ALGORITHMIC FOUNDATION OF FAIR GRAPH MINING

BY

JIAN KANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Doctoral Committee:

Associate Professor Hanghang Tong, Chair
Professor Jiawei Han
Professor Ross Maciejewski, Arizona State University
Assistant Professor Han Zhao

ABSTRACT

In an increasingly connected world, graph mining plays a fundamental role in many real-world applications, such as financial fraud detection, drug discovery, traffic prediction, and so on. Years of research in this area have developed a wealth of theories, algorithms, and systems that are successful in answering *what/who* types of questions, e.g., *who* is most influential in a social network? *What* item should we recommend to a user? Despite the remarkable progress in graph mining, unfairness often occurs in many graph mining tasks. As such, a fundamental question largely remains nascent: *how* can we make graph mining process and its results fair?

To answer this question, it is crucial to propose a paradigm shift, from answering *what* and *who* to answering *how* and *why*. Four desired properties are called for to build an algorithmic foundation of fair graph mining, including: utility that promises strong empirical performance in the mining task, fairness that avoids discriminatory performances over diverse sensitive groups or individuals, robustness that enhances the resilience toward noise or adversarial activities in the complex world, and transparency that renders the accountability and explainability of graph mining algorithms.

The tensions among the desired properties require us to address three key challenges, namely the *auditing* challenge, the *debiasing* challenge, and the *safeguarding* challenge. First, the *auditing* challenge requires to address the tension between utility and transparency by understanding how the mining results of a given graph mining model relate to the input graph. Second, the *debiasing* challenge asks for balancing the trade-off between utility and fairness so as to ensure fairness on graph mining without much sacrifice on its utility. Third, the *safeguarding* challenge connects utility, fairness, robustness, and transparency together, and studies the tensions among them, which could help the deployment of fair graph mining techniques in the real world.

The theme of my Ph.D. research is to build an algorithmic foundation of fair graph mining by developing computational models underpinning all three pillars, namely auditing, debiasing, and safeguarding, to address these key challenges. First, for *auditing*, we develop a family of algorithms Aurora to audit PageRank algorithm from the edge, node, and subgraph level, and a generic algorithmic framework N2N that audits a variety of graph mining algorithms from the optimization perspective. Moreover, we develop JuryGCN, which is the *first* frequentist-based approach to quantify node uncertainty of graph convolutional network without any epoch(s) of model training. JuryGCN is proven to be useful in both active

learning on node classification and semi-supervised node classification, and achieves the best effectiveness and lowest memory usage than the competitors. Second, for *debiasing*, we offer the *first* systematic study of individual fairness on graph mining (InFoRM), including the measurement, mitigation strategies, and cost. We also design a family of algorithms RawlsGCN to debias degree unfairness by analyzing its mathematical root cause. Moreover, we ensure fairness among intersectional groups from the information-theoretic perspective. Third, for *safeguarding*, we explore the adversarial robustness of fair graph mining algorithms by attacking them with a meta learning-based attacking framework named Fate. The developed framework is broadly applicable to various fairness definitions and graph learning models, as well as arbitrary choices of manipulation operations. We also conduct analysis on the poisoned edges to reveal edges with which property would contribute most to the bias amplification on graph neural networks.

ACKNOWLEDGMENTS

First and foremost, I am profoundly grateful to my advisor, Hanghang Tong, for his exceptional guidance, invaluable advice, unwavering support, patience, and encouragement throughout my Ph.D. journey. The significance of his contributions cannot be overstated. He has taught me so many things through the years of study, including conducting cutting-edge and impactful research, mentoring students, giving effective presentations, writing grant proposals, attending PI meetings, and negotiating job offers. His high standards for effective, concise writing and rigorous research methods have brought my own to a higher level. His principles of 'aim high' and 'be kind' has made my doctoral journey fruitful and enjoyable and will always benefit me in my future career.

I would also like to extend my sincere gratitude to the other thesis committee members, Jiawei Han, Ross Maciejewski, and Han Zhao, for their valuable insights, constructive feedback, and time devoted to reviewing and evaluating my thesis. It was the seminal book written by Jiawei that introduced me the field of data mining. I am fortunate to learn from him in both CS 512: Data Mining Principles and my thesis proposal and defense. I am grateful to Ross for his meticulous assistance and mentorship in research and paper writing. The quality and clarity of my research publications are enhanced a lot from his valuable feedback and suggestions. Furthermore, I would like to thank Han for his sharp insights on research. His inspirational research contributions have served as a constant source of motivation and innovation throughout my doctoral journey.

