Explainable Mining of Graphs and Time Series: Algorithms and Applications

Meng-Chieh Lee

October 15, 2024

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christos Faloutsos, Co-Chair
Leman Akoglu, Co-Chair
Geoffrey Gordon
Nina Mishra (Amazon)

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

Keywords: Graph Mining, Random Walks, Belief Propagation, Graph Neural Networks, Linear Graph Neural Networks, Information Theory, Usable Information, Graph Anomaly Detection, Minimum Description Length, Frequent Pattern Mining, Learnable Graph Kernel, Kernel Convolution Networks, Time Series Analysis, Seizure Detection, Anomaly Detection, Group Anomaly Detection, Time Series Anomaly Detection, Time Series Foundation Models, Self-Supervised Learning, Automatic Machine Learning, Human-Trafficking Detection, Temporal Graphs, Financial Graphs, Fraud Detection, Large Language Models, Large Language Model Agent, Retrieval Augmented Generation, Graph Retrieval Augmented Generation, Knowledge Graphs

Abstract

Given a social network, how can we predict the connections between users and determine whether they are based on shared hobbies or common friends? Similarly, how can we identify anomalies in time series data and explain why they are suspicious? Although recent machine learning models with improved performance are being developed, they are often black-box methods that are difficult to interpret. This leads us to explainable artificial intelligence (XAI), which offers valuable insights through its explanations. *In this thesis proposal, each method we proposed is either inherently explainable, or provides explanations for the data or decisions it makes.*

In the first part of graph mining, we focus on node-level tasks. We propose algorithms to analyze various types of graph information, e.g., the network effects of the graph structure, and the usable information in the node features. Our proposed linear methods are not only inherently interpretable and fast, but also outperform baselines in solving node classification and link prediction tasks. In node classification, our method improves the accuracy by *10.3%* over the second-best baseline, while being *2.5 times* faster. In link prediction, our method achieves an *average rank 1.1*, outperforming baselines on *11 out of 12* real-world datasets.

In the second part of graph mining, we focus on graph-level tasks. We discover frequent substructures using the minimum description length (MDL) principle and learnable graph kernels. In graph anomaly detection, our MDL-based method is up to *58 times* faster than the second-best baseline, while achieving *1.3 times* higher average precision. In graph regression, our method with learnable graph kernels improves the mean absolute error by *14.3%*.

For time series mining, we primarily focus on anomaly detection with applications that include medical signals (EEG) and sensor signals. Unlike traditional methods that focus on point anomalies, our algorithm focuses on group anomalies. Moreover, our algorithm is fast and scalable, and discovers and ranks both point and group anomalies in *2 minutes for 1 million* data points on a stock machine. Next, our model that leverages self-supervised learning effectively identifies the ground truth hyperparameters of anomalies in the time series, resulting in an *average rank 2.2* compared to the baselines.

Finally, we include several impactful real-world applications that leverage graph algorithms, such as human trafficking detection. Our method detects human-trafficking ads with *84%* precision, while requiring only *8 hours to process 4 million* documents.

Contents

Chapter 1

Introduction

1.1 Motivation

In the last decade, many effective machine learning (ML) and deep learning methods have been proposed to solve a variety of problems in graphs and time series. However, the majority of the methods are designed to optimize performance, often neglecting the importance of model transparency. In other words, these black-box methods are neither inherently explainable nor do they provide explanations for their decisions.

For this reason, in recent years, explainable artificial intelligence (XAI) has gained a lot of attention. These approaches are designed not only to provide explanations but also to remain effective. XAI paves the way for ML methods to be adopted in the real world, especially in domains that require well-justified solutions. These domains include legal, medical, financial, and so on. For example, if an ML method is developed to assist the doctor making medical decisions, the doctor will need it to provide explanations. It is crucial that the method and the doctor complement each other by allowing the doctor to understand why the decision was made.

Among the many types of data, graph and time series are two of the most common. Graphs, including social networks, financial transaction graphs, and product co-purchasing networks, have been applied to numerous real-world applications. Similarly, time series data has been widely employed for monitoring in many systems, such as server machine metrics, IoT for drinking water, and EEG recordings. This brings us to the critical topic of detecting anomalies in time series.

1.2 Contribution

In this proposal, we specifically focus on introducing explainable ML methods tailored for graphs and time series. *Our carefully designed methods are either inherently explainable, such as linear methods, or provide explanations for either the dataset or the decision made by our method.* They solve a wide range of real-world problems and applications. In summary, our contributions can be diveded into four parts:

- **Graph Mining: Node-Level Tasks.** In Chapter 2.2, we propose a statistical test to identify whether there are network effects in the given graph (i.e. homophily, heterophily, both or none). Thanks to our estimated compatibility matrix, our method is 12.9% more accurate and $3.4\times$ faster for node classification. In Chapter 2.3, we further consider the node features and propose a linear graph neural network (GNN) that effectively classifies the nodes in both homophily and

heterophily graphs. Compared to non-linear GNNs, our method is explainable, while being 10.3% more accurate and $2.5\times$ faster. In Chapter 2.4, we extend this linear GNN to measure and exploit usable information in graphs for both node classification and link prediction. Our method achieves an average rank of 1.1 among the state-of-the-art baselines in link prediction.

- **Graph Mining: Graph-Level Tasks.** In Chapter 3.2, we propose a minimum description length (MDL) method that identifies anomalous graphs in a database containing many graphs, based on frequent substructure mining. Our method is up to $58\times$ faster, while being $1.3\times$ better in average precision. In Chapter 3.3, we further explore the frequent substructure by learning with graph kernels and propose a novel GNN layer. Applied on a large graph regression benchmark, our GNN layer outperforms the baseline by 14.3% better in mean absolute error.
- **Time Series Mining.** In Chapter 4.2, we propose an algorithm to detect group anomalies (such as seizures) and point anomalies (such as noise) in time series EEG signals. Our method is the first to detect both types of anomaly and takes only 2 minutes to run on 1 million data points. In Chapter 4.3, we introduce a method that automatically fine-tunes the best hyperparameters for creating pseudo-time-series anomalies, to learn the anomaly detector in a self-supervised manner. Our method achieves an average rank 2.2 in F1 score. In Chapter 4.4, we further propose to extend this method to search for hyperparameters for multiple types of anomalies simultaneously.
- **Applications.** In Chapter 5.2, we propose an algorithm that incrementally detects templates among text documents by representing them as line graphs. Our method detects human-trafficking ads with 84% precision, while requiring only 8 hours for 4 million documents. In Chapter 5.3, we propose to design a graph algorithm to detect fraudulent transactions on temporal financial graphs. In Chapter 5.4, we propose to develop a retrieval-augmented generation (RAG) framework designed for use with a database containing both text documents and knowledge graphs.

1.3 Organization

The rest of the thesis proposal is organized as follows. In Chapter 2, we present explainable graph algorithms for node-level tasks, solving node classification and link prediction. In Chapter 3, we address graph-level tasks by leveraging the frequent substructures in the graph database. In Chapter 4, we propose several algorithms for time series anomaly detection, including an application in seizure detection. In Chapter 5, we present various explainable methods that utilize graphs to solve real-world applications. In Appendix A and B, we introduce the related works and the datasets used in the thesis. A summary of the organization can be found in Table 1.1.

In each chapter we follow the same organization: *goal*, *approach*, and *results*.

	Node-Level	**Graph-Level**	**Time Series**
Algorithms	Chapter 2.2, 2.3, 2.4	Chapter 3.2, 3.3	Chapter 4.3, 4.4
Applications	Chapter 5.3, 5.4	Chapter 5.2	Chapter 4.2

Table 1.1: **Thesis organization.**

Chapter 2

Graph Mining: Node-Level Tasks

2.1 Preliminaries

Homogeneous Graphs A homogeneous graph is a data structure that consists of nodes and edges, where both the nodes and edges are of the same type. The structure of a graph containing n nodes can be represented by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, where in the matrix, 1 indicates the presence of an edge between two nodes, and 0 indicates no edge. Each node i has a unique label $l(i) \in \{1, \ldots, c\}$, where c is the number of classes. The set of labels $\mathbf{y} = \{l(1), \ldots, l(n)\}$ can be represented by a one-hot matrix $\mathbf{Y} \in \mathbb{R}^{n \times c}$. Graphs that contain node features or attributes $\mathbf{X} \in \mathbb{R}^{n \times f}$, where f is the number of features, are called node-attributed graphs.

Homophily and Heterophily A graph with network effects provides meaningful information through its structure, which can be used to identify node labels or connections between nodes. For example, "talkative people tend to make friends with other talkative people" denotes homophily, while "teenagers are inclined to interact with those of the opposite gender on social networks" denotes heterophily. Some graphs may exhibit either one or both, while others may exhibit neither, meaning there are no network effects.

Message Passing Methods Message-passing methods utilize the graph structure to propagate information from the neighbors of a node to the node itself. Known as sum-product message passing, belief propagation methods [28, 44, 93, 94] directly perform inference on the graph through several propagation iterations. Although they are fast and effective because they require neither parameters nor training, belief propagation methods are mainly designed to solve node classification problems based solely on the graph structure and usually do not consider node features.

Another variety of message passing methods, graph neural networks (GNNs) [32, 40, 59, 91], are a class of deep learning models. They are commonly used to generate low-dimensional embeddings of nodes to perform graph tasks by propagating node features through structure and learning end-to-end with a training objective. Some studies target interpretable models and remove the non-linear functions in GNNs, while still achieving good performance, which we call linear GNNs [58, 97, 98, 118]. One of the many advantages of linear GNNs is that their node embeddings are available prior to model training.

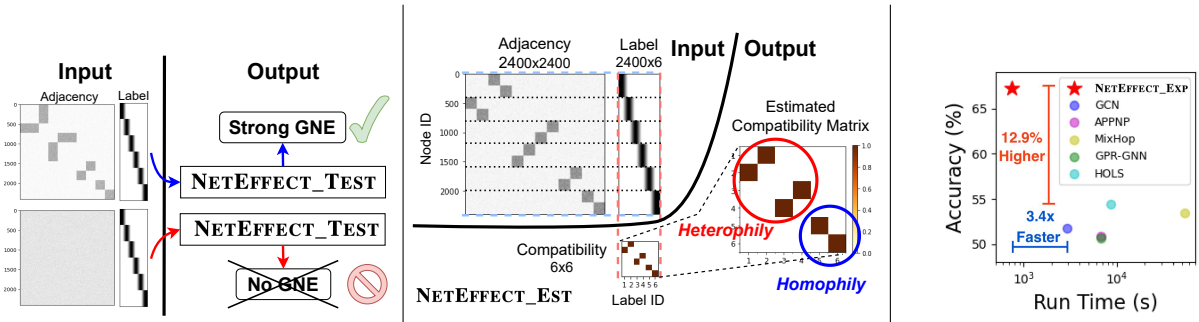
2.2 NETEFFECT: Discovery and Exploitation of Generalized Network Effects

Section based on work that appeared at PAKDD 2024 [50][PDF].

2.2.1 Goal

Given a large graph with few node labels and no node features, how can we determine whether the graph structure is useful for classifying nodes? The graph structure may not provide meaningful information for inference tasks, so a preliminary test is needed. That is to say, we want to know whether the given graph has generalized network-effects (GNE) or not, and to distinguish which GNE the graph has, i.e., homophily, heterophily, or both (which we call "x-ophily"), if there is any. Furthermore, we aim to exploit GNE for better and faster node classification.

Problem 1. Generalized Network-Effects (GNE)
(1) ***Given*** *a graph without node attributes and with a few node labels.*
(2) ***Find****:*
- *(a)* **Hypothesis Testing***: How to identify whether the graph has GNE or not?*
- *(b)* **Estimation***: How to estimate the type of GNE in a principled way?*
- *(c)* **Exploitation***: How to efficiently exploit GNE in node classification?*



(a) NETEFFECT_TEST: Principled.

(b) NETEFFECT_EST: Explainable and General.

(c) NETEFFECT_EXP: Accurate and Scalable.

Figure 2.1: **NETEFFECT works well**, thanks to its three novel contributions: (a) NETEFFECT_TEST statistically tests the existence of GNE. (b) NETEFFECT_EST explains the graph with the x-ophily compatibility matrix. (c) NETEFFECT_EXP wins in node classification and is fast.

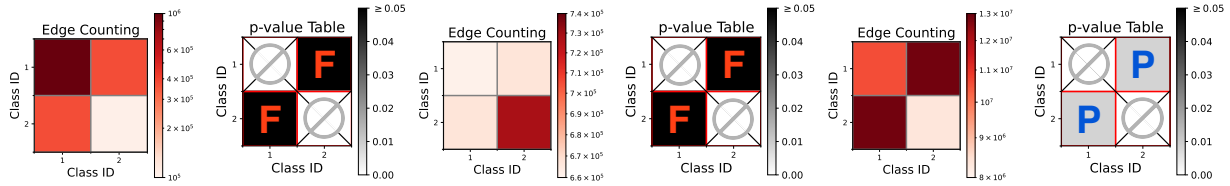
2.2.2 Approach

We propose NETEFFECT, with 3 contributions as the corresponding solutions:

1. **NETEFFECT_TEST** uses statistical tests to verify whether GNE exists at all. Figure 2.1a shows how it works, and Figure 2.2 shows its discovery, where many large real-world datasets known as heterophily graphs have little GNE.
2. **NETEFFECT_EST** explains if the graph is homophily, heterophily, or x-ophily by estimating the $c \times c$ compatibility matrix with the derived closed-form formula. In Figure 2.1b, it explains the interrelations of classes by the estimated compatibility matrix, which implies x-ophily.
3. **NETEFFECT_EXP** efficiently exploits GNE to perform better in node classification. It wins in both accuracy and time on a million-scale heterophily graph "Pokec-Gender", only requiring 14 *minutes* (Figure 2.1c).

NETEFFECT_TEST We use Pearson's χ^2 test to decide whether class c_i (say, "talkative people"), has statistically more, or fewer edges to class c_j (say, "silent people"). Specifically, given a class pair (c_i, c_j), the input to the test is a 2×2 contingency table that contains the counts of edges that connect pairs of nodes whose labels are in $\{c_i, c_j\}$. The null hypothesis of the test is: *"Edges are equally likely to exist between nodes of the same class and those of different classes."* If the p value of the test is not less than 0.05, we accept the null hypothesis, which represents that the chosen class pair (c_i, c_j) exhibits no statistically significant GNE in the graph.

NETEFFECT_TEST identifies the lack of GNE in "Genius" and "Penn94" datasets, which are widely known as heterophily graphs. In "Genius" (Figure 2.2a), we see that both classes 1 and 2 tend to connect to class 1, making class 2 indistinguishable by graph structure. NETEFFECT_TEST thus accepts the null hypothesis, and identifies the lack of GNE. We can observe a similar phenomenon in "Penn94" (Figure 2.2b). On the other hand, NETEFFECT_TEST identifies the strong GNE in "Pokec-Gender" dataset, which exhibit heterophily (Figure 2.2c).



(a) "Genius": No GNE (b) "Penn94": No GNE (c) "Pokec-Gender": Strong GNE

Figure 2.2: **NETEFFECT_TEST works**, discovering that real-world heterophily graphs have little GNE. For each graph, we report the edge counting on the left (not available in practice), and the p-value table from NETEFFECT_TEST on the right, where "P" denotes the presence of GNE, and "F" denotes the absence.

NETEFFECT_EST A compatibility matrix $\mathbf{H}_{c\times c}$ is a row-normalized matrix, where $\mathbf{H}_{ku}$ is the relative influence of class k on class u. It is a natural strategy to explain the network-effects, and is commonly used in belief propagation. To estimate $\hat{\mathbf{H}}$, we turn the compatibility matrix estimation into an optimization problem and solve it with the proposed closed-form formula:

Lemma 1 (Network Effect Formula (NEF))**.** *Given adjacency matrix* $\mathbf{A}$ *and initial beliefs* $\hat{\mathbf{E}}$*, the closed-form solution of vectorized compatibility matrix* $vec(\hat{\mathbf{H}})$ *is:*

$$\boxed{vec(\hat{\mathbf{H}}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}} \tag{2.1}$$

where $\mathbf{X} = \mathbf{I}_{c\times c} \otimes (\mathbf{A}\hat{\mathbf{E}})$ *and* $\mathbf{y} = vec(\hat{\mathbf{E}})$.

The proof is in [50]. $\hat{\cdot}$ denotes the matrix is centered around $1/c$ to ensure the convergence of belief propagation. With few observed labels, NETEFFECT_EST uses ridge regression with leave-one-out cross-validation (RidgeCV) to solve NEF. It is noteworthy that its computational cost is negligible as well, and it requires no training.

NETEFFECT_EXP With only a few labels, it becomes crucial to better utilizing the graph structure for node classification. We propose to pay attention to influential neighbors using our proposed "emphasis" matrix $\mathbf{A}^*$, i.e. a weighted adjacency matrix. The influential neighbors are identified as those that are connected by higher-order structures, e.g., cliques. NETEFFECT_EXP computes the final beliefs iteratively by aggregating the beliefs of neighbors through $\mathbf{A}^*$ until they converge.

