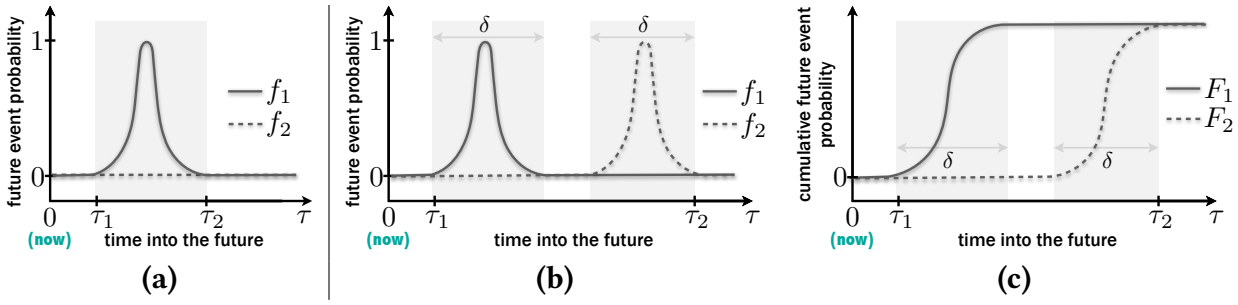


Figure 7.2: Principle of dominance: (a) The trajectory $\mathcal{T}_1$ having a higher likelihood of event in the future and hence a dominating future event probability function f_1 (solid line) should have a higher warning score. Principle of precedence: (b) The trajectory $\mathcal{T}_1$ expected to have an event sooner and hence a dominating cumulative future event probability function F_1 in (c) should have a higher warning score.

Other Related Work: [SBB+09] uses differential equations to model the dynamics of a system and then exploits regime shifts such as a critical slowing down to early warn. As such, they do not learn from data. Reinforcement learning [SB18] can be used to learn optimal intervention policies, e.g., for sepsis [RKC+17]. However, our focus is on a related but different problem of producing credible early warning scores to aid the human decision maker.

As summarized in Table 7.1, SmokeAlarm compares favorably to prior approaches which early warn: the scores are learned with minimal human effort by an interpretable model which follows all principles of an ideal early warning system.

7.3 Preliminaries and Principles

Let $\mathbb{T}$ be a set of trajectories where each trajectory $\mathcal{T} \in \mathbb{T}$ consists of measurements $\mathbf{x}_t$, intervention $\mathbf{y}_t$ and event label ℓ_t observed at discrete time steps $t{=}1, 2, \ldots, T$. Each measurement is a vector $\mathbf{x}_t{=}(x_{t1}, \ldots, x_{tM})$ of recorded values for M measurement types, each taking either real (e.g., temperature) or categorical values (e.g., eye color). Each label ℓ_t is binary with 1 denoting an event occurrence. The values for some measurements and labels may be missing, but the interventions are assumed to be fully observed. We consider a single intervention type (e.g., aspirin) and let each $\mathbf{y}_t \in \{0, 1, \ldots, Y\}$ denote the quantity of intervention given at t where Y is the maximum permissible dose. Handling multiple intervention types is left to future work.

Denote using $\mathbf{x}_{:t}$ and $\mathbf{y}_{:t}$ the measurements and interventions observed upto (and including) time t. Similarly, let $\mathbf{y}_{t+1:}$ denote the interventions after time t, i.e., $t{+}1$ onward. Given a trajectory $\mathcal{T}{=}(\mathbf{x}_{:t}, \mathbf{y}_{:t})$ observed until time t, our goal in this work is to produce an early warning score $w(\mathcal{T}) = w(\mathbf{x}_{:t}, \mathbf{y}_{:t})$ as a scalar reflecting how soon and likely an event occurrence is, when not disturbed by future interventions. The rest of this section formalizes the above intuition as principles. To do so, we will need the following definitions:

Definition 7.1: Intervention-Free Future

A trajectory $\mathcal{T} = (\mathbf{x}_{:t}, \mathbf{y}_{:t})$ is said to have an intervention-free future iff no intervention is given after time t, i.e., $\mathbf{y}_{t+1:} = \mathbf{0}$.

Definition 7.2: Future Event Probability Function

A future event probability function $f{:}\mathbb{Z}_{\geq 0} \rightarrow [0, 1]$ maps a given $\tau{\geq}0$ to the probability of an event τ steps into the future.

Definition 7.3: Dominance and Strict Dominance

For functions $g_1, g_2 : \mathcal{D} \rightarrow \mathbb{R}$ where $\mathcal{D}$ is a countable set, g_1 is said to dominate g_2 iff $g_1(z) \geq g_2(z)\ \forall\ z \in \mathcal{D}$ and is denoted as $g_1 \geq g_2$. Further, g_1 is said to strictly dominate g_2 iff $g_1 \geq g_2$ and $\exists\ \mathcal{D}' \subseteq \mathcal{D}$ for which $g_1(z) > g_2(z)\ \forall\ z \in \mathcal{D}'$. Strict dominance is denoted as $g_1 > g_2$.

7.3.1 Principles of an Ideal Early Warning System

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be a pair of trajectories with intervention-free futures and future event probability functions f_1 and f_2 respectively. Define cumulative future event probability functions $F_1(\tau) = \sum_{\tau'=0}^{\tau} f_1(\tau')$ and $F_2(\tau) = \sum_{\tau'=0}^{\tau} f_2(\tau')$.

Problem 7.1 states that the greater the chances of an event in the future, the higher the early warning score should be.

Principle 7.1: Dominance

Given a pair of trajectories with intervention-free futures, an ideal early warning system gives a (strictly) higher score to the trajectory having a (strictly) dominating future event probability function.

$$f_1 \geq f_2 \implies w(\mathcal{T}_1) \geq w(\mathcal{T}_2); f_1 > f_2 \implies w(\mathcal{T}_1) > w(\mathcal{T}_2) \tag{7.1}$$

The next principle states that the sooner an event is expected in the future, the higher should be the early warning score.

Principle 7.2: Precedence

Given a pair of trajectories with intervention-free futures, an ideal early warning system gives a (strictly) higher score to the trajectory having a (strictly) dominating cumulative future event probability function.

$$F_1 \geq F_2 \implies w(\mathcal{T}_1) \geq w(\mathcal{T}_2); F_1 > F_2 \implies w(\mathcal{T}_1) > w(\mathcal{T}_2) \tag{7.2}$$

Figure 7.2 shows trajectories $\mathcal{T}_1$ and $\mathcal{T}_2$ with future event probability functions f_1 (solid) and f_2 (dashed) differing only in $[\tau_1, \tau_2]$. In Figure 7.2a, f_1 strictly dominates f_2 in $[\tau_1, \tau_2]$, encoding a greater event likelihood. In Figure 7.2b, neither curve dominates the other; rather, the event probabilities are swapped in time, with f_1 predicting an event sooner than f_2. In both cases, the aforementioned principles require $\mathcal{T}_1$ to have a strictly higher early warning score. Observe how the temporal precedence relation between f_1 and f_2 is captured by their cumulative future event probability functions F_1 and F_2 (Figure 7.2c). Also, note that dominance implies precedence, but not vice versa.

The last principle addresses how an ideal early warning system should consider interventions.

Principle 7.3: Intervention-Awareness

The score given by an ideal early warning system is independent of any anticipated future interventions. An ideal early warning system presupposes that no interventions will be given in the future.

$$w(\mathbf{x}_{:t}, \mathbf{y}_{:t}) = w(\mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:} = \mathbf{0}) \tag{7.3}$$

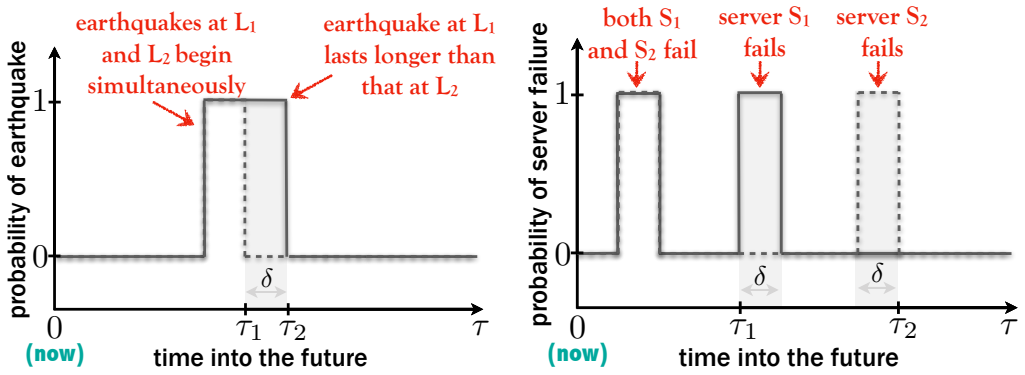
That is, the early warning score should not be boosted or lowered under the pretext that certain interventions will be given in the future. This is important as an agent may change how/when they administer interventions in response to the early warning system. For example, a system assuming that patients will be treated may optimistically not alert a caretaker, who relies on the system alerts to administer treatment.

7.3.2 Prior Works Violate Principles

Consider three 'label functions' that prior approaches typically employ an ML algorithm to predict: *(a) fixed look ahead* (FLA; [CSSS17]): event occurrence exactly after $\tau_* \in \mathbb{N}$ time steps; *(b) variable look ahead* (VLA; [FHH+17]): event occurrence within τ_* time steps; *(c) time to event* (T2E; [HHPS15], [NHR+18]): a monotonically decreasing function of the time until next event.

Approaches predicting FLA, VLA and T2E label functions at time t using past observations $(\mathbf{x}_{:t}, \mathbf{y}_{:t})$ violate all three principles of an ideal early warning system.

As argued in Section 7.1 and also by [PNMS13], predicting any function of the future is intervention-unaware unless intermediate interventions are accounted for; thus Principle 3 is



(a) Short vs. long earthquake with the same onset time

(b) Early vs. late server failure recurrence of same duration

Figure 7.3: (a) The longer earthquake at L_1 has a strictly dominating future event probability function. (b) The server S_1 failing in quick successions has a strictly dominating cumulative future event probability function. Hence, principles of dominance and precedence insist L_1 and S_1 should have strictly higher early warning scores than L_2 and S_2 respectively. However, in both cases, FLA, VLA and T2E produce tied scores when $\tau_* \notin [\tau_1, \tau_2]$.

violated. Now observe that if a label function violates Problem 7.1 and Problem 7.2, so do the approaches predicting it. Figure 7.3 presents two such counterexamples where all the label functions are in violation of the principles of dominance and precedence.