My Ph.D. study has been full of joy and happiness as a member of IDEA lab and iSAIL lab (formerly the DATA lab and STAR lab at Arizona State University). These are not possible without the following wonderful lab members: Jingrui He, Xing Su, Liangyue Li, Chen Chen, Si Zhang, Boxin Du, Rongyu Lin, Scott Freitas, Haichao Yu, Qinghai Zhou, Zhe Xu, Lihui Liu, Baoyu Jing, Yuchen Yan, Shengyu Feng, Yuheng Zhang, Derek Wang, Yian Wang, Zhichen Zeng, Ishika Agarwal, Eunice Chan, Xinyu He, Blaine Hill, Zhining Liu, Ruizhong Qiu, Hyunsik Yoo, Alex Zheng, David Zhou, Ruike Zhu, Xiao Lin, Yaojing Wang, Yu Wang, Ziye Zhu, Hao Wang, Haoran Li, Shweta Jain, Tianwen Chen, Yunyong Ko, Dawei Zhou, Yao Zhou, Arun Reddy Nelakurthi, Xu Liu, Jun Wu, Lecheng Zheng, Xue Hu, Dongqi Fu, Pei Yang, Yikun Ban, Yunzhe Qi, Haonan Wang, Ziwei Wu, Wenxuan Bao, Tianxin Wei, Xinrui He, Isaac Joy, and Zihao Li. I would also like to thank all my friends who has supported my on this incredible journal, both inside and outside of Arizona State University and University of Illinois at Urbana-Champaign.

I had two great experiences for internship in Meta/Facebook AI Applied Research. I would like to extend my sincere appreciation to my mentor, Yan Zhu, and all collaborators: Shujian Bu, Ren Chen, Adam Feldman, Atanu Ghosh, Tiangao Gou, Zhaohui Guo, Yilei He, Yiming Liao, Dionysios Logothetis, Jin Long, Chenglin Lu, Mahi Luthra, Andrey Malevich, Philip Pronin, Yuanyuan Shen, Anders Skog, Lin Su, Fiona Tang, Qifei Wang, Yinglong Xia, Fanny Yang, Pei Yin, and Ranqi Zhu. Their relentless help and generous support helped me onboard in the new industrial work environment smoothly, become familiar with the company code base, and accomplish industrial projects with real-world impacts to billion of users.

In addition to my thesis committee members, labmates, as well as mentors and collaborators during internship, I had a wonderful group of collaborators: Nan Cao, Weilin Cong, Yushun Dong, Wei Fan, Jundong Li, Jiebo Luo, Mehrdad Mahdavi, Fei Wang, Meijia Wang, Hao Wu, Xintao Wu, Tiankai Xie, Baichuan Yuan, and Xin Zhou. Their insights and feedback have helped a lot with my research.

Finally, this thesis is dedicated to my most amazing parents: Yuehu Kang and Fengqin Yin. Their unconditional love and support are the ultimate source of all my motivation, confidence, and strength. I am also immensely grateful to my girlfriend, Ming Hu, for always standing by me and cheering me up, especially during the COVID-19 pandemic.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Graph mining plays a pivotal role in many domains, ranging from information retrieval, social network analysis, to transportation, and computational chemistry. Decades of research in this area have developed a wealth of theories, algorithms, and open-source systems for a variety of mining tasks. Notably, PageRank [1] and HITS [2] are two commonly used ranking algorithms to measure node importance in a graph; and spectral clustering [3] is a well-known graph clustering technique for community detection. In addition, recent advances in machine learning on graphs have produced various node embedding techniques [4, 5, 6] and graph neural networks [7, 8] to learn expressive representation of nodes and graphs. These graph mining models have been widely deployed in various real-world applications, often delivering superior empirical performance in answering *what* and *who* questions. For example, *what* are the most important web pages in the web? *Who* should the algorithm group in the same online community? *What* is the best route from the current location to the desired destination? *What* molecular structure is likely to be toxic?