2.2.3 Results

In Table 2.1, NETEFFECT outperforms all competitors in accuracy and running time. In graphs exhibiting strong GNE, namely "Synthetic" and "Pokec-Gender", NETEFFECT outperforms competitors significantly by more than 34.3% and 12.9% accuracy owing to precise estimations of the compatibility matrix. In Table 2.2, NETEFFECT-Hom outperforms all the competitors on 3 out of 4 homophily graphs, and NETEFFECT performs similarly to NETEFFECT-Hom, which uses an identity matrix as the compatibility matrix. NETEFFECT correctly estimates a near-identity compatibility matrix in homophily graphs. It is also the fastest algorithm among all baselines. For example, in arXiv-Year, NETEFFECT is $57.4\times$ faster than MixHop.

Table 2.1: **NETEFFECT wins on x-ophily and Heterophily datasets.**

Dataset	**Synthetic**			**Pokec-Gender**			**arXiv-Year**			**Patent-Year**		
GNE Strength	Strong x-ophily			Strong Heterophily			Weak x-ophily			Weak Heterophily		
Method	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time
GCN	16.7±0.0	3456	4.1×	51.8±0.1	2906	3.4×	35.3±0.1	132	2.5×	26.0±0.0	894	2.3×
APPNP	18.6±1.1	7705	9.2×	50.9±0.3	6770	7.8×	33.5±0.2	423	8.1×	27.5±0.2	2050	5.2×
MixHop	16.7±0.0	58391	70.0×	53.4±1.2	53871	62.1×	39.6±0.1	2983	57.4×	26.8±0.1	18787	47.6×
GPRGNN	18.9±1.2	7637	9.1×	50.7±0.2	6699	7.7×	30.1±1.4	400	7.7×	25.3±0.1	2034	5.1×
HOLS	46.1±0.1	1672	2.0×	54.4±0.1	8552	9.9×	34.1±0.3	566	10.9×	23.6±0.0	510	1.3×
NETEFFECT-Hom	45.6±0.1	835	1.0×	56.9±0.2	869	1.0×	37.0±0.3	52	1.0×	24.3±0.0	429	1.1×
NETEFFECT	80.4±0.0	841	1.0×	67.3±0.1	867	1.0×	38.9±0.1	52	1.0×	28.7±0.1	395	1.0×

Table 2.2: **NETEFFECT wins on Homophily datasets.**

Dataset	**Facebook**			**GitHub**			**arXiv-Category**			**Pokec-Locality**		
Method	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time	Accuracy	Time (s)	Rel. Time
GCN	67.0±0.8	12	2.0×	81.0±0.6	28	2.2×	25.4±0.3	216	2.3×	17.3±0.4	4002	2.9×
APPNP	50.5±2.2	46	7.7×	74.2±0.0	73	5.6×	19.4±0.6	1176	12.3×	16.8±1.7	11885	8.6×
MixHop	69.2±0.7	296	49.3×	77.8±1.3	526	40.5×	33.0±0.6	3203	33.4×	16.9±0.3	52139	37.9×
GPRGNN	51.9±1.5	47	7.8×	74.1±0.1	75	5.8×	19.7±0.3	1174	12.2×	30.0±2.0	11959	8.7×
HOLS	86.0±0.4	934	155.7×	80.8±0.5	126	9.7×	61.4±0.2	627	6.5×	63.7±0.3	8139	5.9×
NETEFFECT-Hom	85.2±0.5	6	1.0×	81.3±0.5	13	1.0×	61.7±0.2	96	1.0×	66.0±0.2	1437	1.0×
NETEFFECT	85.2±0.5	6	1.0×	81.3±0.5	13	1.0×	58.8±0.6	108	1.1×	64.8±0.8	1377	1.0×

2.3 SLIMG: Accurate, Robust, and Interpretable Graph Mining

Section based on work that appeared at KDD 2023 [110][PDF].

2.3.1 Goal

How can we solve semi-supervised node classification in node-attributed graphs fast and accurately? How can we handle various scenarios of graphs, possibly with noisy features and structures? In these noisy scenarios, graph neural networks (GNNs) [32, 40, 92] easily overfit their parameters to such noise. In Chapter 2.2, we study the network effects in a graph without node features, and now extend the idea to the node-attributed graph. We want to tackle node classification in these real-world graph scenarios, including useless features and useless graph structure (no network effects). Moreover, we want to interpret the graph data by pointing out the information that is useful for identifying the class of a node, e.g., node features, graph structure, or both.

Problem 2. Semi-Supervised Node Classification
(1) ***Given*** *an undirected graph* $G = (\mathbf{A}, \mathbf{X})$ *and labels* $\mathbf{y}$ *of* m *nodes, where* $m \ll n$.
(2) ***Predict*** *the unknown classes of* $n - m$ *testing nodes.*
(3) ***Identify*** *the graph information that is useful for the prediction.*

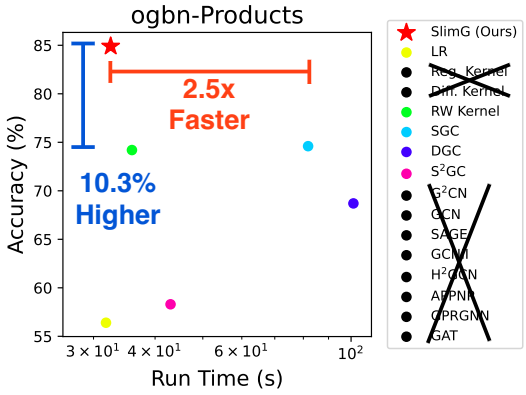
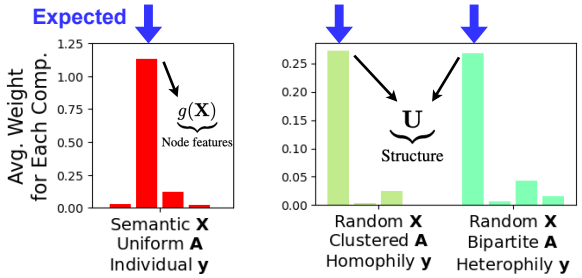


Figure 2.3: **SLIMG wins both on accuracy and training time** on ogbn-Products with 61.9M edges. Several baselines run out of memory (crossed out).



(a) No Network Effects (b) Useless Features

Figure 2.4: **SLIMG is interpretable**: it suppresses useless information and focuses on useful information for each scenario: (a) self-features $g(\mathbf{X})$ and (b) structural features $\mathbf{U}$. The learned weights are directly matched with the expectations.

2.3.2 Approach

We propose SLIMG, a linear GNN model based on the *careful simplicity* principle: a simple, carefully-designed model can be more accurate than complex ones thanks to better generalizability,

Table 2.3: **GNNEXP encompasses popular GNNs.** The * and ** superscripts mark fully and partially linearized models, respectively. We derive pain points and distinguishing factors through GNNEXP.

Model	Type	Propagator function $\mathcal{P}(\mathbf{A}, \mathbf{X})$
LR	Linear	$\mathbf{X}$
SGC	Linear	$\tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}$
DGC	Linear	$[(1-T/K)\mathbf{I} + (T/K)\tilde{\mathbf{A}}_{\text{sym}}]^{K}\mathbf{X}$
S^2GC	Linear	$\sum_{k=1}^{K}(\alpha\mathbf{I} + (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{k})\mathbf{X}$
G^2CN	Linear	$\|_{i=1}^{N}[\mathbf{I} - (T_i/K)((b_i-1)\mathbf{I} + \mathbf{A}_{\text{sym}})^{2}]^{K}\mathbf{X}$
PPNP*	Decoupled	$(\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}})^{-1}\mathbf{X}$
APPNP*	Decoupled	$[\sum_{k=0}^{K-1}\alpha(1-\alpha)^{k}\tilde{\mathbf{A}}_{\text{sym}}^{k} + (1-\alpha)^{K}\tilde{\mathbf{A}}_{\text{sym}}^{K}]\mathbf{X}$
GDC*	Decoupled	$\mathbf{S} = \text{sparse}_{\epsilon}(\sum_{k=0}^{\infty}(1-\alpha)^{k}\tilde{\mathbf{A}}_{\text{sym}}^{k})$ for $\mathbf{S}_{\text{sym}}\mathbf{X}$
GPR-GNN*	Decoupled	$\|_{k=0}^{K}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$

Model	Type	Propagator function $\mathcal{P}(\mathbf{A}, \mathbf{X})$
ChebNet*	Coupled	$\|_{k=0}^{K-1}\mathbf{A}_{\text{sym}}^{k}\mathbf{X}$
GCN*	Coupled	$\tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}$
SAGE*	Coupled	$\|_{k=0}^{K}\mathbf{A}_{\text{row}}^{k}\mathbf{X}$
GCNII*	Coupled	$\|_{k=0}^{K-2}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X} \,\|\, ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{K} + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}$
H_2GCN*	Coupled	$\|_{k=0}^{2K}\mathbf{A}_{\text{sym}}^{k}\mathbf{X}$
GAT**	Attention	$\prod_{k=1}^{K}[\text{diag}(\mathbf{X}\mathbf{w}_{k,1})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{w}_{k,2})]\mathbf{X}$
DA-GNN**	Attention	$\sum_{k=0}^{K}\text{diag}(\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}\mathbf{w})\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$

robustness, and easier training. The design decisions of SLIMG are made to follow this principle by observing and addressing the pain points of existing GNNs, analyzed by our framework, GNNEXP.

GNNEXP Why do GNNs work well when they do? In what scenarios might a GNN not work well? We propose GNNEXP, a framework that derives the *feature propagator* function of GNN models by ignoring nonlinearity, so that they are comparable. GNNEXP represents the characteristic of a GNN as a linear feature propagator function $\mathcal{P}$, which transforms the node features $\mathbf{X}$ by the graph structure $\mathbf{A}$. Based on the linearization in Table 2.3, we derive 4 *pain points* of existing GNNs:

1. **Lack of robustness**: All models do not handle multiple graph scenarios at the same time, i.e., graphs with homophily, heterophily, no network effects, or useless features.
2. **Vulnerability to noisy features**: All models cannot fully exploit the graph structure if the features are noisy, as they depend on the node feature matrix $\mathbf{X}$.
3. **Efficiency and effectiveness**: Concatenation-based models create spurious correlations between feature elements, requiring more parameters than in other models.
4. **Many hyperparameters**: Hyperparameters in $\mathcal{P}$ impair its interpretability and require extensive tuning.

What potential choices do we have in designing a general approach that addresses the pain points? We analyze the GNN variants and point out 3 *distinguishing factors*:

1. **Combination of features**: How should we combine node features, the immediate neighbors' features, and the K-step-away neighbors' features?
2. **Modification of $\mathbf{A}$**: How should we normalize or modify the adjacency matrix $\mathbf{A}$?
3. **Heterophily**: What to do if the direct neighbors differ in their features or labels?

SLIMG We propose SLIMG, which addresses the pain points of existing GNN models with strict adherence to the careful simplicity principle. SLIMG has the following propagator function:

$$\boxed{\mathcal{P}(\mathbf{A}, \mathbf{X}) = \underbrace{\mathbf{U}}_{\text{Structure}} \,\|\, \underbrace{g(\mathbf{X})}_{\text{Features}} \,\|\, \underbrace{g(\mathbf{A}_{\text{row}}^{2}\mathbf{X})}_{\text{2-Step Neighbors}} \,\|\, \underbrace{g(\tilde{\mathbf{A}}_{\text{sym}}^{2}\mathbf{X})}_{\text{Grand Neighbors}}} \tag{2.2}$$

where $g(\cdot)$ is the principal component analysis (PCA) for the orthogonalization of each component, followed by an L2 normalization, and structural features $\mathbf{U} \in \mathbb{R}^{n\times r}$ is derived by running the

low-rank singular value decomposition (SVD) on $\mathbf{A}$. Our 4 *design decisions* of SLIMG are:

1. **Concatenating winning normalizations** (for pain point 1 - robustness): The four components in Equation 2.2 show their strength in different scenarios: structural features $\mathbf{U}$ for graphs with noisy features, self-features $\mathbf{X}$ for a noisy structure, two-step aggregation $\mathbf{A}^2_{\text{row}}$ with no self-loops for heterophily graphs, and smoothed two-hop aggregation with self-loops $\tilde{\mathbf{A}}^2_{\text{sym}}$ of the local neighborhood for homophily graphs.
2. **Structural features** (for pain point 2 - noisy features): We resort to the structure $\mathbf{A}$ ignoring $\mathbf{X}$ when features are missing, noisy, or useless for classification. We adopt low-rank SVD on $\mathbf{A}$ with rank r to extract structural features $\mathbf{U}$.
3. **Orthogonalization and sparsification** (for pain point 3 - collinearity): To improve the efficiency and effectiveness, we run PCA on each of the four components to orthogonalize them and use group LASSO to learn sparse weights on the component level.
4. **Multi-level neighborhood aggregation** (for pain point 4 - hyperparameters): Our propagator function $\mathcal{P}$ considers multiple levels of neighborhoods through the concatenation. This allows us to remove all hyperparameters from $\mathcal{P}$ to tune for each dataset, gaining in both interpretability and efficiency. $\mathbf{X}$, $\tilde{\mathbf{A}}^2_{\text{sym}}$, and $\mathbf{A}^2_{\text{row}}$ aggregate the zero-, one-, and two-hop neighborhood of each node, respectively, considering the self-loops included in $\tilde{\mathbf{A}}^2_{\text{sym}}$.

2.3.3 Results

In Table 2.4, SLIMG outperforms all competitors in 9 out of 13 graphs, and is the only model that exhibits no failures (i.e., no red cells). In Figure 2.3, SLIMG achieves the highest accuracy while being $2.5\times$ faster than the second-best model on the largest graph with 61M edges. In Figure 2.4, SLIMG assigns large weights to the correct factors in different graph scenarios. For example, in Figure 2.4a, when there are no network effects, it assigns the largest weights to node features $g(\mathbf{X})$.

Table 2.4: **SLIMG wins** most of the times on 13 real-world datasets (7 homophily and 6 heterophily graphs) against 15 competitors. Green (■, ■, ■) marks the top three (higher is darker); red (■) marks the ones that are too low (2σ below the third place). SLIMG is the only approach that exhibits no failures (i.e., no red cells) in all datasets. Most competitors cause out-of-memory (OOM) errors on large graphs (marked by red ■).

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec	Avg. Rank
LR	51.5±1.2	52.9±4.5	79.9±0.5	73.9±1.2	79.3±1.5	48.3±1.9	56.4±0.5	24.9±1.7	26.7±1.9	27.8±0.8	63.5±0.5	53.0±0.1	61.3±0.0	11.7 (4.2)
Reg. Kernel	67.8±2.5	62.1±4.4	83.4±1.4	80.3±1.4	87.1±1.2	O.O.M.	O.O.M.	29.4±2.6	24.3±2.3	29.6±1.4	O.O.M.	O.O.M.	O.O.M.	12.2 (3.8)
Diff. Kernel	70.6±1.5	62.7±3.8	82.1±0.4	83.1±1.0	89.8±0.6	O.O.M.	O.O.M.	34.5±7.9	28.3±1.5	24.7±0.9	53.5±0.8	O.O.M.	O.O.M.	11.8 (2.5)
RW Kernel	72.7±1.7	64.1±3.9	83.1±0.7	84.2±0.7	90.6±0.7	63.2±0.2	74.2±0.0	34.9±3.5	25.0±1.6	26.4±1.1	63.1±0.7	57.6±0.1	59.5±0.0	8.3 (3.3)
SGC	76.2±1.1	65.8±3.9	84.1±0.8	83.7±1.6	90.1±0.9	65.0±3.4	74.6±5.1	38.1±4.5	33.1±1.0	24.6±0.8	64.0±1.1	56.5±0.1	69.8±0.0	6.6 (4.2)
DGC	77.8±1.4	66.1±4.2	84.3±0.6	83.9±0.7	90.4±0.2	65.2±4.0	68.7±13.	37.2±3.7	29.2±1.2	25.2±2.1	62.5±0.4	58.2±0.2	60.7±0.1	6.6 (3.2)
S^2GC	78.3±1.5	66.9±4.4	84.3±0.3	83.1±0.8	90.1±0.8	62.0±7.4	58.3±18.	34.9±4.9	27.6±1.8	26.7±1.8	63.1±0.5	58.7±0.1	61.2±0.0	6.6 (2.7)
G^2CN	76.6±1.5	64.2±3.3	81.4±0.6	82.8±1.6	88.8±0.5	O.O.M.	O.O.M.	40.7±2.9	32.1±1.5	24.3±0.5	O.O.M.	O.O.M.	O.O.M.	10.5 (4.5)
GCN	76.0±1.2	65.0±2.9	84.3±0.5	85.1±0.9	91.6±0.5	62.8±0.6	O.O.M.	38.5±3.0	31.4±1.8	26.8±0.4	62.9±0.7	57.0±0.1	63.9±0.4	6.3 (2.4)
SAGE	74.6±1.3	63.7±3.6	82.9±0.4	83.8±0.5	90.6±0.5	61.5±0.6	O.O.M.	39.8±4.3	27.0±1.3	27.8±0.9	O.O.M.	56.6±0.4	68.9±0.1	8.5 (3.5)
GCNII	77.8±1.7	63.4±3.0	84.9±0.8	82.3±1.8	90.8±0.6	45.7±0.5	O.O.M.	30.5±2.5	21.9±3.0	29.0±1.3	64.5±0.5	56.9±0.6	62.1±0.3	8.4 (4.6)
H^2GCN	77.6±0.9	64.7±3.8	85.4±0.4	49.5±16.	75.8±11.	O.O.M.	O.O.M.	31.9±2.6	25.0±0.5	28.9±0.6	63.9±0.4	58.7±0.0	O.O.M.	8.9 (4.9)
APPNP	80.0±0.6	67.1±2.8	84.6±0.5	84.2±1.7	92.5±0.3	53.4±1.3	O.O.M.	30.9±4.7	23.9±3.2	26.1±1.0	63.7±0.9	47.3±0.3	57.4±0.4	7.6 (4.8)
GPR-GNN	78.8±1.3	64.2±4.0	85.1±0.7	85.0±1.0	92.6±0.3	58.5±0.8	O.O.M.	31.7±4.7	26.2±1.6	29.5±1.1	64.5±0.4	57.6±0.2	67.6±0.1	5.4 (3.7)
GAT	78.2±1.2	65.8±4.0	83.6±0.2	85.4±1.4	91.7±0.5	58.2±1.0	O.O.M.	39.1±4.1	28.6±0.6	26.4±0.4	60.5±0.8	O.O.M.	O.O.M.	7.5 (3.7)
SLIMG	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1	1.9 (1.5)

2.4 NETINFOF: Measuring and Exploiting Network Usable Information

Section based on work that appeared at ICLR 2024 [51][PDF].