7.3.3 Problem

Overall, the early warning problem we seek to solve is:

Informal Problem 7.1: Intervention-Aware Early Warning

Given a set of trajectories $\mathbb{T}$, and for each $\mathcal{T} \in \mathbb{T}$, its (a) measurements $\mathbf{x}_{1:T}$ regarding M measurement types (categorical or real-valued), possibly containing missing values, (b) interventions $\mathbf{y}_t \in \{0, 1, \ldots, Y\}$ indicating the quantity administered, and (c) binary event label ℓ_t for an event of interest, at equidistant time instants $t = 1, 2, \ldots, T$, **learn** an early warning scoring function w which provably obeys Principles 1, 2 and 3.

7.4 Proposed Approach

Our main insight is to explicitly model the evolution of measurements in the presence and absence of interventions separately and leverage it to produce early warning scores. Thus, SmokeAlarm has two components:

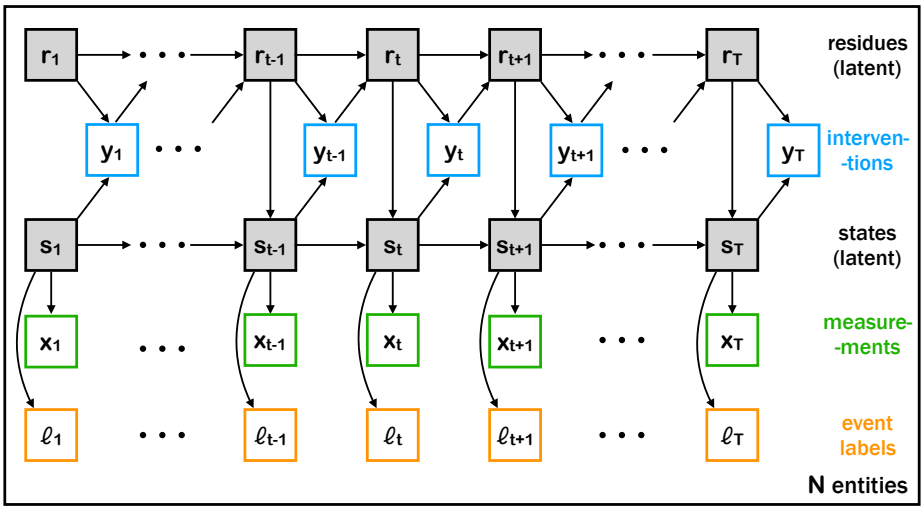


Figure 7.4: Probabilistic model used by SmokeAlarm

- **Intervention-Aware Modeling:** Given past trajectories labeled with the occurrences of a target event, SmokeAlarm learns a probabilistic model which takes the stochastic and prolonged effect of interventions into account. Specifically, the model learns how long the effect interventions last, and how measurements evolve in the presence and absence of their influence.
- **Early Warning Scoring:** When presented a stream of measurements and interventions of a trajectory under observation, SmokeAlarm uses the learned model to produce early warning scores in an online manner.

We describe these below, postponing analysis to Section 7.4.3.

7.4.1 Intervention-Aware Modeling

Figure 7.4 depicts how we model the evolution of observed variables (measurements $\mathbf{x}_t$, interventions $\mathbf{y}_t$ and event labels ℓ_t) shown in white boxes, through latent variables (states $\mathbf{s}_t$ and residues $\mathbf{r}_t$) shown in dark boxes.

Latent Variables: The latent *state* variable captures the progression stage at which the entity is w.r.t. the event (e.g., distant vs. close vs. occurrence) and takes categorical values $\{1, 2, \ldots, S\}$ for an input number of states S. The latent *residue* variable captures the residual quantity of intervention (e.g., medicines, from among those administered in the past) currently active in influencing the progression of states. It takes non-negative integral values $\{0, 1, 2, \ldots\}$.

Initial Variables: Each trajectory begins in a state $\mathbf{s}_1$ with an intervention residue $\mathbf{r}_1$ based on the initial distribution Φ.

Labels and Measurements: At time t, the event label ℓ_t and measurements $\mathbf{x}_t$ are noisy observations of the current state $\mathbf{s}_t$ according to the observation distribution Θ, and are further, conditionally independent of each other given the state. While simplifying the model and inference procedure, this still allows for the labels and measurements to depend on each other through the latent state. Thus,

$$\omega_{\mathbf{s}_t}(\mathbf{x}_t, \ell_t) \triangleq p(\mathbf{x}_t, \ell_t \mid \mathbf{s}_t) = p(\ell_t \mid \mathbf{s}_t) \cdot \prod_m p(\mathbf{x}_{tm} \mid \mathbf{s}_t) \tag{7.4}$$

Following standard practice in generative modeling [Rab89], we allow for per-state measurement distributions which are categorical or Gaussian depending on the type of measurement. As we show in Section 7.5, SmokeAlarm yields useful results even when these assumptions do not hold. Note also that SmokeAlarm naturally handles missing values via marginalization, i.e., simply dropping relevant terms from Equation (7.4).

Interventions: Based on the latent state and latent residue at time t, an expert uses an *intervention policy* π to determine the quantity $\mathbf{y}_t$ of intervention to administer:

$$\pi(\mathbf{y}_t \mid \mathbf{r}_t, \mathbf{s}_t) \triangleq p(\mathbf{y}_t \mid \mathbf{r}_t, \mathbf{s}_t), \quad y \in \{0, 1, \ldots, Y\} \tag{7.5}$$

where Y is the maximum admissible quantity of intervention. For example, when a person is healthy or is already pumped with medications, no further medication may be necessary.

Residues: As older interventions wear off, only a fraction of the $\mathbf{r}_t$ units of residue at time t 'survives' till $t{+}1$. If each residue stays active with an *activation probability* α, we have:

$$p(\mathbf{r}_{t+1} \mid \mathbf{r}_t, \mathbf{y}_t) = \mathcal{B}_{\mathbf{r}_t, \alpha}(\mathbf{r}_{t+1} - \mathbf{y}_t) \tag{7.6}$$

where $\mathcal{B}_{N,p}(\cdot)$ is the probability mass function of binomial distribution with N trials and success probability p. If $\alpha{=}1$, the intervention is always effective and remains so forever. $\alpha{=}0$ captures an intervention that is active for only one time step. Domain knowledge about the intervention can be incorporated by endowing α with a Beta prior, say β.

States: Finally, the next state $\mathbf{s}_{t+1}$ is derived based on current state $\mathbf{s}_t$ and the residual quantity of intervention $\mathbf{r}_{t+1}$ that remains active between t and $t{+}1$. When $\mathbf{r}_{t+1} = 0$, state transitions follow an $S{\times}S$ *intervention-free* state transition matrix $\mathbf{Q}_0$; when $\mathbf{r}_{t+1} > 0$, they follow *intervention-bound* state transition matrices $\mathbf{Q}_{\mathbf{r}_{t+1}}$. To capture the intuition that multiple units of intervention are administered to accelerate state progression, we tie these matrices as $\mathbf{Q}_{\mathbf{r}_{t+1}} = \mathbf{Q}_1^{\mathbf{r}_{t+1}}$ so that the effect of $\mathbf{r}_{t+1} \geq 2$ active units of intervention is equivalent to that of a single active unit lasting $\mathbf{r}_{t+1}$ time steps. This also keeps the number of parameters small. Overall,

$$p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{r}_{t+1}) = \mathbf{Q}_{\mathbf{r}_{t+1}}(\mathbf{s}_t, \mathbf{s}_{t+1}) \tag{7.7}$$

Inference: Given a set of labeled trajectories $\mathbb{T}$, the goal of model inference is to find the optimal parameters $\Lambda^* = \{\Phi^*, \Theta^*, \pi^*, \alpha^*, \mathbf{Q}^*\}$ and the latent residues $\mathbf{R}^*$ and states $\mathbf{S}^*$ for all trajectories which maximize their posterior probability given the observed data $\mathbb{T}$ and activation prior β.

$$\Lambda^*, \mathbf{R}^*, \mathbf{S}^* = \arg\max_{\Lambda, \mathbf{R}, \mathbf{S}} p(\Lambda, \mathbf{R}, \mathbf{S} \mid \mathbb{T}; \beta) \tag{7.8}$$

To perform this optimization efficiently, we adopt coordinate ascent strategy, which is known to converge faster than standard expectation-maximization [DLR77] in practice, while yielding results of a similar quality [YML+14]. This is sketched below.

1. Initialize the model parameters Λ.
2. Fix the model parameters Λ and find the best assignment of latent variables $(\mathbf{R}, \mathbf{S})$ maximizing the objective using a Viterbi dynamic programming algorithm [Rab89].
3. Fix the latent variables $(\mathbf{R}, \mathbf{S})$ and find the optimal model parameters Λ maximizing the objective. Due to the distributions chosen, this can be solved in closed-form.
4. Repeat steps (2) and (3) till $(\Lambda, \mathbf{R}, \mathbf{S})$ change no more or a maximum number of iterations is reached.

We initialize the coordinate ascent procedure using clustering for states (handling missing and categorical data appropriately), mode of activation prior β as the activation probability α, and uniform distribution for transition matrices and initial distributions. We note that (2) is the most computationally intensive step; however, it can be carried out embarrassingly in parallel over all trajectories. In the interest of space, we omit details on steps (2) and (3) which are standard.

7.4.2 Early Warning Scoring

Given the learned model Λ, we propose to use the following early warning function to score a test trajectory $\mathcal{T}$:

Definition 7.4: EDoc Early Warning Function

The EDoc early warning function scores a trajectory $\mathcal{T}$, with future event probability function f, as the Expected Discounted event Occurrence Count in the future.

$$w(\mathcal{T}; \Lambda) \triangleq \sum_{\tau=0}^{\infty} \gamma^\tau f(\tau) = \sum_{0}^{\infty} \gamma^\tau p(\ell_{t+\tau} = 1 | \mathcal{T}; \Lambda) \tag{7.9}$$

Here, $\gamma \in (0, 1)$ is the discount factor determining the relative importance of event occurrences at different times in the future.