In recent years, multiple jurisdictions raise the awareness of ensuring fairness in the use of graph mining algorithms, which is often violated as shown in existing studies [9, 10]. One example is the suicide prevention using graph covering algorithm as shown in Figure 1.1, where only 2 gatekeepers can be selected to watch out potential suicide attempts among their peers (i.e., cover their neighbors in the graph). Though the current selection of gatekeepers would maximize the number of covered nodes, it may fail to take care of certain individual from the minority group that exhibits warning sign of suicide (i.e., the upper left green node). Unfairness on graphs could also cause worrying outcomes in college recruitment, where information about the opportunities of college recruitment diffuses disproportionately to students from high-income families, causing the students from low-income families unaware of the opportunities of attending selective colleges [9].

1.2 ESSENTIAL PROPERTIES OF FAIR GRAPH MINING

The alarming consequences of unfairness on graphs necessitate us to answer a fundamental question: *how* can we make the graph mining process and its results fair? To achieve fair graph mining, it is crucial to propose a paradigm shift for graph mining, from answering *what* and *who* to understanding *how* and *why*. And it is essential to call for four essential

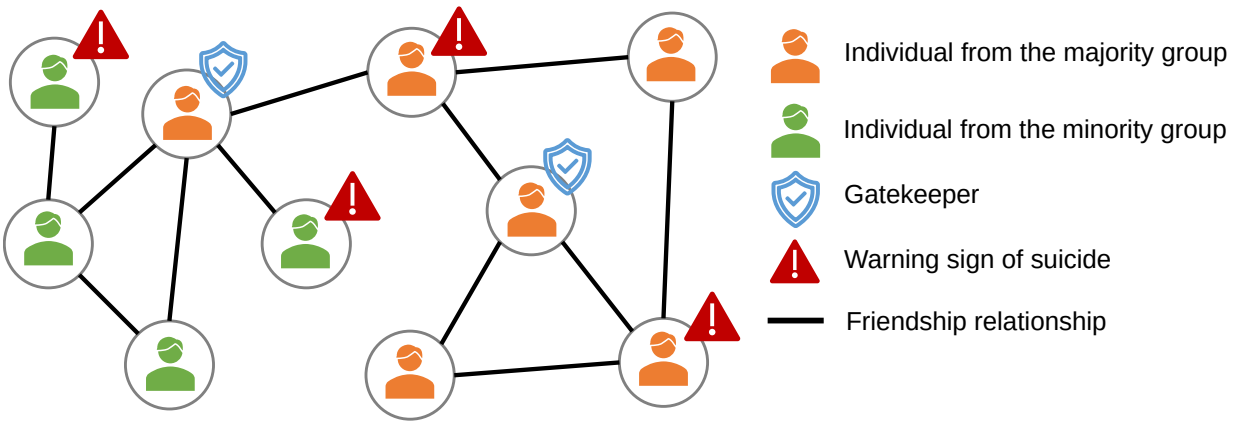


Figure 1.1: An illustration of unfair graph covering for suicide prevention. We use the color-based imaginary race groups to avoid ny potential offense.

properties to build an algorithmic foundation of fair graph mining, including *utility*, *fairness*, *robustness*, and *transparency*.

(1) *Utility*. The basic goal of most, if not all, of the graph mining models is to achieve a strong empirical performance in the corresponding graph mining tasks, in order to be deployed in the real world. Thus, utility helps measure *how useful* a graph mining model is in a graph mining task.

(2) *Fairness*. It is one of the most fundamental goals, which can be told from the name 'fair graph mining' itself. Unfair graph mining results are often caused by the favoritism in the predictions over a demographic group or individual. Thus, a key to guarantee fairness is to study *how* to accommodate differences over diverse demographic groups or individuals so as to achieve similar quality of service.

(3) *Robustness*. Real-world graphs are not perfect, possibly containing malicious nodes or edges. In order to deploy fair graph mining models in real-world scenarios, it is essential to understand *how resilient* they are with respect to noise or adversarial activity.

(4) *Transparency*. The end users of graph mining models are often non-experts in graph mining. Thus, they may not understand *why* the model makes certain prediction or *how normal* the model functions. Enhancing transparency not only could render the crucial explainability and audit model behaviors, but also may help determine to what extent sensitive information is used in decision making.

1.3 RESEARCH CHALLENGES

Having the four desired properties in mind, the key challenges of fair graph mining naturally arise from the tensions among these competing properties.