2.4.1 Goal

Given a graph with node features, how to tell if a graph neural GNN can perform well on graph tasks or not? How can we know what information, if any, is usable to the tasks, namely, link prediction and node classification? In Chapter 2.3, we study this problem in node classification and identify the useful information based on the learned parameters. We want to extend this idea by quantitatively measuring how informative the graph structure and node features are for the task at hand without training, which we call *network usable information (NUI)*. Furthermore, we intend to generalize the method to link prediction as well, which has not yet been done with linear GNNs.

Problem 3. Network Usable Information (NUI)
(1) ***Given*** *an undirected graph* $G = (\mathbf{A}, \mathbf{X})$ *and labels* $\mathbf{y}$ *of* m *nodes, where* $m \ll n$.
(2) ***Measure*** *NUI in the given graph.*
(3) ***Exploit*** *NUI to solve the graph task, i.e., link prediction and node classification.*

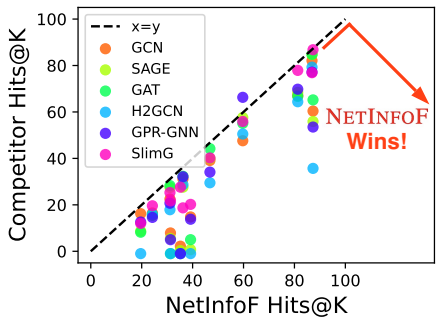
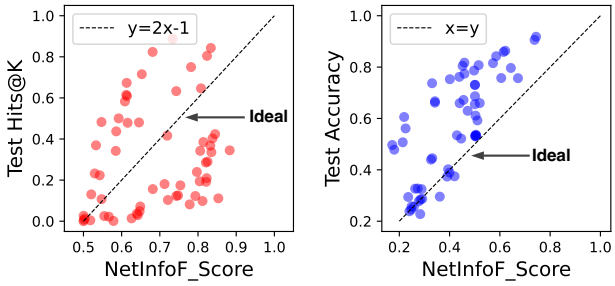


Figure 2.5: **NETINFOF wins** in real-world datasets on link prediction (most points are below or on line $x = y$).



(a) Link Prediction (b) Node Classification

Figure 2.6: **NETINFOF_SCORE highly correlates to test performance** in real-world datasets. Each point denotes the result of a component from each dataset.

2.4.2 Approach

We propose NETINFOF, a framework to measure and exploit NUI in a given graph. First, NETINFOF_PROBE measures NUI of the given graph with NETINFOF_SCORE (Equation 2.3), which

we proved is lower-bound the accuracy (Theorem 1). Next, NETINFOF_ACT solves both the link prediction and node classification by sharing the same backbone with NETINFOF_PROBE. To avoid training, we compute NETINFOF_SCORE by representing different components of the graph with derived node embeddings. For link prediction, we propose the adjustment to node similarity with a closed-form formula to address the limitations when the embeddings are static. In Figure 2.5, NETINFOF_ACT outperforms the GNN baselines most times in link prediction; in Figure 2.6, NETINFOF_SCORE measured by NETINFOF_PROBE highly correlates to the test performance.

NETINFOF_SCORE We propose to analyze the derived node embeddings in linear GNNs to avoid training. We derive 5 different components of node embeddings that can represent the information of graph structure $\mathbf{U}$, neighborhood $\mathbf{R}$, node features $\mathbf{F}$, and features of 2-step neighbors $\mathbf{P}$, and features of grand neighbors $\mathbf{S}$. Compared to SLIMG, a neighborhood node embedding $\mathbf{R}$ is added to capture the local higher-order neighborhood information of nodes with Personalized PageRank (PPR). To analyze the information in the node embedding, we propose NETINFOF_SCORE and prove that it low-bounds the accuracy:

Definition 1 (NETINFOF_SCORE of Y given X)**.** *Given two discrete random variables X and Y,* NETINFOF_SCORE *of Y given X is defined as:*

$$\text{NETINFOF\_SCORE} = 2^{-H(Y|X)} \tag{2.3}$$

where $H(\cdot|\cdot)$ denotes the conditional entropy.

Theorem 1 (NETINFOF_SCORE)**.** *Given two discrete random variables X and Y,* NETINFOF_SCORE *of Y given X low-bounds the accuracy:*

$$\boxed{\text{NETINFOF\_SCORE} = 2^{-H(Y|X)} \leq \textit{accuracy}(Y|X) = \sum_{x \in X} \max_{y \in Y} p_{x,y}} \tag{2.4}$$

where $p_{x,y}$ is the joint probability of x and y.

The proof is in [51]. The intuition behind this theorem is that the conditional entropy of Y (e.g. labels) given X (e.g. node embeddings), is a strong indicator of how good of a predictor X is, to guess the target Y. It provides an advantage to NETINFOF_SCORE by giving it an intuitive interpretation, which is the lower-bound of the accuracy. When there is little usable information for the task, the value of NETINFOF_SCORE is close to random guessing.

Adjusted Node Similarity With the derived node embeddings, how can we measure NUI as well as solve the task? In this proposal, we focus on explaining how to solve link prediction. Compared to general GNNs, the node embeddings of linear GNNs are given by a closed-form formula. Therefore, they are rarely applied on link prediction because of following two reasons:

1. **Node similarity with dissimilar neighbors**: Predicting links by GNNs relies on measuring node similarity, which is incorrect if the neighbors are expected to have dissimilar embeddings; for example, in a bipartite graph, while a source node is connected to a target node, their structural embeddings are very different, resulting in low node similarity by linear GNNs;
2. **Negative edges**: To perform well on Hits@K, it is crucial to suppress the similarity of the nodes of negative edges, i.e. the non-existent connections. Hits@K is the ratio of positive edges that are ranked at K-th place or above among both the positive and negative edges, which is preferred in link prediction. Since the embeddings of linear GNNs are static, they cannot learn to separate the embeddings of nodes on each side of the negative edges.

For these reasons, we propose an adjustment to the similarity of the nodes, which generalizes NETINFOF to link prediction. It is done by using the compatibility matrix $\mathbf{H}^* \in \mathbb{R}^{d\times d}$ to rewrite the node similarity function from a simple dot product (cosine similarity) $\mathbf{z}_i \cdot \mathbf{z}_j$ to $\mathbf{z}_i\mathbf{H}^*\mathbf{z}_j^\intercal$, where $\mathbf{z} \in \mathbb{R}^{1\times d}$ is the L2-normalized node embedding. $\mathbf{H}^*$ is computed as follows:

Lemma 2 (Compatibility Matrix with Negative Edges)**.** *The compatibility matrix with negative edges* $\mathbf{H}^*$ *has the closed-form solution and can be solved by the following optimization problem:*

$$\min_{\mathbf{H}^*} \sum_{(i,j)\in\mathcal{E}} (1 - \mathbf{z}_i\mathbf{H}^*\mathbf{z}_j^\intercal) - \sum_{(i,j)\in\mathcal{E}_{neg}} (\mathbf{z}_i\mathbf{H}^*\mathbf{z}_j^\intercal), \tag{2.5}$$

where $\mathcal{E}_{neg}$ *denotes the set of negative edges.*

The proof is in [51]. We further provide several techniques to ensure the fast computation of $\mathbf{H}^*$.

NETINFOF_PROBE Based on Theorem 1, we propose NETINFOF_PROBE to compute NETINFOF_SCORE without exactly computing the conditional entropy of the high-dimensional variables. By sampling negative edges, we turn the link prediction into a binary classification task. For each component of node embeddings, it estimates its corresponding $\mathbf{H}^*$ and discretizes the adjusted node similarity of edges (e.g. $\hat{\mathbf{U}}_i\mathbf{H}^*_{\hat{\mathbf{U}}} \odot \hat{\mathbf{U}}_j$) into k bins. NETINFOF_SCORE can then be easily computed between two categorical variables using a prediction table $k \times 2$. In Figure 2.6, we find that NETINFOF_SCORE measured by NETINFOF_PROBE highly correlates to the test performance.

NETINFOF_ACT To solve link prediction, NETINFOF_ACT shares the same derived node embeddings and uses a link predictor following by the Hadamard product of the embeddings. We transform the embeddings on one side of the edge with $\mathbf{H}^*$ and concatenate all components:

$$\underbrace{\hat{\mathbf{U}}_i\mathbf{H}^*_{\hat{\mathbf{U}}} \odot \hat{\mathbf{U}}_j}_{\text{Structure}} \| \underbrace{\hat{\mathbf{R}}_i\mathbf{H}^*_{\hat{\mathbf{R}}} \odot \hat{\mathbf{R}}_j}_{\text{PPR}} \| \underbrace{\hat{\mathbf{F}}_i\mathbf{H}^*_{\hat{\mathbf{F}}} \odot \hat{\mathbf{F}}_j}_{\text{Features}} \| \underbrace{\hat{\mathbf{P}}_i\mathbf{H}^*_{\hat{\mathbf{P}}} \odot \hat{\mathbf{P}}_j}_{\substack{\text{Features of}\\ \text{2-Step Neighbors}}} \| \underbrace{\hat{\mathbf{S}}_i\mathbf{H}^*_{\hat{\mathbf{S}}} \odot \hat{\mathbf{S}}_j}_{\substack{\text{Features of}\\ \text{Grand Neighbors}}} \tag{2.6}$$

where $(i, j) \in \mathcal{E} \cup \mathcal{E}_{\text{neg}}$. We use LogitReg as the predictor for its scalability and interpretability.

2.4.3 Results

In Table 2.5, NETINFOF outperforms GNN baselines in 11 out of 12 datasets on link prediction, and has the highest average rank. Compared to non-linear GNNs, SLIMG performs worse in most heterophily graphs, showing that it cannot properly measure the node similarity of heterophily embeddings. By addressing the limitations of linear GNNs, NETINFOF consistently outperforms both SLIMG and non-linear GNNs in both homophily and heterophily graphs.

Table 2.5: **NETINFOF wins** on link prediction in most real-world datasets. Hits@100 is reported for most datasets, and Hits@1000 for the large datasets (Products, Twitch, and Pokec).

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Twitch	Pokec	Avg. Rank
GCN	67.1±1.8	60.4±10.	47.6±13.	22.5±3.1	39.1±1.6	14.8±0.6	02.2±0.1	82.1±4.5	16.5±1.0	31.1±1.7	16.2±0.3	07.9±1.7	4.1 (1.3)
SAGE	68.4±2.8	55.9±2.5	57.6±1.1	27.5±2.1	40.0±1.9	00.7±0.1	00.3±0.2	84.7±3.6	15.5±1.5	27.6±1.4	08.7±0.6	05.5±0.5	4.5 (1.2)
GAT	66.7±3.6	65.2±2.6	55.1±2.4	28.3±1.6	44.2±3.5	05.0±0.8	O.O.M.	84.8±4.5	15.6±0.8	32.3±2.4	08.2±0.3	O.O.M.	4.0 (1.7)
H^2GCN	64.4±3.4	35.7±5.4	50.5±0.9	17.9±0.7	29.5±2.4	O.O.M.	O.O.M.	79.3±4.5	16.0±2.6	28.7±2.1	O.O.M.	O.O.M.	6.2 (1.0)
GPR-GNN	69.8±1.9	53.5±8.1	66.3±3.3	20.7±1.8	34.1±1.1	13.8±0.8	O.O.M.	77.2±5.6	14.6±2.7	32.1±1.3	12.6±0.2	05.0±0.2	4.6 (1.8)
SLIMG	77.9±1.3	86.8±1.0	55.9±2.8	25.3±0.9	40.2±2.5	20.2±1.0	27.6±0.6	76.9±2.8	19.6±1.5	18.7±1.0	12.0±0.3	21.7±0.2	3.5 (1.8)
NETINFOF	81.3±0.6	87.3±1.3	59.7±1.1	31.1±1.9	46.8±2.2	39.2±1.8	35.2±1.1	86.9±2.3	24.2±2.0	36.2±1.2	19.6±0.7	31.3±0.5	1.1 (1.3)

Chapter 3

Graph Mining: Graph-Level Tasks

3.1 Preliminaries

Graph Database A graph database consisting of I graphs $\mathcal{G} = \{G_1, \ldots, G_I\}$, where each graph $G_i(\mathcal{V}_i, \mathcal{E}_i)$ has a set of nodes $\mathcal{V}_i$ and a set of edges $\mathcal{E}_i$. If the database is node-labeled and weighted, each node $v \in \mathcal{V}_i$ has a label $l(v) \in \mathcal{T}$, where $\mathcal{T}$ is the set of unique node labels, and each edge $(u, v) \in E_i$ is associated with a weight $w(u, v)$. If the database is node-attributed, each graph G_i is associated with a node feature matrix $\mathbf{X}_{G_i} \in \mathbb{R}^{|\mathcal{V}_i| \times f}$, where f is the feature dimension.

Graph Anomaly Detection A detailed survey can be found in [5]. On the one hand, some studies aim to identify the graph anomalies by mining the structural features of the graphs. OddBall [4] detects anomalous nodes in a single weighted graph. ANOMRANK [112] detects anomalies in the dynamic graphs. To improve interpretability, features are analyzed in pairs in [38], allowing easy visualization of outliers. LookOut [31] further turns anomaly detection into a two-dimensional plot selection problem, selecting the most explainable plots to highlight the anomalies. SpotLight [24] aims to detect the anomalies in the streaming graphs .

On the other hand, some researchers seek to explain the graph anomalousness by frequent substructures among graphs in a database. Cook *et al.* [16] proposed a graph substructure discovery framework, which Noble and Cook [69] leverage in anomaly detection by using compression rates in each iteration. For numerical edge weights, Yagada [21] uses discretization to assign discrete labels to edges. In summary, graph anomaly detection aims to give anomaly scores a_i for graphs $G_i \in \mathcal{G}$ such that a higher score indicates a higher abnormality.

Kernel Convolution Networks (KCNs) Graph kernels are designed to measure similarity on a pair of graphs. [53] derived the first neural network that outputs the random walk kernel similarity scores between input graph and hidden, learnable path-like graphs. Random walk neural network (RWNN) [68] generalized the work of [53] such that the hidden graphs can have any structure without the path constraints. The designed model is claimed to be interpretable as the learned hidden graphs "summarize" the input graphs. Later, [17] and [26] extended RWNN [68] to a multi-layer architecture, in which each layer compares subgraphs around each node of the input with learnable hidden graphs. We refer to these models as Kernel Convolution Networks (KCNs).

3.2 GAWD: Graph Anomaly Detection in Weighted Graph Databases

Section based on work that appeared at ASONAM 2021 [47][PDF].

3.2.1 Goal

Given a large graph database containing directed weighted node-labeled graphs, how can we detect the anomalous graphs? How can we spot anomalies and summarize the normal behavior without simultaneously losing information? Unlike embedding-based anomaly detector, we aim to propose an anomaly detection that is explainable based on Minimum Description Length (MDL). Moreover, we want it to be lossless during compression, and be able to handle weighted graphs.

Problem 4. Graph Anomaly Detection in Weighted Graph Databases
(1) ***Given*** *a node-labeled weighted graph database* $\mathcal{G} = \{G_1(\mathcal{V}_1, \mathcal{E}_1), \ldots, G_I(\mathcal{V}_I, \mathcal{E}_I)\}$.
(2) ***Output*** *anomaly score* a_i *for each graph* $G_i \in \mathcal{G}$.