Thus, EDoc warning score is equivalent to the reinforcement learning notion of *return* [SB18] when rewards are set to the probability of event occurrences based on the learned model Λ. There are two considerations to bear in mind, however. First, in accordance with Problem 7.3 (Intervention-Awareness), the score must be computed using an intervention-free future, i.e.,

under the no-intervention policy at all times $\tau > t$.

$$\pi_0(y|r,s) = \mathbb{I}[y == 0] \; \forall \; r, s \tag{7.10}$$

Here, $\mathbb{I}[\cdot]$ is the identity function. Second, as the intervention policy π' observed in the test trajectory can differ significantly from π learned during training, the scoring should use π' rather than π up to time t. As π' is typically unknown, a trajectory-specific time-varying policy estimate based on the observed interventions $\hat{\pi}'_t(\mathbf{y}_t \mid r, s) = 1 \; \forall \; r, s$ may be used. Theorem 7.1 shows how to compute early warning scores efficiently online.

7.4.3 Theoretical Analysis

Our main theoretical results are that SmokeAlarm computes EDOC early warning scores in an online manner (Theorem 7.1) and adheres to all principles from Section 7.3.1 (Theorem 7.2).

Theorem 7.1: Online Early Warning

SmokeAlarm produces EDOC early warning scores on an evolving trajectory $\mathcal{T}{=}(\mathbf{x}_{:t}, \mathbf{y}_{:t})$ efficiently in a constant time per new observation $(\mathbf{x}_t, \mathbf{y}_t)$, independent of the length of its history.

Proof. Using Principle 7.3, $w(\mathcal{T})$ can be written as:

$$w(\mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:}{=}\mathbf{0}) = \sum_{\tau=0}^{\infty} \gamma^\tau p(\ell_{t+\tau}{=}1 | \mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:}{=}\mathbf{0})$$

Marginalizing over the latent variables at time t and using the Markov property, we obtain $\sum_{\mathbf{r}_t, \mathbf{s}_t} \sum_{\tau=0}^{\infty} \gamma^\tau p(\mathbf{r}_t, \mathbf{s}_t | \mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:}{=}\mathbf{0}) \cdot p(\ell_{t+\tau}{=}1 | \mathbf{r}_t, \mathbf{s}_t, \mathbf{y}_t, \mathbf{y}_{t+1:}{=}\mathbf{0})$. Thus, we derive:

$$w(\mathcal{T}) = \sum_{\mathbf{r}_t, \mathbf{s}_t} \underbrace{p(\mathbf{r}_t, \mathbf{s}_t | \mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:}{=}\mathbf{0})}_{\textit{latent variable distribution at } t} \cdot \underbrace{w_*(\mathbf{r}_t, \mathbf{s}_t, \mathbf{y}_t)}_{\textit{early warn. score}} \tag{7.11}$$

where $w_*(r, s, y)$ is the early warning score output for a trajectory starting in (r, s) with an intervention y:

$$w_*(r, s, y) = \sum_{\tau=0}^{\infty} \gamma^\tau p(\ell_{t+\tau}{=}1 | \mathbf{r}_t{=}r, \mathbf{s}_t{=}s, \mathbf{y}_t{=}y, \mathbf{y}_{t+1:}{=}\mathbf{0}) \tag{7.12}$$

Observing that w_* does not depend on the history of the trajectory and hence need to be computed exactly once a priori (Lemma 7.1) and that the latent variable distribution at every time step can be updated efficiently online (Lemma 7.2) completes the proof.

Lemma 7.1: Precomputation of Early Warning Table

The $R \times S \times Y$ early warning table containing all early warning scores $w_*(r, s, y)$ can be precomputed in $\mathcal{O}\big(R^2S^2(RS + Y)\big)$ time complexity.

Proof. Unrolling the sum in Equation (7.12) over a single step in the future and marginalizing over the latent state and residue at the next time step, we derive the following recursive relation:

$$w_*(r, s, y) = \rho_s + \gamma \sum_{r',s'} p(r'|r, y) \cdot p(s'|s, r') \cdot w_*(r', s', 0) \tag{7.13}$$

For $y{=}0$, we construct the system of linear equations $w_*(r, s, 0) = \rho_s + \gamma \sum_{r',s'} p(r'|r, 0) \cdot p(s'|s, r') \cdot w_*(r', s', 0), \forall\, r, s$ which can be solved by matrix inversion in $\mathcal{O}\big(R^3S^3\big)$. Plugging these in Equation (7.13), other scores are precomputed in an additional $\mathcal{O}\big(R^2S^2Y\big)$ time. ■

Lemma 7.2: Online Computation of Latent Variables

The distribution $p(\mathbf{r}_t, \mathbf{s}_t|\mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{y}_{t+1:}{=}\mathbf{0})$ of latent variables at every time step t for an evolving trajectory $\mathcal{T} = (\mathbf{x}_{:t}, \mathbf{y}_{:t})$ can be computed using dynamic programming in $\mathcal{O}\big(R^2S^2\big)$ time per new pair $(\mathbf{x}_t, \mathbf{y}_t)$.

Proof. Define $\psi_t(r, s) = p(\mathbf{x}_{:t}, \mathbf{y}_{:t}, \mathbf{r}_t{=}r, \mathbf{s}_t{=}s, \mathbf{y}_{t+1:}{=}\mathbf{0})$ as the probability of observing measurements and interventions till time t, landing in latent variables (r, s) at time t and observing no interventions thereafter. In terms of ψ, the required probability is $\psi_t(r, s)/\sum_{r',s'} \psi_t(r', s')$. Thus, we need only show how to compute $\psi_t(r, s)$ efficiently.

The base case is $\psi_1(r, s) = \Phi(r, s) \cdot p(\mathbf{x}_1|s) \cdot \pi'(\mathbf{y}_1|r, s) \cdot \prod_{\tau=2}^{\infty} \sum_{\mathbf{r}_\tau,\mathbf{s}_\tau} p(\mathbf{r}_\tau, \mathbf{s}_\tau|\mathbf{r}_{\tau-1}, \mathbf{s}_{\tau-1}, \mathbf{y}_{\tau-1}) \cdot \pi_0(\mathbf{y}_\tau|\mathbf{r}_\tau, \mathbf{s}_\tau)$ which can be simplified using Equation (7.10) as $\Phi(r, s) \cdot p(\mathbf{x}_1|s) \cdot \pi'(\mathbf{y}_1|r, s)$. Thus, ψ_1 can computed in $\mathcal{O}\big(RS\big)$ time.

In a similar way, for $t{>}1$, we derive: $\psi_t(r, s) = \sum_{r',s'} \psi_{t-1}(r', s') \cdot p(r'|r, \mathbf{y}_{t-1}) \cdot p(s|s', r) \cdot p(\mathbf{x}_t|s) \cdot \pi'(\mathbf{y}_t|r, s)$ which can be computed in $\mathcal{O}\big(RS\big)$ for every r, s from the latent distribution ψ_{t-1} at the previous time step. ■

This proves the theorem. ■

Theorem 7.2: Adherance to Principles

SmokeAlarm follows all three principles–dominance, precedence and intervention-awareness–of an ideal early warning system.

Proof. Consider two trajectories $\mathcal{T}_1, \mathcal{T}_2$ with future event probability functions f_1 and f_2 and cumulative future event probability functions F_1 and F_2 respectively. For $i = 1, 2$, let $F_i(-1) = 0$ so that $f_i(\tau) = F_i(\tau) - F_i(\tau - 1)\ \forall\ \tau = 0, 1, \ldots$. Using Equation (7.9), $w(\mathcal{T}_1) - w(\mathcal{T}_2) = \sum_{\tau=0}^{\infty} \gamma^\tau (f_1(\tau) - f_2(\tau))$.

Principle 7.1 (Dominance): Suppose $f_1 \geq f_2$. Then, $f_1(\tau) - f_2(\tau) \geq 0\ \forall\ \tau$ and hence $w(\mathcal{T}_1) - w(\mathcal{T}_2) \geq 0$. Suppose instead that $f_1 > f_2$ with $\mathcal{C} = \{\tau : f_1(\tau) > f_2(\tau)\} \neq \{\}$. As $f_1(\tau) = f_2(\tau)\ \forall\ \tau \notin \mathcal{C}$, we obtain $w(\mathcal{T}_1) - w(\mathcal{T}_2) = \sum_{\tau \in \mathcal{C}} \gamma^\tau (f_1(\tau) - f_2(\tau)) > 0$ as desired.

Principle 7.2 (Precedence): In terms of the cumulative future event probability function, $w(\mathcal{T}_1) - w(\mathcal{T}_2) = \sum_{\tau=0}^{\infty} \gamma^\tau [F_1(\tau) - F_1(\tau-1) - F_2(\tau) + F_2(\tau-1)] = \sum_{\tau=0}^{\infty} (\gamma^\tau - \gamma^{\tau+1})(F_1(\tau) - F_2(\tau))$ where $\gamma^\tau - \gamma^{\tau+1} > 0$ as $\gamma \in (0, 1)$. Suppose $F_1 \geq F_2$. Then, $F_1(\tau) - F_2(\tau) \geq 0\ \forall\ \tau$ and hence $w(\mathcal{T}_1) - w(\mathcal{T}_2) \geq 0$. Suppose instead that $F_1 > F_2$ with $\mathcal{C} = \{\tau : F_1(\tau) > F_2(\tau)\} \neq \{\}$. As $F_1(\tau) = F_2(\tau)\ \forall\ \tau \notin \mathcal{C}$, we obtain $w(\mathcal{T}_1) - w(\mathcal{T}_2) = \sum_{\tau \in \mathcal{C}} (\gamma^\tau - \gamma^{\tau+1})(F_1(\tau) - F_2(\tau)) > 0$ as desired.

Principle 7.3 (Intervention-Awareness): This follows by construction from the first line in the proof of Theorem 7.1. ■

7.5 Experiments

We conduct extensive experiments on both synthetic and real-world data to answer the following questions: **[Q1] Accuracy:** How well does SmokeAlarm perform compared to baselines? **[Q2] Interpretability:** Is the model learned easy to interpret? **[Q3] Discoveries:** Does it lead to interesting discoveries in practice? **[Q4] Scalability:** How does the running time scale with input size? Our experiments focus on healthcare settings, but the ideas can be more broadly applied.