1.3.1 The Auditing Challenge

The *auditing* challenge lies in the tension between utility and transparency. The superior performance of graph mining is often achieved by complex models (e.g., graph neural networks) that are incomprehensible even by graph mining experts, let alone the majority of non-expert end users. Without an intuitive way to understand how graph mining models function and why they output a specific outcomes, the end users can hardly trust whether predictions made by graph mining models is accurate and/or fair in high-stake applications. In the meanwhile, though heuristic graph mining models (e.g., solely based on degree centrality) function in a human-understandable way and can be easily explained, they often yield a sub-optimal performance in the downstream tasks. Thus, the auditing challenge calls for a trade-off between utility and transparency and asks the following key research question: *how* do the mining results relate to the input graph? By providing a high-quality audit toward the graph mining results, users can identify potentially influential edges, nodes, or subgraphs that will drastically affect graph mining results if perturbed, thus explaining model behaviors or leading to unfair mining results.

1.3.2 The Debiasing Challenge

The *debiasing* challenge lies in the tension between utility and fairness. Mitigating the bias in graph mining is the key component that enables fair graph mining. A general approach to ensure fairness is to impose a fairness constraint in the optimization problem. It might come at the cost of sacrificing the utility of the graph mining task, because the model would not only optimize the task-specific loss function for utility, but also consider the satisfaction of the fairness constraint in the meanwhile. Consequently, it is crucial to balance the trade-off between utility and fairness. However, the non-IID[1] nature of graph data might bring two caveats compared with existing theories and methods in fair machine learning on IID data. First, it may invalidate the basic assumption behind many existing studies in fair machine learning. For example, neighborhood propagation and aggregation, which are fundamental mining operations in graph ranking and graph neural networks, could amplify

[1]IID refers to 'independent and identically distributed'.

the bias [11, 12, 13]. Thus, debiasing graph mining models may not only ask for mitigating discriminatory information in the input node features, but also require the refinement on the topological structures to avoid bias propagation and amplification through edges. Second, it could introduce new opportunities for measuring and mitigating bias based on the connections among nodes. For example, in addition to defining fairness on nodes (analogous to data points in classic machine learning under IID assumption), the dyadic nature of edges in the graph may bring new fairness definitions that consider the synergy among nodes, such as degree fairness [14] and dyadic fairness [15]. Therefore, to solve the debiasing challenge, it is essential to (1) re-examine the implication(s) of non-IID nature on measuring and mitigating bias and (2) understand how the implication(s) would affect the trade-off between utility and fairness.

1.3.3 The Safeguarding Challenge

The *safeguarding* challenge lies in the tension among all desired properties, including utility, fairness, transparency, and robustness. The need for robustness and transparency is critical to the deployment of debiasing techniques developed in the debiasing challenge in the real world. Regarding the robustness, it is commonplace that the real-world data is often noisy and could contain malicious users. And a deployable debiasing technique should be resilient to such noise or adversarial activity. It is worth pointing out that existing studies on the robustness of graph mining almost exclusively study how resilient the graph mining models are in terms of utility. However, to deploy the techniques we developed in the debiasing challenge in real-world scenarios, it is also important to study whether noise or adversarial activity would make an initially fair graph mining models to be unfair. While for transparency, the complex debiasing techniques might share the same black-box nature as many graph mining models. To enhance the trust of end users toward the debiasing techniques in deployment, transparency should also be considered so as to understand how and why a model makes a fair/biased prediction, which is largely overlooked by previous efforts in the area.

1.4 OVERVIEW OF RESEARCH TASKS

As shown in Figure 1.2, the aforementioned challenges form three key pillars to build an algorithmic foundation of fair graph mining. Thus, the goal of my Ph.D. research is to design novel algorithms and frameworks underpinning all three pillars. Consequently, a paradigm