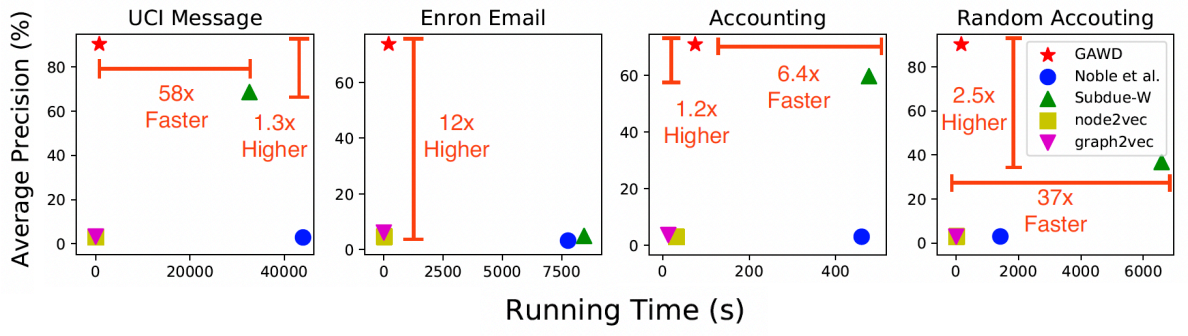


Figure 3.1: **GAWD wins on both effectiveness and scalability**: We evaluate GAWD on 4 graph databases and show the big gap between it and competitors w.r.t. average precision and run time.

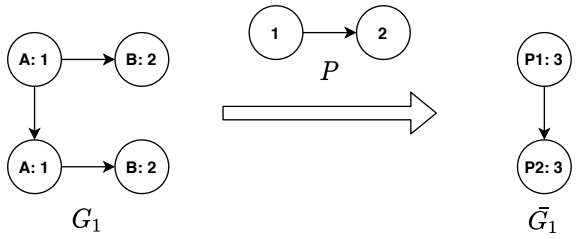
3.2.2 Approach

We propose GAWD, which follows a general information-theoretic framework depicted in Algorithm 1. This framework generalizes previous graph anomaly detection methods [16, 69]. Given a graph database, the idea is to iteratively identify the "best" substructure that yields the largest compression, replacing each of its occurrences with a super node. Each graph in the database is then scored by how much it compresses over iterations — the more the compression, the lower the anomaly score. The heart of this algorithm is the encoding scheme of graphs by Minimum Description Length (MDL), which includes encoding the structure of graphs, i.e., nodes and edges.

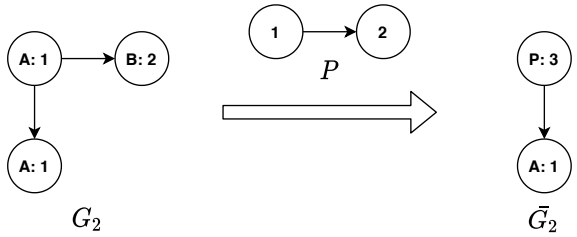
Built upon this encoding scheme, GAWD is improved to (i) handle edge weights, and (ii) enable lossless reconstruction, with weight encoding and rewiring encoding, respectively.

Data: A graph database.
Result: Anomaly scores for graphs in database.
while *True* **do**
  Detect frequent patterns in graph database;
  if *No pattern is found* **then**
    Break;
  end
  Identify pattern which compresses graphs in database the most;
  Compress graphs by this pattern;
end
Compute anomaly scores by compression rate;

Algorithm 1: General framework for graph anomaly detection followed by [16, 69], blue highlights the differences with GAWD.



(a) Case 1: between two super nodes.



(b) Case 2: between a super node and a regular node.

Figure 3.2: Rewiring encoding: Two possible cases that the edge (re)connectivity information are different.

Weight Encoding. Given a substructure $P_j = (\mathcal{V}_j, \mathcal{E}_j)$ at iteration j, $\mathcal{E}_{j,(u,v)}$ denotes all the edges in the instances of substructure P_j in the database corresponding to $(u, v) \in \mathcal{E}_j$. Weight encoding is composed of two steps:

1. **Representative Weight Discovery**: We identify the representative edge weight $w^*_{P_j}(u, v)$ among $\mathcal{E}_{j,(u,v)}$ by turning it into an optimization problem based on the MDL encoding. This optimization problem aims to find out the edge weight that minimizes the bits needed for the correction. Although not convex, the optimization is only 1-dimensional and hence relatively easy to solve. We employ Dichotomous Search [14], which efficiently returns the optimal solution in most cases.
2. **Weight Corrections**: After discovering $w^*_{P_j}$, we now encode the weights in each instance. 1 bit is used to identify whether the weight correction is needed. If so, an extra 1 bit is used to record the sign of the error. The numeric value is encoded by the universal code. We remark that instead of discretizing edge weights into labels, our encoding scheme handles the numeric value and is lossless.

Rewiring Encoding. After replacing P_j with a super node, all the edges connected to P_j merge into super edges. For lossless reconstruction, the edge (re)connectivity information needs to be encoded. When a super edge $e(x, y)$ is created between x and y, there are two possible cases (in Figure 3.2):

1. Both x and y are super nodes after compression.
2. $x = (\mathcal{V}_x, \mathcal{E}_x)$ is a super node and y remains a regular node after compression.

For the former case, we first encode the cardinality of e, denoted $c_e = |\mathcal{V}_j|^2$, depicting how many edges it represents. For each edge in e, we encode substructure node IDs of its source and destination, and then encode its weight. For the latter case, we encode how many edges e branch to, denoted $b_y = |\mathcal{V}_j|$. In other words, b_y denotes how many distinct nodes within x that y connects to. For each edge, we encode the substructure node ID of y's neighbor $n \in \mathcal{V}_x$. We then encode the weight of each edge the same as in the former case.

Anomaly Scoring Function. Once we identify the best substructure P_j at iteration j, we compress the graphs in the database by P_j and save the compression rate c_i^j, for each graph i, defined as:

$$c_i^j = \big(DL^*_{j-1}(G_i) - DL^*_j(G_i)\big)/DL^*_0(G), \tag{3.1}$$

where $DL_j(G_i)$ is the description length of G_i after j iterations. Finally, we compute the anomaly scores as follows:

$$a_i = 1 - \frac{1}{j}\sum_{k=1}^{j}\big[(j-k+1) * c_i^k\big] \tag{3.2}$$

The anomaly score ranges from 0 to 1, where 1 denotes the most anomalous. The compression rates are linearly weighted by the term $j-k+1$, meaning that the earlier we identify the substructure as the best one, the less anomalous that the graphs containing it are.

3.2.3 Results

In Table 3.1, GAWD outperforms most baselines significantly in all datasets. Noble *et al.* and graph2vec fail because it cannot handle edge weights. GAWD shows 31.6%, 1365%, 18.7% and 145% improvement over Subdue-W in average precision on four datasets, respectively, highlighting the insufficiency of discretization to handle edge weights. Even if node2vec accepts edge weights, it is not sufficiently sensitive to detect anomalies in the edge weights.

Method	Precision		Recall		AUC	AP	Time
	@45	@90	@45	@90			
node2vec	6.7	5.6	3	5.1	47.6	3.1	58.6s
graph2vec	2.2	1.1	1.0	1.0	48.7	3.1	2.9s
Noble *et al.*	0.0	0.0	0.0	0.0	47.6	3.1	43988s
Subdue-W	100.0	60.0	45.5	54.5	94.8	68.6	32621s
GAWD	100.0	92.2	45.5	83.8	93.8	90.3	760s

(a) UCI Message Dataset

Method	Precision		Recall		AUC	AP	Time
	@10	@20	@10	@20			
node2vec	10.0	5.0	4.0	4.0	58.7	4.6	24.6s
graph2vec	10.0	5.0	4.0	4.0	59.0	6.1	1.1s
Noble *et al.*	0.0	0.0	0.0	0.0	51.7	3.2	7768s
Subdue-W	10.0	5.0	4.0	4.0	48.2	4.9	8443s
GAWD	90.0	5.0	36.0	68.0	82.1	73.8	207s

(b) Enron Email Dataset

Method	Precision		Recall		AUC	AP	Time
	@200	@400	@200	@400			
node2vec	5.0	3.0	2.1	2.5	47.0	30.	31.3s
graph2vec	3.5	4.5	1.5	3.8	54.0	3.7	12.7s
Noble *et al.*	3.0	3.1	1.2	2.6	50.6	3.1	460s
Subdue-W	82.0	74.0	34.2	61.7	76.0	59.8	477s
GAWD	100.0	81.5	41.7	67.9	88.0	71.0	75s

(c) Accounting Dataset

Method	Precision		Recall		AUC	AP	Time
	@200	@400	@200	@400			
node2vec	2.5	3.3	1.0	0.8	48.3	2.8	25.9s
graph2vec	0.0	2.0	1.7	3.1	49.6	3	14s
Noble *et al.*	3.0	3.0	1.3	2.5	50.0	3	1426s
Subdue-W	63.5	35	26.6	29.3	71.5	36.8	6584s
GAWD	100.0	89.0	41.8	74.5	95.3	90.2	176s

(d) Random Accounting Dataset

Table 3.1: **GAWD significantly outperforms all the baselines**: We show that the performance of GAWD and structure-based and embedding-based baselines on 4 real-world and random graph datasets.

3.3 RWK$^+$: Descriptive Kernel Convolution Network with Improved Random Walk Kernel

Section based on work that appeared at WWW 2024 [52][PDF].

3.3.1 Goal

Given a graph database, how can we spot anomalous graphs by learning the frequent graph patterns? In Chapter 3.2, we introduce how graph anomaly detection can be done by frequent substructure mining and MDL compression. We now turn to explore how it can be done by end-to-end learning in a node-attributed graph database. More specifically, we want to solve this problem by adapting the learnable random walk graph kernel (RWK) in GNNs. We name these "*GNNs meet graph kernels*" style models Kernel Convolutional Networks (KCNs).

Problem 5. Improved Random Walk Kernel
(1) ***Given*** *a graph database with node-attributed graphs.*
(2) ***Find****:*
- *(a) An improved random walk kernel (RWK) that captures better substructural embeddings for graph anomaly detection.*
- *(b) An algorithm that learns the frequent substructures with the improved RWK.*
- *(c) An algorithm that performs well in graph classification and regression.*

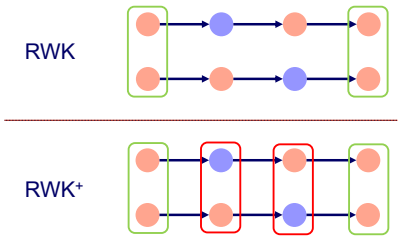


Figure 3.3: **RWK$^+$ is better.** In RWK, these two walks are identical; in RWK$^+$, these two walks are different because of considering the intermediate nodes.

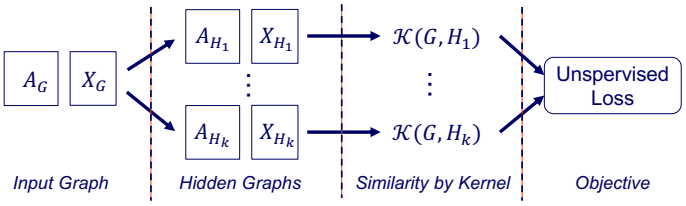


Figure 3.4: RWK$^+$CN: Each input graph is represented by its RWK similarity to a set of small hidden graphs that are learned end-to-end with an unsupervised loss.

3.3.2 Approach

We propose RWK$^+$, RWK$^+$CN, and RWK$^+$Conv to solve three different problems. First, to capture more representative patterns, we introduce several improvements to the RWK in both effectiveness and efficiency and propose an improved graph kernel RWK$^+$. Second, we propose a *descriptive*

KCN RWK$^+$CN by flipping the objective from a discriminative one to a descriptive one that helps us capture relational patterns in the graph database. In addition, for the first time we derive the mathematical connection of RWK$^+$ to layer-wise neural network operators, which inspires us to propose a novel GNN layer RWK$^+$Conv. A summary of our contributions is as follows:

1. **RWK$^+$**, an improved RWK with efficient color-matching.
2. **RWK$^+$CN**, a descriptive KCN that learns descriptive hidden graphs.
3. **RWK$^+$Conv**, a novel GNN layer sharing connection with message-passing.

Data: $G_i = (\mathbf{A}_{G_i}, \mathbf{X}_{G_i})$; $H_j = (\mathbf{A}_{H_j}, \mathbf{X}_{H_j})$; max step t; {H_j are parameters in RWK$^+$Conv }.

Init: $\mathbf{Y}_0 \leftarrow \mathbf{X}_{G_i}\mathbf{X}_{H_j}^\intercal$, $\mathbf{Y} \leftarrow \mathbf{Y}_0$ {$\mathbf{Y}_0 \leftarrow \sigma(\mathbf{X}_{G_i}\mathbf{X}_{H_j}^\intercal)$};

for $i = 1$ ***to*** t **do**

  $\mathbf{Y} \leftarrow \mathbf{A}_{G_i}\mathbf{Y}\mathbf{A}_{H_j}^\intercal$;

  $\mathbf{Y}^{(i)} \leftarrow \mathbf{Y}_0 \odot \mathbf{Y}$;

  $\mathbf{Y} \leftarrow \mathbf{Y}_0 \odot \mathbf{Y}^{(i)}$;

Return $\sum_{i,j} \mathbf{Y}^{(t)}_{i,j}$ or $\sum_{i,j}(\sum_l \lambda_l \cdot \mathbf{Y}^{(l)})_{i,j}$;

Algorithm 2: Fast Color-Matching RWK and RWK$^+$Conv.

Table 3.2: **RWK$^+$CN works** on a GED-based evaluation with 2-regular labeled graph. color-matching and structural colors (SC) improve the quality of structure and label learned by hidden graph.

Method	Additional Features	GED w/o Labels	p-value w/ Row 1	p-value w/ Row 2
RWNN	None	5.25 ± 0.64	-	-
RWK$^+$CN	None	5.02 ± 0.63	0.049^*	-
RWK$^+$CN	Identity	4.81 ± 0.70	$1.0e\text{-}03^{**}$	0.043^*
RWK$^+$CN	SC	4.82 ± 0.70	$1.1e\text{-}03^{**}$	0.043^*
Method	**Additional Features**	**GED w/ Labels**	**p-value w/ Row 1**	**p-value w/ Row 2**
RWNN	None	7.25 ± 0.64	-	-
RWK$^+$CN	None	6.60 ± 0.93	$1.8e\text{-}04^{***}$	-
RWK$^+$CN	Identity	6.12 ± 1.01	$2.9e\text{-}08^{***}$	$1.8e\text{-}03^{**}$
RWK$^+$CN	SC	6.25 ± 1.01	$7.2e\text{-}07^{***}$	0.015^*

RWK$^+$ with efficient color-matching. Let $\mathbf{X}_{G_i} \in \mathbb{R}^{|\mathcal{V}_{G_i}| \times f}$ depict the f-dimensional continuous attributes for all nodes, and $\mathbf{X}_{H_j} \in \mathbb{R}^{|\mathcal{V}_{H_j}| \times f}$ and $\mathbf{A}_{H_j} \in \mathbb{R}^{|\mathcal{V}_{H_j}| \times |\mathcal{V}_{H_j}|}$ depict the learnable node features and the learnable adjacency matrix of the hidden graph H, respectively. For two graphs G and H, let $\mathbf{S} = \mathbf{X}_{H_j}\mathbf{X}_{G_i}^\intercal \in \mathbb{R}^{|\mathcal{V}_{H_j}| \times |\mathcal{V}_{G_i}|}$ encode the dot product similarity between the attributes of the vertices from two graphs, where $\mathbf{s} = \mathrm{vec}(\mathbf{S})$ is the 1-d vectorized representation of $\mathbf{S}$. Random walk neural network (RWNN) [68] proposes to compute the revised RWK with t steps as:

$$\mathcal{K}^t_{rw-}(G, H) = \mathbf{1}^\intercal(\mathbf{s}\mathbf{s}^\intercal \odot A^t_{G\otimes H})\mathbf{1} \ , \tag{3.3}$$

where $\odot$ denotes the element-wise product. We identify that the RWK originally developed in RWNN only enforces the same label at the beginning and end of two walks, while ignoring the intermediates (Figure 3.3). We reformulate it to count a walk as shared only if all corresponding node pairs exhibit the same node label (i.e. color) at all steps along the walk:

$$\mathcal{K}^t_{rw+}(G, H) = \mathbf{1}^\intercal(\mathbf{s}\mathbf{s}^\intercal \odot A_{G\otimes H})^t\mathbf{1} \ , \tag{3.4}$$

We further propose the improved graph kernel RWK$^+$ through transforming its formulation for efficient iterative computation in Algorithm 2.

RWK$^+$CN learning descriptive hidden graphs. We propose RWK$^+$CN (in Figure 3.4), with an unsupervised objective that uses RWK$^+$ as the core kernel and maximizes the total RWK similarity between the graphs in the database and hidden graphs, which are reflective of the frequent walks (i.e. patterns). Moreover, we use additional "structural colors" to help better capture structural similarity between graphs, generated by a fixed randomly initialized GNN. We demonstrate the descriptive learning ability of RWK$^+$CN with our carefully designed testbeds in Table 3.2.