7.5.1 Experimental Setup

We implement SmokeAlarm in Python using the *pomegranate* library for probabilistic modeling [Sch17] and run experiments on a machine with 64 2.67GHz Intel Xeon E7-8837 CPUs.

Baselines: Due to our emphasis on interpretability, we compare to the following linear approaches: **(a) CoxT2E** or Cox Proportional Hazards Model [LW89] is a (linear) survival model to evaluate the effect of multiple variables on the time to an event, **(b) LinearFLA** and **(c) LinearVLA** which predict whether an event happens exactly at or within τ_* steps in the future by performing least squares regression with L2 regularization. All baselines are intervention-unaware: they predict the future without carefully considering the intermediate interventions.

Evaluation Metrics: Principle 7.3 advocates for early warning scores which reflect future event occurrences in the absence of interventions. Thus, we use ground truth future events *only from intervention-free test data* for evaluation. Recall that each method assigns a score per time step of each trajectory, a higher value signifying a greater risk of an impending event. Sorting these in descending order (breaking ties randomly), we compute precision and recall as in [WH98]. Specifically designed for early warning, these metrics normalize for multiple alarms

of the same event and account for the fact that false alarms "located closely together may not be as harmful as the same number spread out over time". We detail the procedure below.

An event is said to be correctly predicted if there is at least one alarm in its preceding L-length window. Conversely, each alarm is 'active' for a duration L after being raised, during which an event is expected to take place. Let *TP(k)* be the number of distinct events correctly predicted using the top k alarms. Let *FP(k)* be the number of complete non-overlapping L-length 'active' windows in which no events take place. If N is the total number of events, we derive: *Precision@k = TP(k) / (TP(k) + FP(k))* and *Recall@k = TP(k) / N*. We summarize the overall accuracy using the area under curve (AUC) measure. In addition, we quantify the earliness of warning using *average lead time* over all early warned events, where the lead time of an event is determined by the earliest alarm in its preceding L-length window. For all metrics, a higher value is more desirable, and all except average lead time lie in $[0, 1]$.

7.5.2 Synthetic Data

Synthetic data allows us to control the interventions without incurring high human costs, e.g., student drop out, patient death. Thus, by varying the intervention policy in the training data, we study how intervention-aware different methods are.

Data Generation: We generate a SyntheticFlu dataset with temperature and white blood cell (WBC) count measurements akin to [DS16] such that high temperature *or* high WBC count indicates flu. 40% of the population is healthy; the rest develop flu due to an escalating measurement (30% each). Aspirin interventions decrease temperature for the next 3 hours but do not affect WBC count. We create two training datasets: *set (+I)* with aspirin interventions and *set (-I)* without. Each method is trained on both sets in turn and tested on the same held-out set of intervention-free data. More details are given below.

Measurements for temperature and WBC count are independently drawn from a Hidden Markov Model with 10 latent states $\{0, 1, \ldots, 9\}$ such that the observed value in state s is normally distributed as $\mathcal{N}(s, \sigma^2)$. Each subject begins in a state $s \leq 3$ for temperature and WBC count. For a stable measurement, a state s decreases to $s-1$, remains the same and increases to $s+1$ during the next hour with probabilities 0.2, 0.7 and 0.1 respectively. The corresponding values for an escalating measurement are 0.2, 0.3 and 0.5 so that the value tends to increase. Subjects in states $s \in \{8, 9\}$ for temperature or WBC count have *flu*. When either reaches state 9, the subject expires and their trajectory terminates. Aspirin is given with probability p when temperature is in states 6-8. When administered, it decreases the temperature by six states over the next three hours. With probability 0.4, it may also *stabilize* the temperature to prevent future escalation. Aspirin interventions are binary (ignoring quantity administered). All values are recorded at hourly intervals for a maximum duration of 50 hours for each trajectory. We use a mixture of aspirin probabilities $p \in \{0.5, 0.3, 0.1\}$ for *set (+I)* and set $p=$ for *set (-I)* and test data. We set the noise level σ^2 to 0.04. All training and test datasets contain 5000 trajectories each.

Parameters: For SmokeAlarm, we set $S=16$ states, discount factor $\gamma=0.75$, and activation prior in the ratio 2:1 (scaled to the dataset size) to incorporate that aspirin lasts around 3 hours

Table 7.2: Accuracy (AUC) on SyntheticFlu When the Trained on Data Untainted (-I) and Tainted (+I) by Interventions

Setting	CoxT2E	LinearFLA	LinearVLA	SmokeAlarm
Untainted set (-I)	0.9189	0.9188	0.9185	**0.9993**
Tainted set (+I)	0.8845	0.8127	0.8452	**0.9985**
Drop in accuracy	3.74%	11.5%	7.98%	**0.08%**

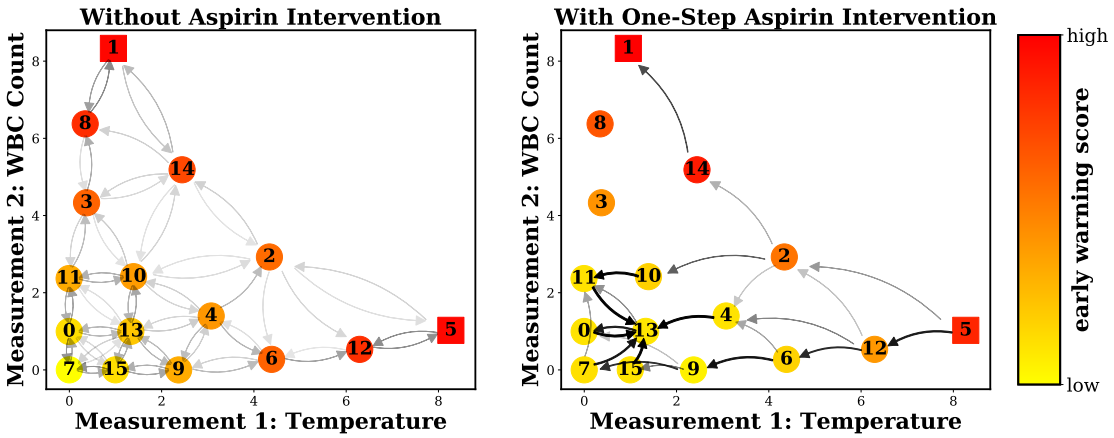


Figure 7.5: Model learned on SyntheticFlu set (+I) peppered with interventions shows that SmokeAlarm successfully learns the evolution of measurements (and hence the risk of flu) in the presence and absence of aspirin.

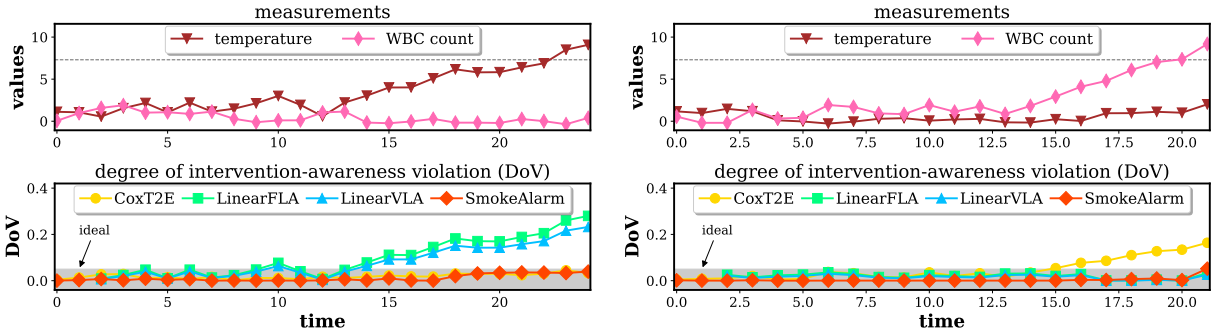
(in expectation). Linear regression baselines use the three most recent measurements and interventions (i.e., shingle size 3) to predict event within/at τ_*=9 hours in the future. Accordingly, we use L=9 hours as the maximum lead time for evaluation.

7.5.2.1 Q1. Accuracy

Table 7.2 summarizes the AUC of all methods using sets (+I) and (-I) for training. Bold indicates the best performing method for each metric. First, note that all methods perform their best when trained using set (-I) whose no-intervention policy (p=0) matches that of the test set. The change from set (-I) to set (+I) hurts baselines the most, with accuracy dropping up to 11.5%. In contrast, the performance of SmokeAlarm remains comparable, suggesting that learning separate models for the presence and absence of interventions pays off. Thus, it is able to produce early warning scores untainted by interventions from limited intervention-free data.

7.5.2.2 Q2. Interpretability

Figure 7.5 depicts the model learned by SmokeAlarm on *set (+I)* in the absence of aspirin (left) and under a single aspirin intervention followed by an intervention-free future (right). States (numbered vertices) are plotted using the mean of their temperature and WBC count distribu-



(a) A SyntheticFlu trajectory with temperature escalation

(b) A SyntheticFlu trajectory with WBC count escalation

Figure 7.6: SmokeAlarm is intervention-aware: Degree of intervention-awareness violation (DoV) on representative trajectories confirms that only SmokeAlarm yields a consistently low DoV within the grey 'ideal' band. The baselines have high DoV which grows in $t{>}15$, which is the crucial period for early warning.

tions. Squares indicate states with flu, i.e., a high value of $p(\ell{=}1 \mid \mathbf{s})$. Colors (yellow=healthy, red=sick) represent their early warning scores in the presence or absence of aspirin, as applicable. Dark and thick arrows depict probable state transitions.

In both figures, healthy states with low temperature and WBC count have the lowest scores (yellow), flu states $\{1, 5\}$ with high temperature or WBC count have the highest scores (red), and the red shade lightens towards the origin. Orange circular vertices are the *early warning states*, where there is no flu, but the score is high and an alarm is triggered. Without aspirin (left), red fades symmetrically along both axes because the measurements evolve and contribute to flu similarly. With aspirin (right), red fades faster along temperature axis going from state 5 to 7. The yellower colors of states 6 and 12 in the presence of aspirin showcases a decreased risk of flu and is consistent with the high probability 'becoming-healthier' transitions from states 5 to 12 to 6 to 9. Thus, SmokeAlarm successfully learns that without aspirin, high temperature states $\{6, 12\}$ are as dangerous as high WBC count states $\{3, 8\}$ with respect to flu; however, aspirin lowers temperature and hence also the imminent danger of flu from high temperature states.