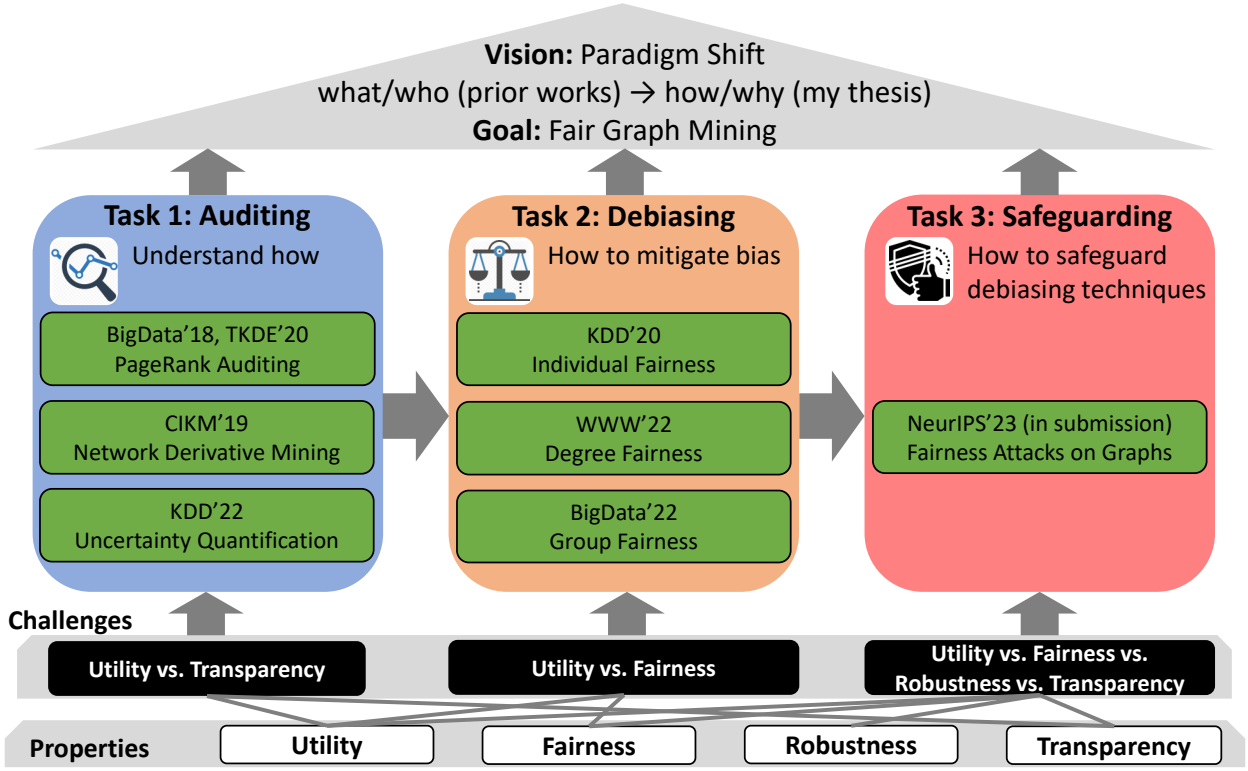


Figure 1.2: An overview of the algorithmic foundation of fair graph mining. Three key pillars refer to three challenges in achieving the paradigm shift from answering who and what to understanding how and why.

shift from answering *what* and *who* in graph mining to understanding *how* and *why* in fair graph mining could be achieved.

Task 1 – Auditing. As a prelude in achieving fair graph mining, it focuses on the transparency of graph mining by demonstrating how the mining results of a given graph mining model relate to the input graph. We start by developing a family of algorithms AURORA to audit PageRank algorithm. AURORA explores the diminishing returns property of the graph element influence to identify a set of influential graph elements with theoretical guarantees on the optimality. Going beyond PageRank, we further generalize the key insights developed in AURORA and design a general algorithmic framework N2N to audit a variety of graph mining models, including HITS ranking, spectral clustering, and matrix completion. Furthermore, observing that the real-world graphs are often uncertain, we introduce the first frequentist-based uncertainty quantification technique on graph convolutional network named JURYGCN [16] to audit how uncertain the graph mining results are, which is useful to improve active learning on node calssification and semi-supervised node classification with the lowest memory usage than the competitors and a 130× speed-up than the naive model

re-training based variant.

Task 2 – Debiasing. As the key component in enabling fair graph mining, it explores how to ensure various fairness definitions on graph mining. In this task, we offer the first principled study of individual fairness on graph mining (INFORM [17]), including how to measure individual bias of a graph mining result, how to mitigate individual bias, and what the cost is if individual fairness is satisfied. Moreover, we study degree fairness on graph convolutional networks, which exhibits performance disparity with respect to node degrees, resulting in worse predictive accuracy for low-degree nodes. We theoretically prove that the degree unfairness is rooted in the gradient computation with respect to weight matrices. Based on the theoretical insights, we design a family of debiasing algorithms RAWLSGCN [18] that can mitigate up to 89% of the degree unfairness with no additional memory consumption and up to 8× speed-up in time. Other than that, considering the co-existence of multiple sensitive attribute in real-world applications, we develop a model-agnostic information-theoretic debiasing framework INFOFAIR [19] that directly optimizes the fairness definition itself to ensure group fairness with respect to multiple sensitive attribute.