RWK$^+$Conv, a novel GNN layer. By unrolling RWK$^+$, we discover that the derivation can be re-written as a sequence (i.e. multiple layers) of graph convolutional operations, connecting with regular GCN layers [40]. In Algorithm 2 (gray part), by viewing hidden graphs as learnable parameters, we transform the RWK$^+$ algorithm into a novel GNN layer called RWK$^+$Conv. The RWK$^+$Conv layer brings better expressiveness than the GCN layer thanks to two major improvements:

1. Element-wise product operation with $\mathbf{Y}_0$ motivated from node color matching; and
2. Multi-step within a single convolution layer that shares the same parameter $\mathbf{A}_H$ and $\mathbf{X}_H$.

3.3.3 Results

We evaluate RWK$^+$ on supervised graph anomaly detection with 10 real-world chemical compound graph databases. In Table 3.3, iGAD with our proposed RWK$^+$ outperforms the original model on *all* datasets (p-val <0.001). This suggests that the hidden graphs learned through RWK$^+$ are consistently better than the ones extracted by RWK, assisting iGAD in better detecting anomalous graphs. In Figure 3.5, a graph classification base model with RWK$^+$ is only slightly slower than with RWK, although the overhead is negligible (<1 second), and scales linearly.

We evaluate RWK$^+$Conv on graph regression and classification with three real-world datasets, ZINC, ogbg-molhiv and ogbg-molpcba. In Table 3.4, we find that RWK$^+$Conv outperforms both baselines significantly across all datasets and tasks. This empirically demonstrates the better expressiveness of RWK$^+$Conv than of GCNConv. We also report the run time per epoch on the largest node classification dataset PubMed in Table 3.5, where RWK$^+$Conv only creates negligible computational overhead (less than 0.05 seconds).

Table 3.3: **RWK$^+$ works** in graph anomaly detection on 10 real-world datasets. Recall is reported. iGAD using our proposed RWK$^+$ as structural feature extractor outperforms original iGAD on all datasets.

Dataset	MCF-7	MOLT-4	PC-3	SW-620	NCI-H23	OVCAR-8	P388	SF-295	SN12C	UACC-257
iGAD + RWK	75.1±1.1	74.1±0.8	77.9±1.2	78.6±0.9	78.7±1.3	78.8±0.3	83.1±1.7	78.3±1.1	79.4±0.6	78.0±1.0
iGAD + RWK$^+$	76.4±0.6	74.3±1.0	78.8±1.1	79.2±0.5	79.5±2.2	79.2±0.9	84.0±1.4	78.5±0.9	79.5±1.6	79.5±0.7

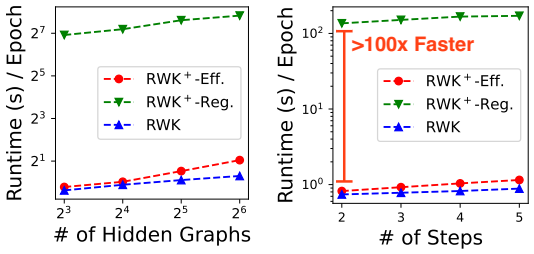


Figure 3.5: **Our efficient computation of RWK$^+$ is fast**: Runtime of a base model computed by Algorithm 2, regular Equation 3.4, and vanilla RWK.

Table 3.4: **RWK$^+$Conv wins** in all 3 real-world graph regression and classification datasets.

Dataset	ZINC	ogbg-molhiv	ogbg-molpcba
Metric	MAE ↓	ROC-AUC ↑	AP ↑
GCNConv	0.3258±0.0067	76.06±0.97	20.20±0.24
GINConv	0.2429±0.0033	77.78±1.30	22.66±0.28
RWK$^+$Conv	0.2082±0.0025	78.61±0.61	24.90±0.12

Table 3.5: **RWK$^+$Conv is fast**, with only negligible runtime overhead in seconds.

Step Length	2	3	4	5
GCNConv	0.0269	-	-	-
RWK$^+$Conv	0.0371	0.0464	0.0522	0.0619

Chapter 4

Time Series Mining

4.1 Preliminaries

Group Anomaly Detection. Earlier group anomaly detection approaches [11, 65, 101] require prior knowledge of group memberships, while the solution proposed by [114] requires information on pairwise relations among data points. We assume that the observations $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ are given, where $\mathbf{x}_i \in \mathbb{R}^m$. Since time series can be transformed into some form of $m-$dimensional cloud, group anomaly detection can be easily implemented in time series. This has significant potential because anomalies in time series data usually consist of a series of abnormal patterns, rather than a single abnormal point, which is usually recognized as noise.

Time Series Anomaly Detection Consider a univariate time series $\mathbf{x} = \{x_1, x_2, \ldots, x_K\}$, where $\mathbf{x}$ is a sequentially ordered collection of K data points. Each $x_i \in \mathbb{R}$ corresponds to a scalar observation at each time step. Then, let $\mathcal{D}_{trn}$ be a set of training data consisting of only normal time series, and $\mathcal{D}_{test}$ be a set of unlabeled test data containing both normal and anomalous time series. For sequence-level anomaly detection, the problem is to detect all anomalous time series in $\mathcal{D}_{test}$, i.e., to assign the accurate label $y_i \in \{-1, +1\}$ for each $\mathbf{x}_i \in \mathcal{D}_{test}$. For point-level anomaly detection, the problem is to detect the anomalous time points, i.e., to assign the accurate label $\mathbf{y}_i = \{y_1, \ldots, y_K\}$ for each $\mathbf{x}_i \in \mathcal{D}_{test}$, where $y_i \in \{-1, +1\}$.

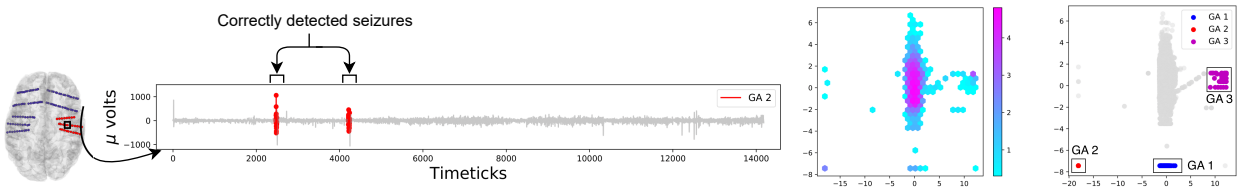
4.2 GEN^{2}OUT: Detecting and Ranking Generalized Anomalies - Seizure Detection in EEG

Section based on work that appeared at Big Data 2021 [48][PDF].

4.2.1 Goal

How can we spot and rank point- as well as group-anomalies? How can we design an anomaly score function, so that it assigns intuitive scores to both point-anomalies, as well as group-anomalies? We refer to them as generalized anomalies. Our motivating application is seizure detection in EEG, where the seizure is composed of a series of abnormal patterns in EEG signal. Our goal is to design a principled and fast anomaly detection algorithm for a given cloud of m-dimensional point-cloud data that provides a unified view as well as a scoring function for each generalized anomaly.

Problem 6. Detecting and Ranking Generalized Anomalies
(1) ***Given*** *a point-cloud dataset from an application setting.*
(2) ***Detect and rank*** *point-anomalies and group-anomalies, i.e., generalized anomaly.*

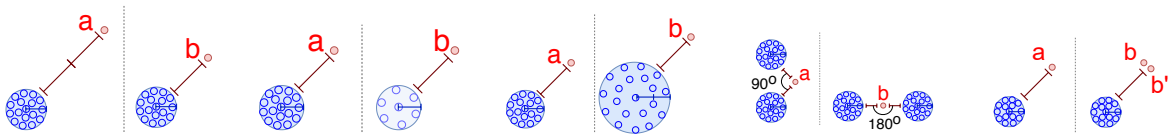


(a) EEG data (b) Heatmap of http data (c) Detected group anomalies

Figure 4.1: (a) **GEN^{2}OUT matches ground truth.** Brain scan of the patient with electrode positions (left), and detected groups in red (right), matching the ground truth seizure locations. (b) Heatmap of *http* intrusion detection dataset, and (c) **GEN^{2}OUT correctly spots group (DDoS) attacks**, marked GA1, GA2 and GA3.

4.2.2 Approach

We propose GEN^{2}OUT, an anomaly detector that detects generalized anomalies, i.e., point-anomalies and group-anomalies. The main insight is to exploit sampling, which drops the point-anomalies and make the group-anomalies become point-anomalies. To ensure that the anomaly scores are comparable across (sampled) datasets, we propose GEN^{2}OUT$_0$ as the based detector of GEN^{2}OUT, which obeys carefully designed 5 axioms. In Figure 4.1, GEN^{2}OUT detects group anomalies that correspond to seizure period in the patient; and, detects DoS/DDoS attack as group anomalies.



(a) A1: Distance Axiom (b) A2: Density Axiom (c) A3: Radius Axiom (d) A4: Angle Axiom (e) A5: Group Axiom

Figure 4.2: Illustration of Axioms.

Table 4.1: **GEN^{2}OUT wins** as it obeys all the axioms a generalized anomaly detector should follow. We compare the methods statistically, by conducting two-sample t-test based on scores obtained for points a, b. A positive difference in score indicates that the detector follows that axiom (see Figure 4.2). ■ indicates that the detector follows the axiom, ■ indicates that the detector does not obey the axiom.

	LODA [72]		RRCF [30]		IForest [61]		GEN^{2}OUT$_0$	
	Statistic	p-value	Statistic	p-value	Statistic	p-value	Statistic	p-value
A1: Distance Axiom	0	1	3.6	0.002**	2.1	0.054	11.4	1.2e-9***
A2: Density Axiom	7e15	2e-275***	-0.14	0.89	-10	8.6e-9***	25.2	1.7e-15***
A3: Radius Axiom	0	1	6.4	4.8e-6***	11.9	5.9e-10***	21.3	3.4e-14***
A4: Angle Axiom	6.6	3.2e-6***	17.5	9.6e-13***	-0.2	0.83	53.7	2.5e-21***
A5: Group Axiom	-14.7	1.8e-11***	1.1	0.27	0.95	0.35	28.2	2.6e-16***

GEN^{2}OUT$_0$. We propose 5 axioms (see Figure 4.2) to examine whether an anomaly detector is provided with the ability to compare the scores across datasets: producing higher anomaly scores when an instance is farther away from data kernel (*distance* aware), or lies in low density locality (*density, radius* and *group* aware), and not aligned with majority of data (*angle* aware [45]).

Moreover, we discover an insight of ATOMICTREE. Given $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, ATOMICTREE is grown through recursive division of $\mathcal{X}$ by randomly selecting an attribute and a split value until all the leaf nodes contain exactly one instance of observations. We observe the following property:

Insight 1 (Power Depth Property)**.** *The growth of the tree depth with the logarithm counts of observations is linear irrespective of the data distribution.*

Finally, we propose GEN^{2}OUT$_0$, of which Isolation Forest (IForest) [61] is a special case, and it fulfils all the axioms. The major differences between GEN^{2}OUT$_0$ and IForest are:

1. **Random ATOMICTREE construction with all data points**: allows GEN^{2}OUT$_0$ to be aware of the distance between normal and anomalous points; while IForest samples the data points and excludes the anomalous points during construction in most cases.
2. **Data-dependent ATOMICTREE depth estimation**: allows GEN^{2}OUT$_0$ to give more accurate anomaly scores by fitting a linear regression model based on Insight 1; while IForest treats all datasets the same with a fixed average path length function.

The scoring function of GEN^{2}OUT$_0$ is based on depth, where the shallower a given sample point is in ATOMICTREE (easier to separate it from the majority), the more anomalous. In Table 4.1, we demonstrate that GEN^{2}OUT$_0$ is the only anomaly detector fulfilling all axioms, and thus can be used in GEN^{2}OUT to detect group-anomalies.

GEN^{2}OUT. We propose GEN^{2}OUT, which spots and scores both point- as well as group-anomalies. The idea is that, with a sampling rate of a point of the group anomalies will be stripped of its

cohorts, and thus behave like a point-anomaly, exhibiting a high anomaly score. We refer to this sampling process as '*qualification*', and to the sampling rate as *'qualification rate'* q. $\mathcal{Q} = \{1, 1/2^1, 1/2^2, \ldots, 1/2^{10}\}$ denotes the set of qualification rates. We firstly need some definitions:

Definition 2 (X-ray line)**.** *For a given data point* $\mathbf{x}_i$*, the X-ray line is* $\{\text{GEN}^2\text{OUT}_0(\mathbf{x_i}, q), q\}$.

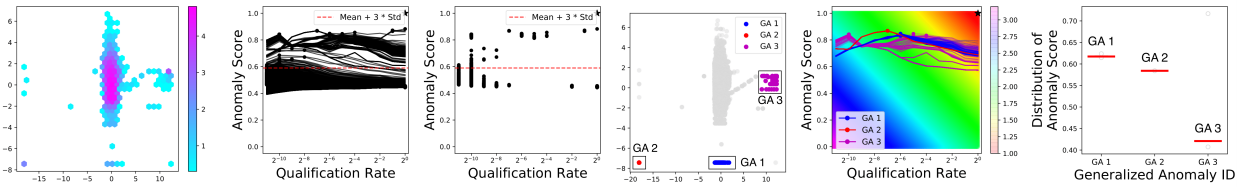
Definition 3 (X-ray plot)**.** *For a cloud of* n *points, the X-ray plot is the* 2*-d plot of all* n *X-ray lines.*

Definition 4 (Apex)**.** *Apex of point* $\mathbf{x_i}$ *is the point with the highest anomaly score among* $\mathcal{Q}$*, i.e.,* $\max_{q \in \mathcal{Q}} \text{GEN}^2\text{OUT}_0(\mathbf{x_i}, q)$.

The procedure of GEN^{2}OUT is as follows: (1) In Figure 4.3b, it plots the X-ray plot for each data point. (2) In Figure 4.3c, it extracts the apex for each data point and keep those with scores that are three standard deviation higher than the mean score. (3) In Figure 4.3d, it groups the anomalies with a clustering algorithm. (4) In Figure 4.3e, it scores each anomaly with the distance between its apex and the most anomalous point $\{\text{GEN}^2\text{OUT}_0 \text{ score} = 1, q = 1\}$ (top right) on the X-ray plot. (5) In Figure 4.3f, it outputs the score of each group-anomaly by averaging the scores in the group.

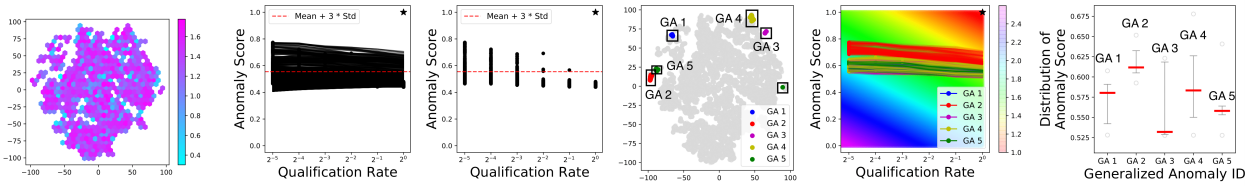
4.2.3 Results

In Figure 4.3, GEN^{2}OUT matches the ground truth as it detects three group-anomalies, which correspond to DDoS attacks. In Figure 4.4, GEN^{2}OUT detects the ground truth seizure by giving the corresponding group-anomaly the highest score. It distinguishes seizure as a series of anomalies (group-anomaly) on time series, and filters out unintentional noises (point-anomaly).



(a) Data heatmap (b) X-ray plot (c) Apex extraction (d) Outlier grouping (e) Anomaly isocurves (f) Scoring

Figure 4.3: **GEN^{2}OUT detects DDoS attacks** on intrusion detection http dataset.



(a) Heatmap of tSNE representation of data (b) X-ray plot (c) Apex extraction (d) Outlier grouping (e) Anomaly isocurves (f) Scoring

Figure 4.4: **GEN^{2}OUT works on real-world EEG data**. Assigns highest anomaly score to group anomaly GA2 that corresponds to seizures as shown in Figure 4.1a.

4.3 TSAP: Self-tuning Self-supervised Time Series Anomaly Detection

Section based on work that is under review.

4.3.1 Goal

Given a time series dataset, how can we determine the characteristics of the anomalies? The anomalies in the dataset may have the same characteristics (or hyperparameters), such as starting location, duration, or severity. How can we use this information to better detect anomalies? Our goal is to propose a framework that automatically detects the hyperparameters of the anomalies by fine-tuning, and leverages this for better sequence-level time series anomaly detection. To enable a gradient-based optimization of hyperparameters, a differential augmentation model pre-trained with pseudo anomalies is needed.