7.5.2.3 Q3. Discoveries

Figure 7.6 depicts two representative trajectories–with different ways of flu escalations–from test data. The top panels show the temperature and WBC count measurements; the person has the flu if at least one of them cross the dotted line. The bottom panels plot the *degree of intervention-awareness violation* (DoV) which measures the extent to which Principle 7.3 is violated. If the early warning scores produced by a method on a trajectory are w_+ and w_- when trained on sets (+I) and (-I) respectively, DoV$=|w_+ - w_-|$. Ideally, $w_- = w_+$ and DoV=0 as the underlying risk of flu does not depend on training data. However, Figure 7.6 reveals that the baselines produce a large DoV in at least one type of flu escalation. Notably, DoV is high for

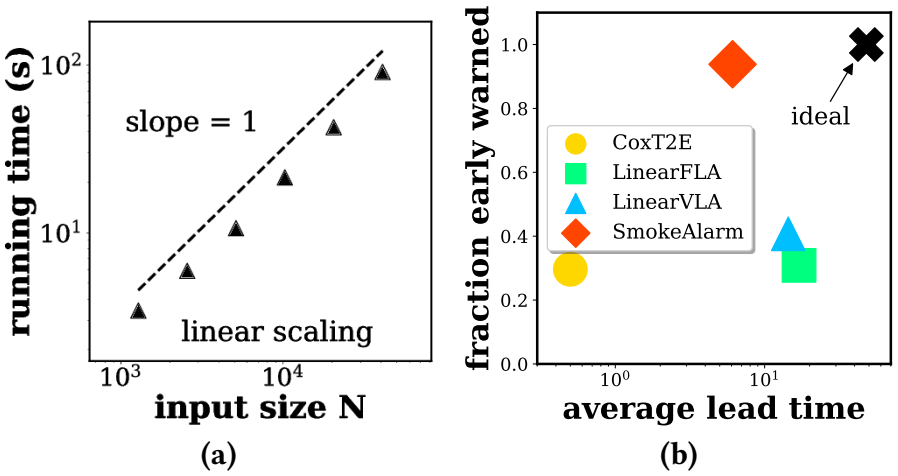


Figure 7.7: SmokeAlarm (a) scales linearly with input size N; (b) early warns 93.7% patients (avg. lead time: 6.1 hours) before the onset of septic shock.

$t \geq 15$ which is the crucial period for early warning. Only SmokeAlarm yields a consistently low DoV lying within the gray 'ideal' band at all times and for both flu escalations.

7.5.2.4 Q4. Scalability:

We vary the input size, i.e., total number of time steps across all trajectories, and measure the average time taken (excluding IO operations) for early warning scoring (testing phase) over five runs. Figure 7.7(a) yields a line with slope 1 in log-log scales; thus, the running time is linear on the input size N. The time for model inference (training phase) is also linear, but it is not shown in the interest of space.

7.5.3 Real-World Data

We consider the task of early warning against septic shock, an adverse outcome of bacterial infection in the ICU and a leading cause of mortality. Data is extracted from the publicly-available MIMIC-III Clinical Database [JPS+16]. We extract values of SOFA scores, serum lactate, mean arterial pressure, and vasopressor interventions, preprocessing and aggregating appropriately in 4-hour intervals as detailed in the following paragraph. The data is labeled following Sepsis-3 guidelines [SDS+16] assuming a baseline that all subjects are suspected of infection. We use data with and without vasopressor interventions for training (3469 subjects; around 30% with septic shock and 17% vasopressor-free) but validate and test only on subjects who were never given vasopressors during their stay (400 and 600 subjects respectively, with $13-13.5\%$ having septic shock).

Data Preprocessing: From the Metavision information system of MIMIC-III, we extract seven relevant measurements–*bilirubin, creatinine, Glasgow coma score, mean arterial pressure, PaO2/FiO2, platelet and serum lactate*–processing them to ignore extreme values (e.g., typos) and clipping

Table 7.3: Precision and Recall for Early Warning of Septic Shock (underline shows significant differences at p=0.0001)

Method	Precision @ k			Recall @ k		
	300	600	900	300	600	900
CoxT2E	0.89	0.81	0.78	0.38	0.62	0.79
LinearFLA	0.63	0.78	0.82	0.14	0.40	0.60
LinearVLA	0.64	0.81	0.83	0.13	0.48	0.62
SmokeAlarm	**1.00**	**0.95**	**0.89**	**0.51**	**0.85**	**0.97**

them to appropriately defined intervals based on the tail behavior of the per-measurement distributions. We further combine the first six measurements into Sequential Organ Failure Assessment (SOFA) score, a routinely evaluated measure of ICU mortality by doctors. We aggregate all values in 4-hour intervals, by averaging the real-valued measurements and OR-ing the binary interventions (ignoring the quantity). We focus on subjects at least 15 years of age at the time of hospital admission who were not admitted due to septic shock and whose stay lasts between 2-50 days. If the same subject is admitted multiple times, we treat each such admission as a different trajectory, ignoring trajectories which have excessive missing values ($> 80\%$) or do not have at least one assessment for each measurement. While SmokeAlarm naturally handles missing values, the baselines do not. Thus, for a fair comparison, we supply the imputed data to all methods, where imputation is done using linear interpolation followed by backward/forward fill at the beginning/end of the trajectories. We follow the Sepsis-3 guidelines [SDS+16] for labeling of septic shock (SOFA score ≥ 2, mean arterial pressure < 65 despite vasopressors/fluid resuscitation, serum lactate $>$ 2mmol/L) assuming a baseline that all subjects are suspected of infection. This resulted in a cohort of 4469 subjects, with 1374 (30.7%) positive with septic shock and 1606 vasopressor-free throughout their stay.

Parameters: For SmokeAlarm, we set the activation prior in the ratio 1:1 to capture that patients in ICU are given fast-acting drugs. We tune the number of states $S \in \{8, 12, 16, 20, 24\}$ and discount factor $\gamma \in \{0.2, 0.4, 0.6, 0.8\}$ to maximize validation AUC. The best performing model (S=20, γ=0.4) was used for evaluation. The linear baselines use a shingle size of three to predict septic shock in τ_*=36 hours. We use a larger window of L=48 hours as the maximum lead time in evaluation.

7.5.3.1 Q1. Accuracy

Table 7.3 tabulates the precision and recall of all methods at cut-off ranks $k \in \{300, 600, 900\}$. Bold shows the best performing method according to each metric. We see that SmokeAlarm consistently outperforms all baselines, achieving $8 - 288\%$ higher precision and recall at all ranks considered. The gains are statistically significant according to a two-sided micro sign test [YL99].

Figure 7.1(a), plotting the precision vs. recall at all cut-off ranks k, reveals that the curve for SmokeAlarm (red) lies completely above those of all baselines. Said another way, SmokeAlarm

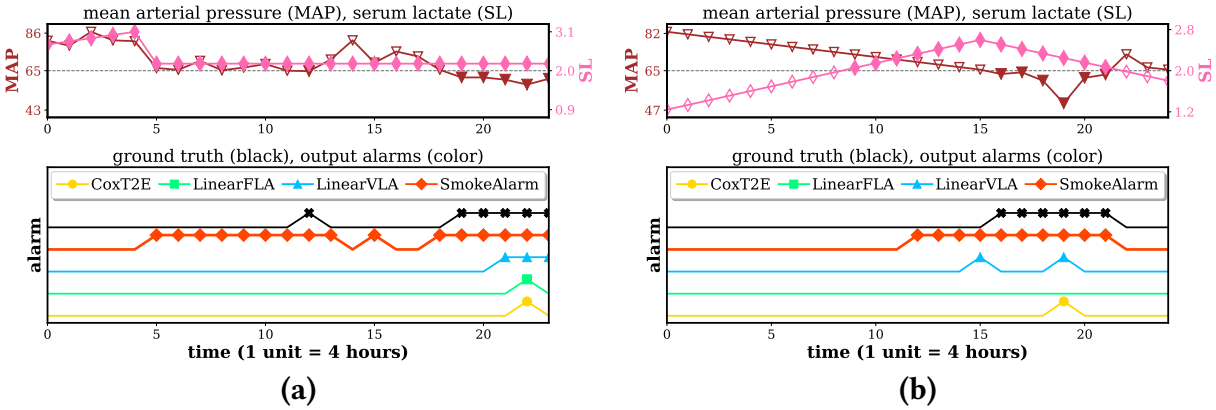


Figure 7.8: Trajectories of patient with septic shock from the test set showing measurements (except SOFA score; top panel), ground truth septic shock label (black; bottom panel) and alarms output by various methods (bottom panel).

achieves higher precision for every recall value. This is also reflected in AUC=0.98 of SmokeAlarm, which is 16% higher than the best baseline. The operating point (shown as stars) is chosen at a precision of 0.827, the highest common value of precision achieved by all methods.

As the onset of septic shock is the hardest and the most valuable to predict, we now measure performance of each method based only on the first septic shock event of all patients who are labeled positive. Specifically, we compute the fraction of septic shock patients who are early warned of their first septic shock event, and also the average lead time of such a warning. Figure 7.7(b), plotting the results, shows that SmokeAlarm warns 93.7% patients with a mean lead time of 6.1 hours before septic shock onset. The baselines warn only $<41\%$ patients; so their mean lead time arguably does not matter.

7.5.3.2 Q2. Interpretability

Figure 7.1(b) depicts the model learned by SmokeAlarm on Mimic-III data without vasopressors (left) and with one-step vasopressor followed by an intervention-free future (right). States (numbered vertices) are plotted by their mean arterial pressure (MAP) and serum lactate (SL) values and colored based on their early warning scores. SOFA score turned out to be uniformly high across most patients (and most states) and hence is omitted. Squares indicate 'septic shock' states with a high value for $p(\ell{=}1 \mid \mathbf{s})$. Only high probability transitions (≥ 0.05, i.e., uniform distribution over $S{=}20$ states) are shown. Darker and thicker arrows indicate more likely state transitions.

In both figures, healthy states with low SL and/or high MAP have the lowest scores (blue), septic shock states $\{1, 7, 17\}$ with high SL (>2) and low MAP (<65) have the highest scores (red). The early warning scores decrease diagonally downward as indicated by color change from red to yellow to green to blue. Yellow and green circular vertices are the *early warning states*, where there is no septic shock, but the score is high enough that an alarm is triggered at operating point (red star in Figure 7.1(a)). We note an increased number of thicker/darker arrows pointing rightward in the presence of vasopressors as they tend to increase MAP by con-

stricting blood vessels. Thus, the scores of low MAP states $\{1, 7, 17\}$) decreases considerably, with the change in vertex color of states 7 and 17 being the most apparent. Thus, SmokeAlarm successfully learns that high SL and low MAP are linked to septic shock and that vasopressor interventions tend to increase low MAP and consequently decreases risk of septic shock.

7.5.3.3 Q3. Discoveries

Figure 7.8 shows trajectories from two patients in the test set. The top panel plots their MAP and SL measurements (SOFA score was consistently above 2 and hence omitted). The healthy range for the measurements are MAP$\geq$65 (above the dashed line; brown hollow triangles) and SL$\leq$2 (below the dashed line; pink hollow diamonds). Abnormal values are indicated using filled markers. When all measurements are abnormal, the patient has septic shock; these appear as spikes with crosses in the black curve of the bottom panel. Alarms output by various methods at their operating points are also indicated via markers and spikes in the bottom.

For the patient in Figure 7.8(a), serum lactate (SL) value is unhealthy throughout. MAP starts at a healthy range around 85 and declines rapidly at t=5 and steadily thereafter until septic shock occurrence at t=12. Despite a brief recovery around t=14, the person goes on to have septic shock for $t\geq$19. As seen, SmokeAlarm is the only method which alarms the first septic shock occurrence. Notably, the lead time is 28 hours and the first alarm coincides with the first sharp decline in MAP at t=5. The alarm stops briefly when MAP appears to increase around t=14, but as MAP declines further, the alarms restart at t=18, four hours before the septic shock at t=19. The baselines entirely miss the first septic shock incidence, and only provide late alarms for even the longer– and perhaps more severe–septic shock incidence at $t\geq$19.

The patient in Figure 7.8(b) begins with measurements which are normal, but escalating. Among the baselines, LinearFLA provides no warning and CoxT2E issues only a late alarm 12 hours into septic shock. Only LinearVLA raises an early alarm (lead time: 4 hours), but it still fails to continue warning through the period of septic shock. In contrast, SmokeAlarm warns 16 hours before the values decline past the dashed line and continues to warn until the patient has safely recovered.

Thus, our experiments show that SmokeAlarm outperforms baselines, scales linearly with data size, is visually interpretable, and yields interesting discoveries on real-world data.

7.6 Conclusion

We considered the problem of learning to interpretably early warn from labeled data tainted by interventions. We proposed SmokeAlarm, an intervention-aware method which learns offline from past labeled data containing interventions and produces early warning scores online. Moreover, it is "bi-inspectable", i.e., the model can be visualized both in the presence and in the absence of an intervention. SmokeAlarm also provably obeys all three principles of an ideal early warning system, Applied on real-world data, it outperforms baselines by $16 - 38\%$ in terms of AUC, while also early warning with an average lead time of 6.1 hours before the onset of septic shock.

Part IV

Conclusion

Chapter 8

Conclusion and Future Work

8.1 Summary

This thesis provides a suite of algorithms for anomaly detection in static and dynamic graphs which leveraging several key insights from real-world data. Specifically,

Static Graphs: We broaden the scope of present literature on graph semi-supervised learning–a core problem in mining anomalies within static graphs when given a few labeled examples–in the following ways:

- Our **ZooBP** algorithm [EGF+17b] described in Chapter 2 takes into account the *heterogeneity in vertex and edge types* of real-world graphs to provide a principled accurate approximation of Belief Propagation, which not only has closed-form solution and convergence guarantees, but also provides 2-600× platform-dependent speedups.
- Our **NetConf** algorithm [EGF17a] described in Chapter 3 exploits the *skewed-degree distributions* of vertices in real-world graphs to incorporate a notion of confidence or uncertainty during inference. By carefully propagating messages which capture not just point estimates, but their underlying uncertainty, NetConf improves classification accuracy by up to 5% absolute percentage points, while still having closed-form solution and convergence guarantee.
- Our **HOLC** metric described in Chapter 4 establishes that *higher-order network structures* such as triangles and 4-cliques are more consistent in labels compared to edges in real-world graphs. Subsequently, our **HOLS** algorithm [EKF20] leverages the signal present in higher-order network structures to improve classification accuracy by up to 4.7% relative percentage points, in comparable runtime.

Dynamic Graphs: We push the state-of-the-art in near real-time detection and early warning of anomalies and events in the following ways:

- Our **SpotLight** algorithm [EFGM18] described in Chapter 5 adopts the powerful randomized sketching based approach to provably detect the *sudden appearance or disappearance of large dense directed subgraphs* under the stringent time and memory constraints

of the streaming setting, leading to 11-46% (statistically significant) relative percentage points improvements compared to prior approaches.

- Our **SedanSpot** algorithm [EF18] described in Chapter 6 exploits the *lockstep behavior* of anomalies to detect anomalous *bridge edges* under time and space constraints of the streaming setting. The main idea was to lower the probability of sampling anomalous edges, and maintain a better representation of normal behavior, leading to 270% improvement in accuracy in terms of relative percentage points in 3× lesser time.
- Our **SmokeAlarm** algorithm [EFMN19] described in Chapter 7 learns to early warn against upcoming anomalies, by utilizing supervision, in an online and interpretable manner satisfying three principles of an ideal early warning system – *dominance, precedence and intervention-awareness.* Applied on real-world data, SmokeAlarm outperforms baselines by 16-38% AUC, while early warning 6.1 hours before septic shock onset on average.

For reproducibility, we open-source most of the algorithms proposed in the thesis and use publicly-available datasets wherever possible.

8.2 Vision and Future Work

This thesis takes a step towards pushing the boundary of anomaly detection by developing *principled, efficient, effective* for large-scale static and dynamic graphs. Below, we outline three concrete research direction towards our ultimate vision for this niche area, which is *holistic, self-learning, adversarially robust* algorithms for anomaly detection.

Holistic Algorithms for Anomaly Detection: Our proposed algorithms leveraged connectivity and temporal information to flag anomalies. But in many real-world applications, vertices and edges may have attributes (e.g. demographics of users, ratings/reviews on edges), which can act as valuable side-information. For example, a set of users who each have a single (positive) rating, to the same product in the same month may only be a little suspicious, since there are not enough ratings per user to make a high-confidence prediction. But this suspicion could be further bolstered by observing that these users are named user001, user002, user003, and so on. Thus, we need approaches which can holistically model the simultaneous evolution of connectivity structure and attributes to more accurately detect anomalies.

Self-Learning Algorithms for Anomaly Detection: In many real-world settings including security and health care, anomaly detection systems are used for human decision making. In such cases, labels (true positive or false positive) for flagged anomalies are readily available based on human feedback. An open challenge is to develop a principled algorithmic framework which can leverage this feedback in an online manner to tailor the definition of anomalousness to the needs of the user, either by tweaking the underlying anomaly detector itself or by filtering out uninteresting/expected anomalies using a online classifier in conjunction with the anomaly detector.

Adversarially Robust Algorithms for Anomaly Detection: Consider anomalies which arise from malicious behavior of fraudsters, e.g., fake reviews or followers, network intrusion attacks. If an anomaly detector is deterministic and does not account for adversaries, intelligent fraudsters can adjust their behaviors in response to the improvements in detection algorithms and successfully evade detection. A first step towards deterring such attacks is the use of randomization as in SpotLight and SedanSpot, so that even though the adversary knows the algorithm employed for detection, the exact sequence of random coin tosses remains private, making evasion difficult. The next step would be to leverage game theoretic ideas for adversarially robust anomaly detection.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002. 58, 66

[ACF13] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. In *ICWSM*, pages 2–11, 2013. 16, 22, 38, 42, 58

[ACR+16] Ahmed El Alaoui, Xiang Cheng, Aaditya Ramdas, Martin J. Wainwright, and Michael I. Jordan. Asymptotic behavior of $_{\mathrm{p}}$-based laplacian regularization in semi-supervised learning. *CoRR*, abs/1603.00564, 2016. 61

[ADWR17] Nesreen K Ahmed, Nick Duffield, Theodore L Willke, and Ryan A Rossi. On sampling from massive graph streams. *PVLDB*, 10(11):1430–1441, 2017. 112

[AG05] Lada A. Adamic and Natalie S. Glance. The political blogosphere and the 2004 U.S. election: divided they blog. In *LinkKDD*, pages 36–43. ACM, 2005. 51, 65

[AMA19] Ghadeer AbuOda, Gianmarco De Francisci Morales, and Ashraf Aboulnaga. Link prediction via higher-order motif features. *CoRR*, abs/1902.06679, 2019. 61

[AMF10] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. oddball: Spotting anomalies in weighted graphs. In *PAKDD*, volume 6119, pages 410–421. Springer, 2010. 87, 110

[ANK14] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *TKDD*, 8(2):7, 2014. 112

[APK+19] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 21–29. PMLR, 2019. 57, 60, 62

[ATK15] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.*, 29(3):626–688, 2015. 87, 110

[Ayr62] Frank Ayres. *Schaum's outline of theory and problems of matrices*. McGraw-Hill, 1962. 19

[AZY11] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. Outlier detection in graph streams. In *ICDE*, pages 399–409. IEEE, 2011. 87, 98, 108, 110, 111, 123

[B+08] Shumeet Baluja et al. Video suggestion and discovery for youtube: Taking random walks through the view graph. In *WWW*, pages 895–904, 2008. 55, 56

[BAS+18] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon M. Kleinberg. Simplicial closure and higher-order link prediction. *Proc. Natl. Acad. Sci. U.S.A.*, 115(48):E11221–E11230, 2018. 61

[BGHS12] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *SIGKDD*, pages 1258–1266, 2012. 16

[BGL16] Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016. 61

[BMN04] Mikhail Belkin, Irina Matveeva, and Partha Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT*, volume 3120 of *Lecture Notes in Computer Science*, pages 624–638. Springer, 2004. 57, 60

[BXG+13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, pages 119–130. ACM, 2013. 86, 87, 108

[CG97] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997. 72

[Cha04] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, volume 3202 of *LNCS*, pages 112–124. Springer, 2004. 110