Problem 7. Sequence-Level Time Series Anomaly Detection (TSAD)
(1) ***Given*** $\mathcal{D}_{trn} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{D}_{trn}|}\}$ *containing only normal time series, and* $\mathcal{D}_{test} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{D}_{test}|}\}$ *containing both normal and anomalous time series without labels.*
(2) ***Predict*** *the label* $y_i \in \{-1, +1\}$ *for each time series* $\mathbf{x}_i \in \mathcal{D}_{test}$*, where* $y_i = +1$ *denotes there is at least one anomaly in* $\mathbf{x}_i$*.*

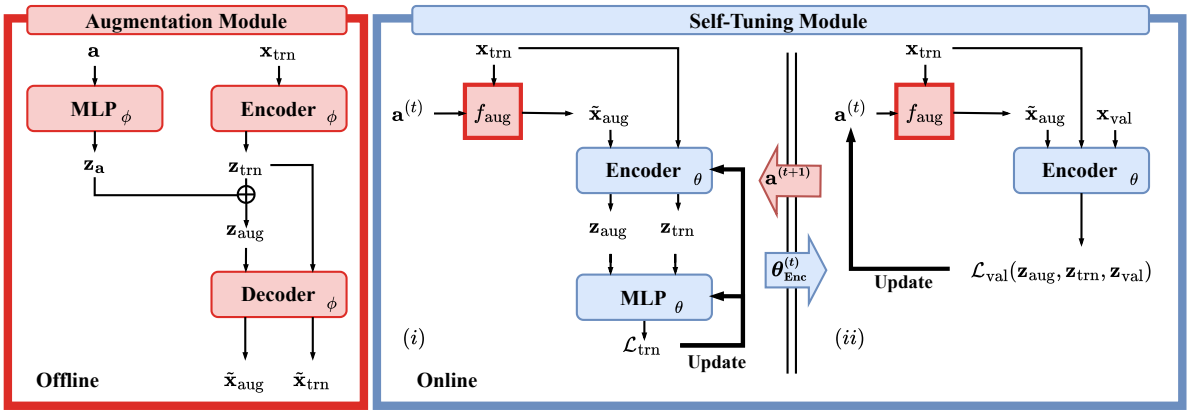


Figure 4.5: TSAP framework for end-to-end self-tuning TSAD. **Left**: Offline trained, differentiable augmentation model $f_{\text{aug}}(\cdot;\boldsymbol{\phi})$ takes as input the normal data and augmentation hyperparameter(s) $\mathbf{a}$, and outputs pseudo-anomalies $\tilde{\mathbf{x}}_{\text{aug}}$. **Right**: Self-tuning engine incorporates the pre-trained f_{aug}, alternating between two phases: (i) detection phase – given $\mathbf{a}^{(t)}$ at iteration t, estimate parameters $\boldsymbol{\theta}^{(t)}$ of detector f_{det}, by optimizing $\mathcal{L}_{\text{trn}}$ (binary classification loss); (ii) alignment phase – given $f^{\text{enc}}_{\text{det}}(\cdot;\boldsymbol{\theta}^{(t)})$, update augmentation (governed by $\mathbf{a}$) to better align the embeddings $\mathbf{z}_{\text{trn}} \cup \mathbf{z}_{\text{aug}}$ with $\mathbf{z}_{\text{val}}$. Note that $\mathbf{x}_{\text{val}}$ contains both normal and anomalous time series, but labels are *not* known or used at any point during training time.

4.3.2 Approach

We aim to use gradient-based optimization to update the continuous hyperparameters of the anomalies, so that we can generate pseudo anomalies and train the anomaly detector in a self-supervised learning (SSL) manner. However, there are two notable challenges:

1. **Differentiable Augmentation**: Developing an augmentation function that is differentiable with respect to its hyperparameters, enabling gradient-based hyperparameter optimization.
2. **Comparable Validation Loss**: Formulating a validation loss that quantifies alignment between $\mathcal{D}_{\text{trn}} \cup \mathcal{D}_{\text{aug}}$ and $\mathcal{D}_{\text{test}}$, which contain both normal and anomalous time series.

To solve these challenges, we propose TSAP with following contributions:

1. **Differentiable Augmentation Module**: In Figure 4.5 left, TSAP implements the augmentation function f_{aug} to generate pseudo anomalies conditioned on $\mathbf{a} \in \mathcal{A}^P$, where $\mathcal{A}^P$ is the domain of all possible hyperparameter values. This module is pre-trained independently, establishing it as an *offline* component of the framework.
2. **Self-Tuning Module**: In Figure 4.5 right, at test time (online), TSAP iteratively refines the detector f_{det}'s parameters $\boldsymbol{\theta}$ and augmentation hyperparameters $\mathbf{a}$, through alternating detection and alignment phases. Alignment is performed on part of the unlabeled $\mathcal{D}_{\text{test}}$, referred to as $\mathcal{D}_{\text{val}}$. This is illustrated in Figure 4.6.

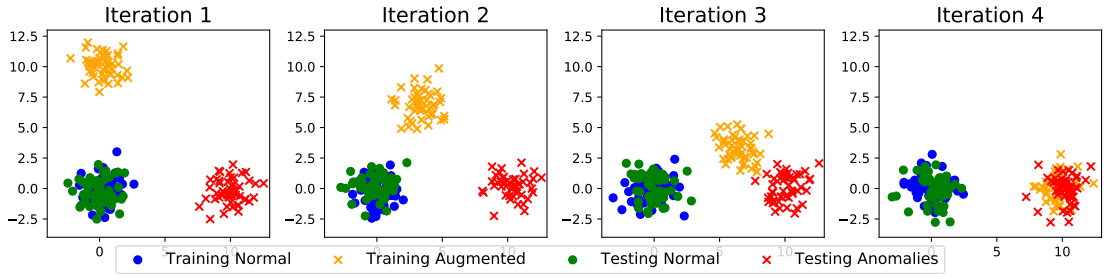


Figure 4.6: TSAP improves alignment between $\mathcal{D}_{\text{trn}} \cup \mathcal{D}_{\text{aug}}$ and $\mathcal{D}_{\text{test}}$ over iterations by tuning $\mathbf{a}$.

Differentiable Augmentation Module. We propose an anomaly generation scheme g by carefully considering 6 types of common time series anomalies; namely, trend, extremum, amplitude, mean shift, frequency shift, and platform (in Figure 4.7). Each type of anomaly has three hyperparameters; including its starting position (`location`), duration (`length`), and severity (`level`). Based on $\mathcal{D}_{\text{trn}}$, g creates an augmented dataset $\mathcal{D}_{\text{aug}} = \{g(\mathbf{x}_{\text{trn}}; \mathbf{a}) \mid \mathbf{x}_{\text{trn}} \in \mathcal{D}_{\text{trn}}, \mathbf{a} \sim \mathcal{A}^P\}$, where $\mathbf{a}$ is randomly sampled in $\mathcal{A}^P$. Its loss function is based on the reconstruction of both $\mathcal{D}_{\text{trn}}$ and $\mathcal{D}_{\text{aug}}$:

$$\mathcal{L}_{\text{aug}} = \frac{1}{|\mathcal{D}_{\text{trn}}|} \sum_{\substack{\mathbf{x}_{\text{trn}} \in \mathcal{D}_{\text{trn}} \\ \mathbf{a} \sim \mathcal{A}^P}} (\mathbf{x}_{\text{trn}} - \tilde{\mathbf{x}}_{\text{trn}})^2 + \big(g(\mathbf{x}_{\text{trn}}, \mathbf{a}) - f_{\text{aug}}(\mathbf{x}_{\text{trn}}; \mathbf{a})\big)^2 \tag{4.1}$$

Self-Tuning Module. The self-tuning module of TSAP operates by iteratively refining the detector's parameters $\boldsymbol{\theta}$, and the augmentation hyperparameters $\mathbf{a}$. The process is structured into two phases:

1. **Detection Phase**: This focuses on estimating the parameters $\boldsymbol{\theta}^{(t)}$ of the detector f_{det} by minimizing the cross-entropy loss $\mathcal{L}_{\text{trn}}$. This aims to classify between the normal samples $\mathbf{x}_{\text{trn}}$ and the augmented pseudo anomalies $\tilde{\mathbf{x}}_{\text{aug}}$ by their embeddings $\mathcal{Z}_{\text{trn}}$ and $\mathcal{Z}_{\text{aug}}$.

2. **Alignment Phase**: This adjusts $\mathbf{a}$ to optimize the unsupervised differentiable validation loss $\mathcal{L}_{\text{val}}$, computed based on the embeddings from the updated $f_{\text{det}}^{\text{enc}}$. Following [111], $\mathcal{L}_{\text{val}}$'s objective is to measure the degree of alignment between $\mathcal{D}_{\text{trn}} \cup \mathcal{D}_{\text{aug}}$ and $\mathcal{D}_{\text{val}}$ in the embedding space by the Wasserstein distance.

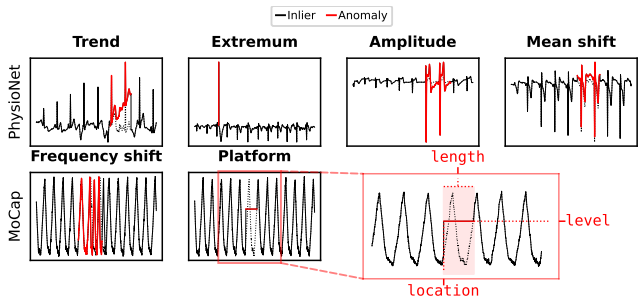


Figure 4.7: Examples of 6 different types of time series anomalies: original time series in black, and pseudo anomalies generated by g in red.

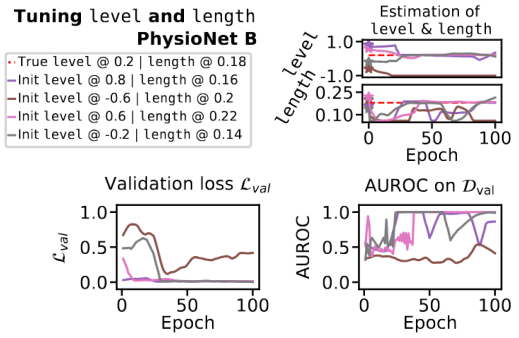


Figure 4.8: **TSAP works.** Augmentation hyperparameters `level` and `length`, are accurately tuned to near-true values (left), guided by val. loss (center), achieving high AUROC (right).

4.3.3 Results

We evaluate TSAP on 6 tasks: 4 in a controlled setting (manually injected anomaly type) using the PhysioNet ECG data, and the remaining 2 with natural anomalies (unknown real-world anomaly type) using the MoCap data. In Table 4.2, TSAP outperforms all baselines in most cases and achieves the best average rank among all baselines. While some competing methods perform strongly on a subset of tasks, they lack consistency across all TSAD tasks. In Figure 4.8, TSAP accurately tunes the hyperparameter of anomalies.

Table 4.2: **TSAP is accurate**, which outperforms most baselines and has the best (lowest) average rank. Detection performance of baselines w.r.t. F_1 and AUROC on test data across 6 tasks.

Methods	PhysioNet A		PhysioNet B		PhysioNet C		PhysioNet D		MoCap A		MoCap B		Avg. Rank	
	F_1	AUROC	F_1	AUROC	F_1	AUROC	F_1	AUROC	F_1	AUROC	F_1	AUROC	F_1	AUROC
OC-SVM	0.182	0.468	0.182	0.472	0.373	0.803	0.393	0.806	1.000	1.000	0.546	0.806	7.2 (3.7)	6.8 (3.3)
LOF	0.999	1.000	0.999	1.000	0.354	0.738	0.358	0.725	0.196	0.506	0.221	0.577	6.8 (3.9)	6.5 (4.4)
ARIMA	0.885	0.960	0.829	0.965	0.991	0.999	0.999	0.999	0.870	0.955	0.225	0.537	4.3 (3.2)	4.7 (3.7)
IF	0.255	0.587	0.232	0.576	0.183	0.402	0.182	0.356	0.864	0.965	0.342	0.758	8.8 (1.7)	8.7 (1.9)
MP	0.812	0.743	0.812	0.744	0.280	0.712	0.284	0.734	1.000	1.000	1.000	1.000	5.0 (3.6)	4.8 (3.4)
EncDec-LSTM	0.190	0.508	0.190	0.508	0.415	0.812	0.442	0.819	0.980	0.999	0.909	0.996	6.5 (2.0)	6.7 (1.9)
SR-CNN	0.965	0.990	0.964	0.998	0.983	0.999	0.991	0.999	0.302	0.700	0.214	0.512	5.2 (4.2)	4.8 (4.5)
USAD	0.183	0.425	0.184	0.428	0.430	0.822	0.409	0.828	1.000	1.000	1.000	1.000	5.5 (4.0)	5.7 (4.5)
NeuTraL-AD	0.211	0.732	0.263	0.679	0.561	0.868	0.526	0.862	1.000	1.000	1.000	1.000	4.2 (2.9)	3.8 (2.5)
TimeGPT	0.327	0.714	0.318	0.711	0.218	0.580	0.217	0.525	0.348	0.743	0.385	0.683	8.0 (1.9)	8.3 (1.6)
TSAP	1.000	1.000	1.000	1.000	0.973	0.999	0.991	0.998	0.889	0.969	1.000	1.000	2.3 (2.0)	2.2 (2.0)

4.4 [Proposed] Detecting and Characterizing Time Series Anomalies Automatically

4.4.1 Goal

Given a multivariate time series dataset, how can we tell what kinds of anomalies are there in the dataset? How can we further leverage these understanding of anomalies to better detect them? In Chapter 4.3, the method only works when the anomalies in the dataset share the same type and hyperparameters and training the decoder in the augmentation model is difficult. Therefore, we aim to improve the previous work by training the encoder with contrastive learning, and searching for the hyperparameters with automated machine learning (AutoML). Our proposed method not only detects the anomalies, but also gives the user insight of them for further analysis.

Problem 8. Point-Level Time Series Anomaly Detection (TSAD)
(1) ***Given*** $\mathcal{D}_{trn} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{D}_{trn}|}\}$ *consisting of only normal time series, and* $\mathcal{D}_{test} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{D}_{test}|}\}$ *containing both normal and anomalous time series without labels.*
(2) ***Predict*** $\mathbf{y}_i = \{y_1, \ldots, y_K\}$ *for each* $\mathbf{x}_i \in \mathcal{D}_{test}$*, where* K *is the length of each time series and* $y_i \in \{-1, +1\}$.

4.4.2 Approach

Given a set of normal time series data, we generate various types of pseudo anomalies, such as platform, mean shift, spike, and frequency. We then pre-train a time series encoder using contrastive learning [70, 83], ensuring that in the embedding space:

1. Anomalies of the same type and with similar hyperparameters should be close.
2. Anomalies of different types should be far apart.
3. Anomalies should be distant from normal data.

Our encoder generates time series embeddings capable of recognizing the hyperparameters of anomaly, such as anomaly severity, duration, and ratio within the dataset. It also ensures a smooth alignment loss using the Wasserstein distance. Therefore, we adopt Bayesian optimization to automatically search for the optimal hyperparameters that minimize the alignment loss between the embeddings of the testing and augmented data (which includes normal and pseudo anomalies).

4.4.3 Future Work

Our proposed method is evaluated using a synthetic dataset containing two types of anomalies, namely platform and mean shift. Using AutoML [37, 67], Bayesian optimization has been shown to precisely search for the hyperparameters of true anomalies. However, when more types of anomalies are included, a more powerful encoder is required. In this case, we consider combining a fixed time series foundation model with a trainable projection head. Finally, we need to enable our method to identify the location of anomalies in the time series data.

Chapter 5

Applications

5.1 Preliminaries

MDL for Anti-Human Trafficking The Minimum Description Length principle (MDL) [76] assumes that the best model $M \in \mathbb{M}$ for data $d_i \in \mathcal{D}$ minimizes $C(\mathcal{D}, M) = C(M) + C(\mathcal{D}|M)$, where $C(d_i)$ is defined as the cost, in bits, needed to describe d_i losslessly. In anti-human trafficking, $\mathcal{D}$ represents a set of escort advertisements, and M are text templates with "slots". Slots are used to represent certain types of information, such as time, location, or age. MDL penalizes both the model cost $C(M)$, as well as the encoding of errors from the model $C(\mathcal{D}|M)$. In short, it automatically searches for the best text templates M by minimizing the bits $C(M)$ needed to describe the templates, and the bits $C(\mathcal{D}|M)$ needed to recover the ads from templates.

5.2 DELTASHIELD: Information Theory for Human-Trafficking Detection

Section based on works appeared at ICDE 2021 [49][PDF] and TKDD 2023 [89][PDF].

5.2.1 Goal

Given a million escort advertisements, how can we spot near-duplicates? Such micro-clusters of ads are usually signals of human trafficking. How can we visually summarize them to convince law enforcement to act? That is, given a set of text documents, our goal is to propose an algorithm for near-duplicate detection and summarization. These near-duplicates are usually recognized as organized crime activity in human trafficking, or spam in Twitter bot detection.