[CKHF11] Duen Horng Chau, Aniket Kittur, Jason I Hong, and Christos Faloutsos. Apolo: making sense of large network data by combining rich user interaction and machine learning. In *ACM SIGCHI*, pages 167–176, 2011. 16, 42, 43

[CLG+15] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *KDD*, pages 1721–1730. ACM, 2015. 132, 134

[CSSS17] Edward Choi, Andy Schuetz, Walter F. Stewart, and Jimeng Sun. Using recurrent neural network models for early detection of heart failure onset. *JAMIA*, 24(2):361–370, 2017. 134, 137

[CSZ06a] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Transductive Inference and Semi-Supervised Learning*. MIT Press, 2006. 21, 55

[CSZ+06b] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. Semi-supervised learning. 2006. 16

[dbl14] Dblp network dataset. `http://konect.uni-koblenz.de/networks/dblp_coauthor`, 2014. 122

[DBS18] Maximilien Danisch, Oana Denisa Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *WWW*, pages 589–598. ACM, 2018. 61, 66, 73

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. 89

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–22, 1977. 141

[DS16] Kirill Dyagilev and Suchi Saria. Learning (predictive) risk scores in the presence of censoring due to interventions. *Machine Learning*, 102(3):323–348, 2016. 132, 134, 145

[EF18] Dhivya Eswaran and Christos Faloutsos. Sedanspot: Detecting anomalies in edge streams. In *ICDM*, pages 953–958. IEEE Computer Society, 2018. 4, 6, 107, 156

[EFGM18] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *KDD*, pages 1378–1386. ACM, 2018. 4, 6, 85, 155

[EFMN19] Dhivya Eswaran, Christos Faloutsos, Nina Mishra, and Yonatan Naamad. Intervention-aware early warning. In *ICDM*, pages 953–958. IEEE Computer Society, 2019. 4, 6, 131, 156

[EGF17a] Dhivya Eswaran, Stephan Günnemann, and Christos Faloutsos. The power of certainty: A dirichlet-multinomial model for belief propagation. In *SDM*, pages 144–152. SIAM, 2017. 4, 41, 155

[EGF+17b] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. Zoobp: Belief propagation for heterogeneous networks. *PVLDB*, 10(5):625–636, 2017. 4, 13, 55, 58, 87, 110, 155

[EJL+18] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *WWW*, pages 1775–1784. ACM, 2018. 111

[EKF20] Dhivya Eswaran, Srijan Kumar, and Christos Faloutsos. Higher order label homogeneity and spreading in graphs. *TheWebConf*, 10(5):625–636, 2020. 4, 57, 155

[EMK06] Gal Elidan, Ian McGraw, and Daphne Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. *UAI*, pages 165–173, 2006. 16, 17

[ES06] Pavlos S. Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006. 112, 114, 118

[FH06] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *IJCV*, pages 41–54, 2006. 16, 43

[FHC12] Yuan Fang, Bo-June Paul Hsu, and Kevin Chen-Chuan Chang. Confidence-aware graph regularization with heterogeneous pairwise features. In *SIGIR*, pages 951–960, 2012. 55, 56

[FHH+17] Joseph Futoma, Sanjay Hariharan, Katherine A. Heller, Mark Sendak, Nathan Brajer, Meredith Clement, Armando Bedoya, and Cara O'Brien. An improved multi-output gaussian process RNN with real-time validation for early sepsis detection. In *MLHC*, volume 68, pages 243–254. PMLR, 2017. 134, 137

[FK96] Brendan J. Frey and Frank R. Kschischang. Probability propagation and iterative decoding. In *Allerton Conference on Communications, Control and Computing*, pages 482–493, 1996. 16

[FMI] Marc PC Fossorier, Miodrag Mihaljevic, and Hideki Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on communications*, pages 673–680. 16

[FNG14] Timothy La Fond, Jennifer Neville, and Brian Gallagher. Anomaly detection in dynamic networks of varying size. *CoRR*, abs/1411.3749, 2014. 85, 87

[FYZ+19] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *AAAI*, pages 3558–3565. AAAI Press, 2019. 73

[Gat15] Wolfgang Gatterbauer. The linearization of pairwise markov networks. *arXiv preprint arXiv:1502.04956*, 2015. 16, 17, 40

[GGF14a] Nikou Günnemann, Stephan Günnemann, and Christos Faloutsos. Robust multivariate autoregression for anomaly detection in dynamic product ratings. In *WWW*, pages 361–372. ACM, 2014. 108

[GGF14b] Stephan Günnemann, Nikou Günnemann, and Christos Faloutsos. Detecting anomalies in dynamic rating data: a robust probabilistic model for rating evolution. In *KDD*, pages 841–850, 2014. 54

[GGKF15] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and single-pass belief propagation. *PVLDB*, 8(5):581–592, 2015. 15, 16, 17, 20, 21, 40, 54, 55, 56

[GGSH12] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. Integrating community matching and outlier detection for mining evolutionary community outliers. In *KDD*, pages 859–867. ACM, 2012. 110

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864. ACM, 2016. 61, 74, 75

[GLL+17] Yifeng Gao, Qingzhe Li, Xiaosheng Li, Jessica Lin, and Huzefa Rangwala. Trajviz: A tool for visualizing patterns and anomalies in trajectory. In *ECML/PKDD (3)*, volume 10536, pages 428–431. Springer, 2017. 134

[GMRS16] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *ICML*, volume 48, pages 2712–2721. JMLR.org, 2016. 88, 91, 97, 98

[Hav03] Taher H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *TKDE*, 15(4):784–796, 2003. 110

[Haw80] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980. 3

[HEF+10] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B. Aditya Prakash, and Hanghang Tong. Metric forensics: a multi-level approach for mining volatile graphs. In *KDD*, pages 163–172. ACM, 2010. 85, 87, 88

[HHPS15] Katharine E Henry, David N Hager, Peter J Pronovost, and Suchi Saria. A targeted real-time early warning score (trewscore) for septic shock. *Science translational medicine*, 7(299):299ra122–299ra122, 2015. 134, 137

[HR05] Robert A Hanneman and Mark Riddle. Introduction to social network methods, 2005. 58, 66

[HS81] Harold V Henderson and Shayle R Searle. The vec-permutation matrix, the vec operator and kronecker products: A review. *Linear and multilinear algebra*, pages 271–288, 1981. 18, 19, 51

[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 *SDM*, pages 495–503. SIAM, 2016. 54, 110

[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 *KDD*, pages 895–904. ACM, 2016. 87

[IK04] Tsuyoshi Idé and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *KDD*, pages 440–449. ACM, 2004. 87, 88

[Jac10] Matthew O Jackson. *Social and economic networks*. Princeton university press, 2010. 58, 66

[JBC+15] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A general suspiciousness metric for dense blocks in multimodal data. In *ICDM*, pages 781–786. IEEE, 2015. 87

[JNG04] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *KDD*, pages 593–598. ACM, 2004. 16

[Joa99] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *ICML*, pages 200–209. Morgan Kaufmann, 1999. 61, 74, 75

[JPS+16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016. 148

[JRBT12] Matthew O Jackson, Tomas Rodriguez-Barraquer, and Xu Tan. Social capital and social quilts: Network patterns of favor exchange. *American Economic Review*, 102(5):1857–97, 2012. 58, 66

[JS17] Shweta Jain and C. Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *WWW*, pages 441–449. ACM, 2017. 61

[JS20] Shweta Jain and C. Seshadhri. The power of pivoting for exact clique counting. In *WSDM*, pages 268–276. ACM, 2020. 80

[Kat53] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. 110

[KB09] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. 110

[KKK+11] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *ECML PKDD*, pages 245–260, 2011. 15, 16, 17, 20, 40, 55, 110

[KLFH06] Eamonn J. Keogh, Jessica Lin, Ada Wai-Chee Fu, and Helga Van Herle. Finding unusual medical time-series subsequences: Algorithms and applications. *IEEE Trans. Information Technology in Biomedicine*, 10(3):429–439, 2006. 134

[KSV+16] Danai Koutra, Neil Shah, Joshua T. Vogelstein, Brian Gallagher, and Christos Faloutsos. Deltacon: Principled massive-graph similarity function with attribution. *TKDD*, 10(3):28:1–28:43, 2016. 87, 88, 92, 110, 111

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017. 57, 60, 62, 74, 75

[LCF+99] Richard Lippmann, Robert K. Cunningham, David J. Fried, Isaac Graf, Kris R. Kendall, Seth E. Webster, and Marc A. Zissman. Results of the DARPA 1998 offline intrusion detection evaluation. In *Recent Advances in Intrusion Detection*, 1999. 97, 121

[Lip18] Zachary C. Lipton. The mythos of model interpretability. *Commun. ACM*, 61(10):36–43, 2018. 132

[LK03] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559. ACM, 2003. 110

[LKF07] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1):2, 2007. 65

[LL10] Weiping Liu and Linyuan Lü. Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007, 2010. 111, 124

[LM04] Amy N Langville and Carl D Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2004. 123, 124

[LRHB06] Xiangyang Lan, Stefan Roth, Daniel P. Huttenlocher, and Michael J. Black. Efficient belief propagation with learned higher-order markov random fields. In *ECCV (2)*, volume 3952 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2006. 61

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *ICDM*, pages 413–422. IEEE, 2008. 88

[LW89] Danyu Y Lin and Lee-Jen Wei. The robust inference for the cox proportional hazards model. *Journal of the American statistical Association*, 84(408):1074–1078, 1989. 144

[McG14] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014. 88

[MK07] Joris M Mooij and Hilbert J Kappen. Sufficient conditions for convergence of the sum–product algorithm. *IEEE Transactions on Information Theory*, pages 4422–4437,

2007. 16, 43, 55

[MMA16] Emaad A. Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *KDD*, pages 1035–1044. ACM, 2016. 87, 110

[MSLC01] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001. 58, 66

[MTVV15] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In *MFCS*, volume 9235, pages 472–482. Springer, 2015. 88

[MWJ99] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, pages 467–475, 1999. 16, 43

[New03] Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003. 65

[NHR+18] Shamim Nemati, Andre Holder, Fereshteh Razmi, Matthew D Stanley, Gari D Clifford, and Timothy G Buchman. An interpretable machine learning model for accurate prediction of sepsis in the icu. *Critical care medicine*, 46(4):547–553, 2018. 134, 137

[NJ00] Jennifer Neville and David Jensen. Iterative classification in relational data. In *AAAI Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000. 16

[NSZ09] Boaz Nadler, Nathan Srebro, and Xueyuan Zhou. Statistical analysis of semi-supervised learning: The limit of infinite unlabelled data. In *NIPS*, pages 1330–1338. Curran Associates, Inc., 2009. 61

[nyt18] Nyc taxi & limousine corporation - trip record data. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`, 2018. 97

[OC12] Matan Orbach and Koby Crammer. Graph-based transduction with confidence. In *ECMLPKDD*, pages 323–338. Springer, 2012. 55, 56

[PAS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, pages 701–710. ACM, 2014. 61

[PCWF07] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*, pages 201–210, 2007. 16, 43

[Pea82] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *AAAI*, pages 133–136, 1982. 16

[Pea14] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014. 16, 43

[Pev16] Tomás Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016. 88, 91

[PL08] Brian Potetz and Tai Sing Lee. Efficient belief propagation for higher-order cliques using linear constraint nodes. *Computer Vision and Image Understanding*, 112(1):39–54, 2008. 61

[PNMS13] Chris Paxton, Alexandru Niculescu-Mizil, and Suchi Saria. Developing predictive models using electronic medical records: challenges and pitfalls. In *AMIA Annual Symposium Proceedings*, page 1109, 2013. 132, 134, 137

[PSS+10] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In *PAKDD*, volume 6119, pages 435–448. Springer, 2010. 87

[Rab89] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *IEEE*, 77(2):257–286, 1989. 140, 141

[RGNH13] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM, 2013. 110

[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. 85, 87, 88, 98, 108, 110, 111, 123

[RKC+17] Aniruddh Raghu, Matthieu Komorowski, Leo Anthony Celi, Peter Szolovits, and Marzyeh Ghassemi. Continuous state-space models for optimal sepsis treatment: a deep reinforcement learning approach. In *MLHC*, volume 68, pages 147–163. PMLR, 2017. 135

[RLG+05] Chotirat Ann Ralanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. Mining time series data. In *Data mining and knowledge discovery handbook*, pages 1069–1103. Springer, 2005. 134

[RRK+19] Ryan A. Rossi, Anup Rao, Sungchul Kim, Eunyee Koh, Nesreen K. Ahmed, and Gang Wu. Higher-order ranking and link prediction: From closing triangles to closing higher-order motifs. *CoRR*, abs/1906.05059, 2019. 61

[RSG16] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144. ACM, 2016. 132

[RSK+15] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015. 87, 110

[RTG17] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Finding dynamic dense subgraphs. *TKDD*, 11(3):27:1–27:30, 2017. 110

[SA04] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4(1):120–128, 2004. 97, 121

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. SIAM, 2003. 32, 52, 72

[SB13] Lovro Subelj and Marko Bajec. Model of complex networks based on citation dynamics. In *WWW (Companion Volume)*, pages 527–530. International World Wide Web Conferences Steering Committee / ACM, 2013. 65

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 135, 141

[SBB+09] Marten Scheffer, Jordi Bascompte, William A Brock, Victor Brovkin, Stephen R Carpenter, Vasilis Dakos, Hermann Held, Egbert H Van Nes, Max Rietkerk, and George Sugihara. Early-warning signals for critical transitions. *Nature*, 461(7260):53, 2009. 135

[Sch17] Jacob Schreiber. pomegranate: Fast and flexible probabilistic modeling in python. *JMLR*, 18:164:1–164:6, 2017. 144

[SD14] Kumar Sricharan and Kamalika Das. Localizing anomalous changes in time-evolving graphs. In *SIGMOD*, pages 1347–1358. ACM, 2014. 85, 87, 88, 110, 111

[SDS+16] Mervyn Singer, Clifford S Deutschman, Christopher Warren Seymour, Manu Shankar-Hari, Djillali Annane, Michael Bauer, Rinaldo Bellomo, Gordon R Bernard, Jean-Daniel Chiche, Craig M Coopersmith, et al. The third international consensus definitions for sepsis and septic shock (sepsis-3). *Jama*, 315(8):801–810, 2016. 148, 149

[SERU17] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(4):43, 2017. 112

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696. ACM, 2007. 85, 87, 88, 108, 110

[SGB17] Ann Sizemore, Chad Giusti, and Danielle S Bassett. Classification of weighted networks through mesoscale homological features. *Journal of Complex Networks*, 5(2):245–273, 2017. 58, 66

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948. 62

[SHF16] Kijung Shin, Bryan Hooi, and Christos Faloutsos. M-zoom: Fast dense-block detection in tensors with quality guarantees. In *ECML/PKDD*, volume 9851, pages 264–280. Springer, 2016. 87

[SHF18] Kijung Shin, Bryan Hooi, and Christos Faloutsos. Fast, accurate, and flexible algorithms for dense subtensor mining. *ACM Transactions on Knowledge Discovery from Data*, 12(3):28:1–28:30, 2018. 108, 110

[SHGY09] Yizhou Sun, Jiawei Han, Jing Gao, and Yintao Yu. itopicmodel: Information network-integrated topic modeling. In *ICDM*, pages 493–502. IEEE, 2009. 51

[SNB+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008. 60

[SP10] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010. 127

[SS17] Peter Schulam and Suchi Saria. Reliable decision support using counterfactual models. In *NIPS*, pages 1697–1708, 2017. 134

[ST17] Dejan Slepcev and Matthew Thorpe. Analysis of p-laplacian regularization in semi-supervised learning. *CoRR*, abs/1707.06213, 2017. 61

[STF06] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD*, pages 374–383. ACM, 2006. 85, 87, 88, 98, 110

[SZS03] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Transactions on pattern analysis and machine intelligence*, pages 787–800, 2003. 16

[TC09] Partha Pratim Talukdar and Koby Crammer. New regularized algorithms for transductive learning. In *ECML/PKDD*, pages 442–457, 2009. 55, 56, 57, 60

[TFP06] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622. IEEE, 2006. 16, 111

[TL11] Hanghang Tong and Ching-Yung Lin. Non-negative residual matrix factorization with application to graph anomaly detection. In *SDM*, pages 143–153. SIAM, 2011. 110

[TQW+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. *CoRR*, abs/1503.03578, 2015. 61

[TZ12] Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference & International Workshop Present Day Trends of Innovations*, 2012. 51, 65

[VFMV03] Alexei Vazquez, Alessandro Flammini, Amos Maritan, and Alessandro Vespignani. Global protein function prediction from protein-protein interaction networks. *Nature biotechnology*, 21(6):697, 2003. 11, 58

[Vit85] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985. 112

[VLBB08] Ulrike Von Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, pages 555–586, 2008. 60, 71

[Vos91] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on software engineering*, 17(9):972–975, 1991. 117

[WFLW15] Teng Wang, Chunsheng Victor Fang, Derek Lin, and Shyhtsun Felix Wu. Localizing temporal anomalies in large evolving graphs. In *SDM*, pages 927–935. SIAM, 2015. 110

[WH98] Gary M. Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *KDD*, pages 359–363. AAAI Press, 1998. 134, 144

[WRC08] Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 1168–1175. ACM, 2008. 60

[WS98] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440, 1998. 65, 77

[WSW14] Xiang Wang, David Sontag, and Fei Wang. Unsupervised learning of disease progression models. In *KDD*, pages 85–94. ACM, 2014. 134

[WZF+14] Ke Wu, Kun Zhang, Wei Fan, Andrea Edwards, and Philip S. Yu. Rs-forest: A rapid density estimator for streaming anomaly detection. In *ICDM*, pages 600–609. IEEE, 2014. 88, 91

[YAMW13] Weiren Yu, Charu C Aggarwal, Shuai Ma, and Haixun Wang. On anomalous hotspot discovery in graph streams. In *ICDM*, pages 1271–1276. IEEE, 2013. 85, 87, 88, 108, 110, 111

[YBL18] Hao Yin, Austin R Benson, and Jure Leskovec. Higher-order clustering in networks. *Physical Review E*, 97(5):052306, 2018. 61, 65

[YBZ+17] Carl Yang, Lanxiao Bai, Chao Zhang, Quan Yuan, and Jiawei Han. Bridging collaborative filtering and semi-supervised learning: A neural approach for POI recommendation. In *KDD*, pages 1245–1254. ACM, 2017. 11

[YCS16] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 40–48. JMLR.org, 2016. 57, 60, 62

[YFK15] Yuto Yamaguchi, Christos Faloutsos, and Hiroyuki Kitagawa. Socnl: Bayesian label propagation with confidence. In *PAKDD*, pages 633–645, 2015. 55, 56

[YFK16] Yuto Yamaguchi, Christos Faloutsos, and Hiroyuki Kitagawa. Camlp: Confidence-aware modulated label propagation. In *SDM*, 2016. 16, 17, 22, 49, 55, 56

[YFW03] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, pages 239–269. Morgan Kaufmann Publishers Inc., 2003. 17, 19, 42, 43, 55, 56, 57, 60, 62

[YJK18] Minji Yoon, Woojeong Jin, and U. Kang. Fast and accurate random walk with restart on dynamic graphs with guarantees. In *WWW*, pages 409–418. ACM, 2018. 111

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *SIGIR*, pages 42–49. ACM, 1999. 75, 99, 121, 149

[YM16] Weiren Yu and Julie A. McCann. Random walk with restart over dynamic graphs. In *ICDM*, pages 589–598. IEEE, 2016. 111

[YML+14] Jaewon Yang, Julian J. McAuley, Jure Leskovec, Paea LePendu, and Nigam Shah. Finding progression stages in time-evolving event sequences. In *WWW*, pages 783–

794. ACM, 2014. 134, 141

[YNY+19] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha P. Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In *NeurIPS*, pages 1509–1520, 2019. 73

[YZU+18] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Zachary Zimmerman, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Min. Knowl. Discov.*, 32(1):83–123, 2018. 134

[ZBL+03] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *NIPS*, pages 321–328. MIT Press, 2003. 57, 59, 60, 62, 72, 74, 75

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919. AAAI Press, 2003. 57, 59, 60, 62, 69, 74, 75

[ZHS06] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*, pages 1601–1608. MIT Press, 2006. 73

[Zhu05] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. 16, 55, 56