Problem 9. Near-Duplicate Detection and Summarization
(1) ***Given*** *a set of text documents.*
(2) ***Detect*** *near-duplicates/micro-clusters among documents.*
(3) ***Summarize*** *the documents with the templates visually.*

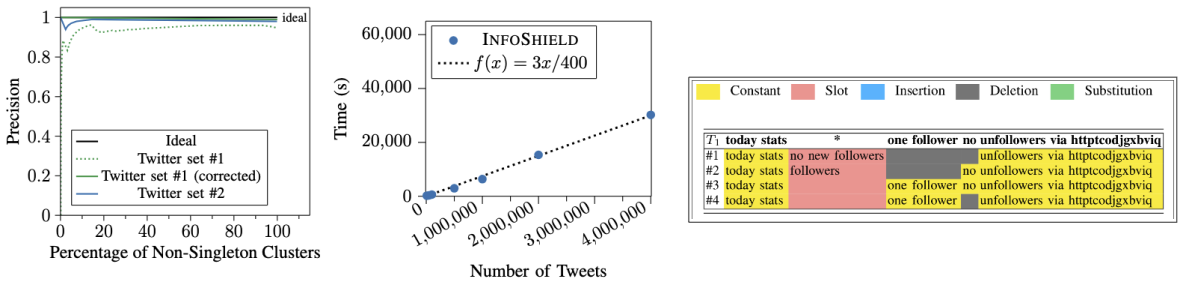
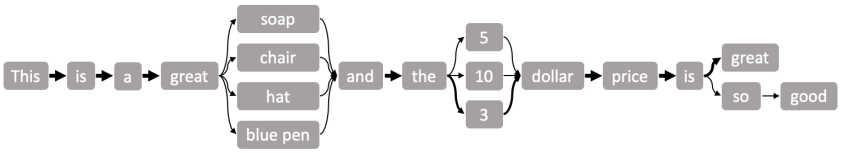


Figure 5.1: **INFOSHIELD works:** being precise (left), scalable (middle), and interpretable (right), detecting and visualizing slots (in red), i.e. portions of tweets that highly differ between otherwise duplicate documents.

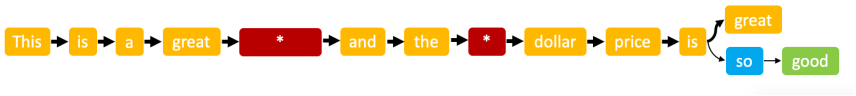
5.2.2 Approach

We propose INFOSHIELD, an information theory based algorithm, which uses the Minimum Description Length (MDL) principle to find good templates. A template represents a cluster of text documents, with "slots", i.e., parts of the template that differ for each document (highlighted red in Figure 5.1). Next, we propose DELTASHIELD, an incremental version of INFOSHIELD, which incrementally updates the templates without re-doing the template search process. It automatically assigns the new incoming documents to the existing templates, or generates new templates if needed.



(a) Step 1: Candidate Alignment

(b) Step 2: Consensus Search



(c) Step 3: Slot Detection

Figure 5.2: **Example pipeline of INFOSHIELD** : The output after each step of INFOSHIELD.

INFOSHIELD A quick algorithm is firstly used to create coarse-grained clusters of documents with high text similarity (omit for brevity). Then, INFOSHIELD is composed of three steps:

1. **Candidate Alignment** (Figure 5.2a): identifies the candidate set for a template by aligning documents with multiple sequence alignment (MSA). Given data $\mathcal{D}$ from one cluster, we first represent documents as line graphs and align them with the first document d_1 individually and then calculate the cost $C(d|d_1)$ and $C(d)$ for every $d \in \mathcal{D}$; if $C(d|d_1)$ is smaller than $C(d)$, we add d to the set D_i containing all similar documents found in iteration i. Finally, we generate the alignment A_i with all documents in D_i.
2. **Consensus Search** (Figure 5.2b): searches for the best consensus document in the alignment. We decide which tokens are part of the template, and which are insertions/deletions/substitutions by MDL. We only keep edges between words that occur more than h times in A_i. We search for the best threshold h_i^* to generate the consensus of alignment $T_i^{'}$ at the lowest cost.
3. **Slot Detection** (Figure 5.2c): detects slots in the consensus document to generate a template. We first recognize the operation types of words by each alignment $a \in A_i$, which are either insertions or substitutions. We identify which words each potential slot p contains in the given consensus document $T_i^{'}$, and only keep slots that decrease the total cost and store them in T_i.

DELTASHIELD We propose DELTASHIELD to incrementally update the templates, which includes a preprocessing step and a template update step. The preprocessing step encodes an incoming document using existing templates in its coarse cluster and selects the one that results in the lowest cost. To address the high time complexity of examining all the existing templates, we adopt an early-stopping (ES) mechanism. We order the templates by the lengths of intersection between unigrams in the incoming document and each template. Then, we select the first template that lowers the encoding cost of the document. Once the new documents are added to a template, its representation will be slightly changed. Hence, we update the template every t batch.

5.2.3 Results

We report the experiments in Table 5.1. In Twitter data, INFOSHIELD always performs within ten points of the top contender, despite using no features specific to Twitter. In HT data, INFOSHIELD reports the highest precision; this avoids giving false positives to law enforcement.

For interpretability, in Table 5.2, we give a description of the type of text they represent for the slots. Notice that slots tend to include consistent user-specific information. For example, the second slot, if not empty, always discusses time, "until 9pm", "9 P.M", etc.

Next, we compare DELTASHIELD with update frequency every batch and every three batches. In Figure 5.3a, as the number of incoming batches increases, the gap between two methods also increases. Nevertheless, the running time of updating every three batches in Figure 5.3b is 1.4 times and 2.8 times faster than the one of updating every batch and INFOSHIELD, respectively.

Twitter Data

Dataset	Test Set #1				Test Set #2			
Metric	ARI	Prec.	Rec.	F1	ARI	Prec.	Rec.	F1
INFOSHIELD	83.2	93.0	91.2	92.1	75.7	96.7	88.9	92.6
Cresci	n/a	98.2	97.2	97.7	n/a	100.0	85.8	92.3
BotOrNot	n/a	47.1	20.8	28.9	n/a	63.5	**95.0**	76.1
Yang	n/a	56.3	17.0	26.1	n/a	72.7	40.9	52.4
Ahmed	n/a	94.5	94.4	94.4	n/a	91.3	93.5	92.3

Human Trafficking Data

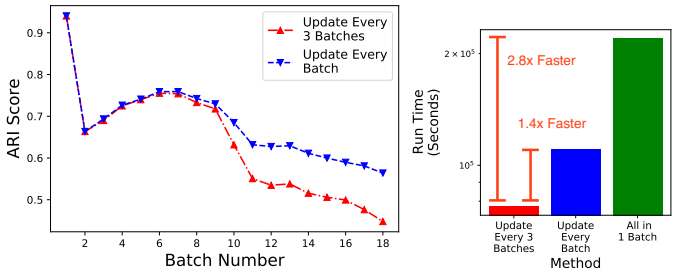
Dataset	Trafficking10k			Cluster Trafficking			
Metric	Prec.	Rec.	F1	Prec.	Rec.	F1	ARI
INFOSHIELD	84.8	50.7	63.5	85.4	99.8	92.0	43.1
Word2Vec-cl	19.4	10.7	13.8	71.7	99.5	83.1	9.6
Doc2Vec-cl	25.6	10.9	15.3	74.2	98.8	84.7	16.2
FastText-cl	28.4	22.4	25.1	69.6	99.6	81.9	6.8
HTDN	71.4	62.2	66.5	—	—	—	n/a

Table 5.1: **INFOSHIELD performs well**: INFOSHIELD outperforms or approaches the best *domain-specific* method in both settings. shows the best score, shows the methods within 10 points of the best. are supervised method, while INFOSHIELD is unsupervised.

Constant Slot Insertion Deletion Substitution

T_1	**not shown for victim's safety**							
#1	(empty)		time		(empty)		(empty)	
#2	personal description		time		(empty)		cost	
#3	(empty)		time		(empty)		cost	
#4	personal description		(empty)		preferences		cost	
...18 similar ads								

Table 5.2: **INFOSHIELD works for human-trafficking detection**: detects the template from HT dataset.



(a) ARI Score Over Time (b) Total Run Time

Figure 5.3: **DELTASHIELD offers strong trade-off**: (a) shows large update frequency loses effectiveness increasingly over time (20K / batch); (b) shows increasing update frequency leads to a much lower run time.

5.3 [Proposed] Fraud Detection in Temporal Financial Graphs

5.3.1 Goal

Given a bipartite temporal graph, how can we detect anomalous edges? In a financial graph, the source and target nodes represent customers and merchants, respectively, and the edges correspond to the transactions between them. More specifically, how can we detect fraudulent transactions in such a financial graph? In addition, fraudulent activities can be categorized into different types. Our goal is to propose an interpretable anomaly detection method that identifies both point and group anomalies, while also providing explanations to the user.

Problem 10. Fraud Detection in Temporal Financial Graphs
(1) ***Given*** *a bipartite and temporal financial graph.*
(2) ***Detect*** *fraudulent transactions.*
(3) ***Explain*** *to the user why these transactions are recognized as fraudulent.*

5.3.2 Approach

We plan to separate our method into three major steps:

1. **Feature Selection**: Given thousands of features, most are not useful for detecting anomalies, and including them may significantly hurt performance. Therefore, using a few labeled anomalies, we plan to apply sequential forward feature selection to find a small subset of effective features.
2. **Graph Feature Construction**: We plan to construct features using linear GNNs. However, in the bipartite graph, merchants do not have features, so we will propagate the customer features from two steps away.
3. **Group Anomaly Detection**: We plan to begin by using GEN^{2}OUT, introduced in Chapter 4.2.

5.3.3 Future Work

In the future, we plan to improve each step of our method. For feature selection, due to the small fraction of labeled data, the final selection may easily overfit. Furthermore, features that are highly correlated with the selected ones should be excluded. For graph feature construction, relying on two-step neighbors may not be the most effective approach for spotting anomalies, and this requires further analysis. Utilizing temporal features with feature propagation also remains a challenge. For group anomaly detection, we aim to further enhance GEN^{2}OUT, improving its effectiveness.

5.4 [Proposed] An Agentic Framework for Graph Retrieval-Augmented Generation

5.4.1 Goal

Given a question, a database containing unstructured documents and knowledge graphs (KGs), can we tell which context from the database is useful for a large language model (LLM) to answer the question? In addition to a single text knowledge base in Retrieval-Augmented Generation (RAG), Graph RAG (GRAG) includes KGs as an additional knowledge base. In other words, in GRAG, can we identify whether we need the context from a graph retriever, a text retriever, or neither, to answer the question? Providing an LLM with all available contexts can introduce noise, potentially leading to an incorrect answer. Moreover, given only one opportunity, it is difficult to make the correct decision directly. Therefore, we aim to propose an agentic framework for GRAG.

Problem 11. Graph Retrieval-Augmented Generation (GRAG)
(1) ***Given*** *a database that includes free-text documents and knowledge graphs (KGs).*
(2) ***Select*** *the appropriate retriever to retrieve the context for answer generation.*
(3) ***Optimize*** *the action for retriever-use through self-reflection.*

5.4.2 Approach

We propose an agentic framework for GRAG that includes two novel contributions:

1. **Retriever-Use**: Similarly to the terminology tool-use, the agent interacts with the retriever bank and selects the most appropriate retriever to answer the question. Our proposed retriever bank contains both text and graph retriever modules. They can be used for solving questions that need contexts from different sources, e.g., web search and KGs.
2. **Self-Reflection**: We propose a critic that includes a validator and a commentor. The validator decides whether to continue optimizing the agent's action based on the correctness of the output. If the output is rejected, the commentor leverages in-context learning and generates feedback for the agent to adjust its action.

Moreover, our method can be used with any pre-trained LLMs, which avoids expensive fine-tuning and issues such as catastrophic forgetting [86].

5.4.3 Future Work

We are currently working on implementing our method and the baselines on two GRAG benchmarks, namely STARK [99] and CRAG [105]. While STARK focuses on evaluating the retrieval capability of our method, CRAG focuses on evaluating both the retrieval and generation capabilities. We also need a comprehensive ablation study to justify the design choices of our method.

Chapter 6

Timelines

My expected timeline is provided in Table 6.1.

Table 6.1: Expected Timeline.

Time	Tasks	Expected Outputs
October 2024	Thesis Proposal	
November 2024 - February 2025	Paper Writing (Chapter 4.4, 5.3, and 5.4)	Three New Conference Papers
March 2025 - May 2025	Job Application and Interview	
June 2025 - August 2025	Thesis Writing	
September 2025	Thesis Defense	

Chapter 7

Conclusions

In this thesis, we propose various explainable approaches to address the limitations of black-box machine learning methods. These methods are either inherently explainable, or provide explanations of the data or decision-making process to users. Specifically, we focus on designing algorithms and solving applications related to graph and time series data. In the first and second parts of this thesis, we propose graph mining algorithms to solve node-level and graph-level tasks, respectively. Our node-level algorithm outperforms competitors on *11* out of *12* datasets in link prediction, while our graph-level algorithm improves mean absolute error by *14.3%* in graph regression. In the third part, we introduce time series anomaly detection algorithms that provide users with insights into the datasets. Our group anomaly detection method requires only *2* minutes to run for *1* million data points. In the final part, we address various applications using graph algorithms, including anti-human trafficking, fraud detection in financial graphs, and graph retrieval-augmented generation. Our method detects human trafficking with *84%* precision. For reproducibility and the benefit of the community, we make most of the algorithms and datasets used in this thesis publicly available at `https://mengchillee.github.io/`.

Bibliography

[1] 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 *international conference on machine learning*, pages 21–29. PMLR, 2019. A

[2] Faraz Ahmed and Muhammad Abulaish. A generic statistical approach for spam detection in online social networks. *Comput. Commun.*, 36(10-11):1120–1129, 2013. A

[3] Nesreen K Ahmed, Ryan Rossi, John Boaz Lee, Theodore L Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. Learning role-based graph embeddings. *arXiv preprint arXiv:1802.02896*, 2018. B

[4] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Proceedings of PAKDD*, pages 410–421. Springer, 2010. 3.1

[5] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015. 3.1

[6] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. USAD: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD*, page 3395–3404. ACM, 2020. A

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*, 5:135–146, 2017. A

[8] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015. A

[9] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Identifying density-based local outliers. In *SIGMOD*, page 93–104, 2000. ISBN 1581132174. A

[10] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*, 2022. A

[11] Raghavendra Chalapathy, Edward Toth, and Sanjay Chawla. Group anomaly detection using deep generative models. In *ECML-PKDD*, pages 173–189, 2018. 4.1

[12] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020. A

[13] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. A

[14] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*. John Wiley & Sons, 2004. 1

[15] Gari Clifford, Chengyu Liu, Benjamin Moody, Li-Wei Lehman, Ikaro Silva, Qiao Li, Alistair Johnson, and Roger Mark. AF classification from a short single lead ECG recording. In *CinC 2017*. Computing in Cardiology, September 2017. B

[16] Diane J Cook and Lawrence B Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1993. 3.1, 3.2.2, 1

[17] Luca Cosmo, Giorgia Minello, Michael Bronstein, Emanuele Rodolà, Luca Rossi, and Andrea Torsello. Graph kernel neural networks. *arXiv preprint arXiv:2112.07436*, 2021. 3.1

[18] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. Dna-inspired online behavioral modeling and its application to spambot detection. *IEEE Intell. Syst.*, 31(5):58–64, 2016. A

[19] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In *WWW*, page 963–972, 2017. ISBN 9781450349147. B

[20] Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. Botornot: A system to evaluate social bots. In *WWW*, page 273–274, 2016. A

[21] Michael Davis, Weiru Liu, Paul Miller, and George Redpath. Detecting anomalies in graphs with numeric labels. In *Proceedings of the ACM CIKM*, pages 1197–1202, 2011. 3.1

[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016. A

[23] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. B

[24] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1386, 2018. 3.1

[25] Dhivya Eswaran, Srijan Kumar, and Christos Faloutsos. Higher-order label homogeneity and spreading in graphs. In *TheWebConf*, pages 2493–2499, 2020. A

[26] Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. KerGNNs: Interpretable graph neural networks with graph kernels. *ArXiv Preprint: https://arxiv. org/abs/2201.00491*, 2022. 3.1

[27] Azul Garza and Max Mergenthaler-Canseco. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023. A

[28] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and single-pass belief propagation. *PVLDB*, 8(5):581–592, 2015. 2.1, A

[29] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the ACM SIGKDD*, pages 855–864, 2016. A

[30] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *ICML*, pages 2712–2721, 2016. 4.1

[31] Nikhil Gupta, Dhivya Eswaran, Neil Shah, Leman Akoglu, and Christos Faloutsos. Beyond outlier detection: Lookout for pictorial explanation. In *Proceedings of the ECML*, pages 122–138. Springer, 2018. 3.1

[32] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 2.1, 2.3.1

[33] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017. A

[34] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. B

[35] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS*, 33:22118–22133, 2020. B, B

[36] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *ICLR*, 2021. A

[37] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019. 4.4.3

[38] U Kang, Jay-Yoon Lee, Danai Koutra, and Christos Faloutsos. Net-ray: visualizing and mining billion-scale graphs. In *Proceedings of the PAKDD*, pages 348–361. Springer, 2014. 3.1

[39] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*, 2021. A

[40] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 2.1, 2.3.1, 6, A

[41] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018. A

[42] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019. A

[43] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of European Conference on Machine Learning*, pages 217–226. Springer, 2004. B.2, B

[44] Danai Koutra, Tai-You Ke, U. Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *ECML PKDD*, volume 6912, pages 245–260. Springer, 2011. 2.1, A

[45] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *KDD*, pages 444–452, 2008. 4.2.2

[46] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In

ICML, page II–1188–II–1196, 2014. A

[47] Meng-Chieh Lee, Hung T Nguyen, Dimitris Berberidis, Vincent S Tseng, and Leman Akoglu. Gawd: graph anomaly detection in weighted directed graph databases. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 143–150, 2021. 3.2

[48] Meng-Chieh Lee, Shubhranshu Shekhar, Christos Faloutsos, T Noah Hutson, and Leon Iasemidis. Gen 2 out: Detecting and ranking generalized anomalies. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 801–811. IEEE, 2021. 4.2

[49] Meng-Chieh Lee, Catalina Vajiac, Aayushi Kulshrestha, Sacha Levy, Namyong Park, Cara Jones, Reihaneh Rabbany, and Christos Faloutsos. Infoshield: Generalizable information-theoretic human-trafficking detection. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1116–1127. IEEE, 2021. 5.2

[50] Meng-Chieh Lee, Shubhranshu Shekhar, Jaemin Yoo, and Christos Faloutsos. Neteffect: Discovery and exploitation of generalized network effects. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 299–312. Springer, 2024. 2.2, 2.2.2

[51] Meng-Chieh Lee, Haiyang Yu, Jian Zhang, Vassilis N. Ioannidis, Xiang song, Soji Adeshina, Da Zheng, and Christos Faloutsos. Netinfof framework: Measuring and exploiting network usable information. In *The Twelfth International Conference on Learning Representations*, 2024. 2.4, 2.4.2, 2.4.2

[52] Meng-Chieh Lee, Lingxiao Zhao, and Leman Akoglu. Descriptive kernel convolution network with improved random walk kernel. In *Proceedings of the ACM on Web Conference 2024*, pages 457–468, 2024. 3.3

[53] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning*, pages 2024–2033. PMLR, 2017. 3.1

[54] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014. B

[55] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005. B

[56] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019. A

[57] Mingjie Li, Xiaojun Guo, Yifei Wang, Yisen Wang, and Zhouchen Lin. G^2cn: Graph gaussian convolution networks with concentrated graph filters. In *ICML*, 2022. A

[58] Mingjie Li, Xiaojun Guo, Yifei Wang, Yisen Wang, and Zhouchen Lin. G^2cn: Graph gaussian convolution networks with concentrated graph filters. In *International Conference on Machine Learning*, pages 12782–12796. PMLR, 2022. 2.1

[59] Peiyuan Liao, Han Zhao, Keyulu Xu, Tommi Jaakkola, Geoffrey J Gordon, Stefanie Jegelka, and Ruslan Salakhutdinov. Information obfuscation of graph neural networks. In *International conference on machine learning*, pages 6600–6610. PMLR, 2021. 2.1

[60] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *NeurIPS*, 2021. B

[61] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *ICDM*, pages 413–422. IEEE, 2008. 4.1, 4.2.2, A

[62] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *KDD*, 2020. A

[63] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016. A

[64] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013. A

[65] Krikamol Muandet and Bernhard Schölkopf. One-class support measure machines for group anomaly detection. *arXiv preprint arXiv:1303.0309*, 2013. 4.1

[66] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017. A

[67] Renato Negrinho, Matthew Gormley, Geoffrey J Gordon, Darshan Patil, Nghia Le, and Daniel Ferreira. Towards modular and programmable architecture search. *Advances in neural information processing systems*, 32, 2019. 4.4.3

[68] Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. *Advances in Neural Information Processing Systems*, 33:16211–16222, 2020. 3.1, 6

[69] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ACM SIGKDD*, pages 631–636, 2003. 3.1, 3.2.2, 1

[70] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 4.4.2

[71] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social networks*, 31(2): 155–163, 2009. B.2, B

[72] Tomáš Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2): 275–304, 2016. 4.1

[73] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. Neural transformation learning for deep anomaly detection beyond images. In *ICML*, pages 8703–8714, 2021. A

[74] Shebuti Rayana. Odds library, 2016. URL `http://odds.cs.stonybrook.edu`. B

[75] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proc. of the 25th ACM SIGKDD*, 2019. ISBN 9781450362016. A

[76] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, September

1978. 5.1

[77] Benedek Rozemberczki and Rik Sarkar. Twitch gamers: A dataset for evaluating proximity preserving and structural role-based node mmbeddings. *arXiv preprint arXiv:2101.03091*, 2021. B

[78] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019. B

[79] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *J. Complex Networks*, 9(2), 2021. B

[80] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *NIPS*, 1999. A

[81] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008. B

[82] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. B

[83] Alessandro Sordoni, Nouha Dziri, Hannes Schulz, Geoff Gordon, Philip Bachman, and Remi Tachet Des Combes. Decomposed mutual information estimation for contrastive representation learning. In *International Conference on Machine Learning*, pages 9859–9869. PMLR, 2021. 4.4.2

[84] Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International Scientific Conference and International Workshop Present Day Trends of Innovations*, volume 1, 2012. B

[85] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *KDD*, 2009. B

[86] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019. 5.4.2

[87] Edmund Tong, Amir Zadeh, Cara Jones, and Louis-Philippe Morency. Combating human trafficking with multimodal deep models. In *ACL*, pages 1547–1556, Vancouver, Canada, 2017. A, B

[88] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012. B

[89] Catalina Vajiac, Meng-Chieh Lee, Aayushi Kulshrestha, Sacha Levy, Namyong Park, Andreas Olligschlaeger, Cara Jones, Reihaneh Rabbany, and Christos Faloutsos. Deltashield: Information theory for human-trafficking detection. *ACM Transactions on Knowledge Discovery from Data*, 17(2):1–27, 2023. 5.2

[90] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018. A

[91] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 2.1

[92] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 2.3.1

[93] Tal Wagner, Sudipto Guha, Shiva Kasiviswanathan, and Nina Mishra. Semi-supervised learning on data streams via temporal label propagation. In *International Conference on Machine Learning*, pages 5095–5104. PMLR, 2018. 2.1

[94] Borui Wang and Geoffrey Gordon. Learning general latent-variable graphical models with predictive belief propagation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6118–6126, 2020. 2.1

[95] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020. B

[96] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *NeurIPS*, 2021. A

[97] Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *Advances in Neural Information Processing Systems*, 34:5758–5769, 2021. 2.1

[98] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019. 2.1, A

[99] Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. Stark: Benchmarking llm retrieval on textual and relational knowledge bases. *arXiv preprint arXiv:2404.13207*, 2024. 5.4.3

[100] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021. A

[101] Liang Xiong, Barnabás Póczos, Jeff Schneider, Andrew Connolly, and Jake VanderPlas. Hierarchical probabilistic models for group anomaly detection. In *AISTATS*, pages 789–797, 2011. 4.1

[102] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. A

[103] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 433–444, 2008. B

[104] Chao Yang, Robert Chandler Harkreader, and Guofei Gu. Empirical evaluation and new

design for fighting evolving twitter spammers. *IEEE Trans. Inf. Forensics Secur.*, 8(8): 1280–1293, 2013. A

[105] Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, et al. Crag–comprehensive rag benchmark. *arXiv preprint arXiv:2406.04744*, 2024. 5.4.3

[106] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016. B

[107] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *IEEE ICDM*, pages 1317–1322, 2016. A

[108] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018. A

[109] Jaemin Yoo, Hyunsik Jeon, and U Kang. Belief propagation network for hard inductive semi-supervised learning. In *IJCAI*, 2019. A

[110] Jaemin Yoo, Meng-Chieh Lee, Shubhranshu Shekhar, and Christos Faloutsos. Less is more: Slimg for accurate, robust, and interpretable graph mining. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3128–3139, 2023. 2.3

[111] Jaemin Yoo, Lingxiao Zhao, and Leman Akoglu. End-to-end augmentation hyperparameter tuning for self-supervised anomaly detection. *arXiv preprint*, 2023. 2

[112] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 647–657, 2019. 3.1

[113] Minji Yoon, Théophile Gervet, Baoxu Shi, Sufeng Niu, Qi He, and Jaewon Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2046–2056, 2021. A

[114] Rose Yu, Xinran He, and Yan Liu. Glad: group anomaly detection in social media analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(2):1–22, 2015. 4.1

[115] Ge Zhang, Zhenyu Yang, Jia Wu, Jian Yang, Shan Xue, Hao Peng, Jianlin Su, Chuan Zhou, Quan Z Sheng, Leman Akoglu, et al. Dual-discriminative graph neural network for imbalanced graph-level anomaly detection. In *Advances in Neural Information Processing Systems*, 2022. A, B

[116] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. A

[117] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *ICLR*, 2021. A

[118] Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2021. 2.1

[119] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems*, 33:7793–7804, 2020. A

Appendix A

Related Works

Message Passing Baselines

FABP [44] and LINBP [28] accelerate BP by approximating the final assignment of beliefs. HOLS [25] is a BP-based method, which propagates labels by weighing with higher-order cliques.

There exist many recent GNN variants; recent surveys [100, 116] group them into spectral models [22, 40, 102], sampling-based models [33, 108, 113, 119], attention-based models [10, 39, 62, 90], and deep models with residual connections [12, 56]. Decoupled models [13, 41, 42] separate the two main functionalities of GNNs: the node-wise feature transformation and the propagation. MIXHOP [1], GPRGNN [13], and H_2GCN [119] make no assumption of homophily. GNNs are often fused with graphical inference [36, 109] to further improve the predicted results.

The first linear GNN, SGC [98], removes the nonlinear activation functions of GCN, reducing the propagator function to a simple matrix multiplication. [96] and [117] improved SGC by manually adjusting the strength of self-loops with hyperparameters, increasing the number of propagation steps. G^2CN [57] improves the accuracy of DGC [96] on heterophily graphs by combining multiple propagation settings (i.e. bandwidths).

In graph anomaly detection, graph embedding can be used to detect anomalous graphs in conjunction with off-the-shelf anomaly detectors. node2vec [29] could highly identify graph structures with biased random walks using BFS and DFS. graph2vec [66] uses document embedding neural networks to embed node-labeled graphs. Node embedding methods could also be used in graph embedding by averaging the embedding of nodes. iGAD [115] incorporates random walk kernels as a structural feature extractor to identify graph-level anomalies.

Time Series Anomaly Detection (TSAD) Baselines

The traditional TSAD methods consist of different modeling approaches; namely, one-class support vector machines [80]; local outlier factor [9]; ARIMA [8]; isolation forest [61]; and matrix profile [107]. On the deep learning side, we benchmark against the encoder-decoder LSTM [63]; spectral residual cnvolutional neural network [75]; unsupervised anomaly detection for TSAD [6]; a time series foundation model called TimeGPT [27]; and a method that learns augmentations in the embedding space, called neural transformation learning [73].

Near-Duplicate Detection Baselines

Most state-of-the-art methods for HT detection are not open-source. Instead, we compare against HTDN [87], which uses the same Trafficking10k dataset, and develop three baselines using state-of-the-art text embedding methods Word2Vec [64], FastText [7], and Doc2Vec [46]. In Twitter data, we compare with three supervised methods [2, 20, 104] and one unsupervised method [18]. All of these methods use Twitter-specific features, such as number of mentions, favorites, retweets, posting time, etc.

Appendix B

Datasets

Real-World Homogeneous Graphs

Table B.1: **Graph dataset statistics.** The first 7 datasets are homophily, and the last 6 are heterophily graphs.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1433	7
CiteSeer	3,327	4,732	3703	6
PubMed	19,717	44,338	500	3
Computers	13,752	245,861	767	10
Photo	7,650	119,081	745	8
ogbn-arXiv	169,343	1,166,243	128	40
ogbn-Products	2,449,029	61,859,140	100	30
Chameleon	2,277	36,101	2325	5
Squirrel	5,201	216,933	2089	5
Actor	7,600	29,926	931	5
Penn94	41,554	1,362,229	4814	2
Twitch	168,114	6,797,557	7	2
Pokec	1,632,803	30,622,564	65	2

In Chapter 2.2, we include 3 citation networks: "arXiv-Year" [35], "Patent-Year" [55], and "arXiv-Category" [95], and 4 social networks: "Pokec-Gender" [84], "Facebook" [78], "GitHub" [78], and "Pokec-Locality" [84].

In Chapter 2.3 and 2.4, we use 7 homophily and 6 heterophily graph datasets in experiments. Table B.1 shows a summary of dataset information. Cora, CiteSeer, and PubMed [81, 106] are homophily citation graphs between research articles. Computers and Photo [82] are homophily Amazon co-purchase graphs between items. ogbn-arXiv and ogbn-Products are large homophily graphs from Open Graph Benchmark (OGB) [34]. Since we use only 2.5% of all labels as training data, we omit the classes with instances fewer than 100. Chameleon and Squirrel [79] are heterophily Wikipedia web graphs. Actor [85] is a heterophily graph connected by the co-occurrence of actors on Wikipedia pages. Penn94 [60, 88] is a heterophily graph of gender relations in a social network. Twitch [77] and Pokec [54] are large graphs, which have been relabeled by [60] to be heterophily.

In Chapter 2.4, we also conduct experiments on 3 link prediction datasets from OGB [34], namely ogbl-ddi, ogbl-collab, and ogbl-ppa.

Graph Databases

Table B.2: **Summary of graph databases.**

Name	Graphs	Nodes [min, max]	Edges [min, max]
UCI Message Dataset [71]	3320	[2, 159]	[1, 193]
Enron Email Dataset [43]	843	[2, 87]	[1, 127]
Accounting Dataset	16,026	[2, 13]	[1, 20]
Random Accounting Dataset	15,935	[2, 13]	[1, 18]

In Chapter 3.2, we use four datasets illustrated in Table B.2. The detailed description of all datasets are shown as follows:

- **UCI Message Dataset [71]:** This recorded the communications between students at UCI where nodes and edges denote students and messages respectively. To capture the role information, we adopt role2vec [3] to embed nodes in the complete graph, and use the 10 groups clustered by Agglomerative Clustering as the node labels. The data is split into hours to form a graph database.
- **Enron Email Dataset [43]:** This contains the emails passing between colleagues in Enron Company from 2000 to 2002. We assign the job positions to each employee as node labels. The data is split into day communication graphs to form a graph database.
- **Accounting Dataset:** This is from an anonymous institution, containing accounts (nodes) and transactions (edges) that precisely reflect the money flow between company accounts. Each graph captures a set of transactions within a unique expense report.
- **Random Accounting Dataset:** Since the accounting dataset is proprietary, we generate a synthetic database with generated graphs following the same statistical characteristics as in the accounting graphs.

In Chapter 3.3, we evaluate supervised graph anomaly detection with 10 real-world datasets from PubChem [103], as in [115]. Each graph is a chemical compound and labeled by its outcome from anti-cancer screen tests (active or inactive). For graph regression and classification, we use three real-world datasets, ZINC [23], ogbg-molhiv and ogbg-molpcba [35].

Anomaly Detection Datasets

In Chapter 4.2, we analyzed intracranial electroencephalographic (EEG) signals recorded at the Epilepsy Monitoring Unit of a large public universityfrom one patient with refractory epilepsy. Electrodes were stereotactically placed in the brain and EEG signals were then recorded across 122 electrode contacts at a sampling rate of 2KHz with focal region in the right temporal lobe.

Our benchmark set consist of real-world outlier detection datasets from ODDS repository [74]. The datasets cover diverse application domains and have diverse range dimensionality and outlier percentage. The ODDS datasets provide ground truth outliers that we use for the quantitative evaluation of the methods.

In Chapter 4.3, for controlled tasks, we use the 2017 PhysioNet Challenge dataset [15], which includes real-world ECG recordings. The natural TSAD tasks are derived from the CMU Motion Capture (MoCap) dataset [1]. Table B.3 shows the anomaly profiles of all TSAD tasks.

Table B.3: **Anomaly profile of different TSAD tasks.**

	Dataset	Type	Level	Location	Length
PhysioNet	PhysioNet A	Platform	Fixed	Random	Random
	PhysioNet B	Platform	Fixed	Random	Fixed
	PhysioNet C	Trend	Fixed	Random	Random
	PhysioNet D	Trend	Fixed	Random	Fixed
MoCap	MoCap A	Jump	Fixed	Random	Random
	MoCap B	Run	Fixed	Random	Random

Near-Duplicate Detection Datasets

In Chapter 5.2, we use Twitter bot detection data from [19]. This data includes the tweet text and user id. The Trafficking 10k dataset is created in [87], where expert annotators manually labeled 10,265 ads from 0-6. 0 represents "Not Trafficking", 3 represents "Unsure", and 6 represents "Trafficking".There are 6,551 ads labeled as not HT, 354 labeled as "Unsure", and 3,360 labeled as HT. Cluster Trafficking is a new dataset provided by Marinus Analytics. This data contains cluster labels, provided by domain experts, for both HT and spam advertisements. It consists of 157,258 ads, with 6,283 spam ads, 50,985 HT ads, and 99,990 normal ads.

[1] http://mocap.cs.cmu.edu/