Task 3 – Safeguarding. As a postlude of fair graph mining, it aims to make fair graph mining deployable in the real world by balancing the tension among all aforementioned desired properties. In this task, we study fairness attacks on graph learning models, which amplifies the bias with respect to a fairness definition and maintains the utility in the downstream task. We design a meta learning-based framework FATE to perform poisoning attacks on the input graph for fairness attacks, which is capable of attacking any fairness definition for any graph learning model. We instantiate FATE to attack both group fairness and individual fairness on graph neural networks with and without fairness consideration. Experimental results demonstrate that FATE can amplify the bias with respect to both group fairness and individual fairness while maintaining the classification accuracy of node classification.

1.5 ORGANIZATION

The rest of the thesis is organized as follows. In Chapter 2, we will review existing literature in relation to the four properties in the context of graph mining. In Chapter 3, we will introduce our works that address the auditing task via PageRank auditing, network derivative mining, and uncertainty quantification. In Chapter 4, we will present our works on ensuring various definitions of algorithmic fairness. In Chapter 5, we will introduce how we deal with the safeguarding challenge via fairness attacks on graph mining. Finally, in Chapter 6, we conclude the thesis and discuss the future research directions.

CHAPTER 2: LITERATURE REVIEW

2.1 GRAPH MINING – UTILITY

Graph mining aims at finding interesting patterns from the underlying networked data [1, 2, 3, 4, 5, 7, 8, 20]. Therefore, utility, which targets for strong empirical performance in graph mining tasks, is a fundamental property across all research tasks in the thesis. To date, graph mining models have been applied in a variety of graph mining tasks, including ranking, clustering, embedding, and many more.

For graph ranking, it aims to measure the node importance or node proximity with respect to a query node. PageRank [1] and HITS [2] are two most widely used algorithms to measure the importance of nodes. To be specific, PageRank [1] repeat the following procedure until convergence to calculate the node importance: it randomly walks to a neighboring node with some probability, or re-starts the random walk process at any node in the graph uniformly otherwise. HITS [2] measures the node importance with a hub score and an authority score, which can be calculated with rank-1 singular value decomposition. Ever since, both graph ranking algorithms inspired the development of many variants. Typical examples include personalized PageRank [21], random walk with restart [22], and subspace HITS [23].

Regarding graph clustering, one of the most representative clustering method is the spectral clustering [3]. To cluster nodes into k clusters, it finds the eigenvectors associated with k smallest eigenvalues of a (normalized) graph Laplacian, and then performs k-means clustering [24] on the eigenvectors. Other well-known clustering methods on graphs include hierarchical clustering [25, 26] and online evolutionary clustering [27].

Recent advances in graph machine learning have produced a variety of embedding techniques that encode the structural and/or attributive information of the input graph into a low-dimensional manifold. Matrix factorization-based approaches learn the node representations by factorizing the input graph into low-rank matrices [20, 28, 29, 30, 31, 32]. For example, LLE [28], IsoMap [29], and Laplacian Eigenmap [30] are classic embedding methods that factorizes the graph Laplacian. Other techniques like NMF [31] and collaborative filtering [20, 32] factorize the input adjacency matrix. More advanced graph embedding techniques developed in recent years include random walk-based approaches and graph neural networks. Representative examples of random walk-based embedding approaches include DeepWalk [4], LINE [5], and node2vec [6]. NetMF [33] further generalizes the random walk-based approaches and reveals their intrinsic relationships to matrix factorization. Another important line of research in graph embedding is graph neural networks. Existing works can be categorized into

spectral-based methods and spatial-based methods. For spectral-based methods, they often construct the frequency filtering using the eigenfunctions of the graph Laplacian. Representative examples include Graph Convolutional Neural Network (GCNN) [34], ChebyNet [35], and Graph Convolutional Network (GCN) [7]. Spatial-based methods often aggregate information within the node neighborhood, e.g., GraphSAGE [36], Diffusion-Convolution Neural Network (DCNN) [37], and Graph Attention Network (GAT) [8].

For more related works, we refer to recent surveys [38, 39]. It should be noted that these existing works almost exclusively optimize for utility while overlooking other properties (i.e., fairness, transparency and robustness).

2.2 GRAPH MINING – FAIRNESS

Fairness is the fundamental property in achieving fair graph mining. It aims to accommodate the discriminatory differences among multiple sensitive groups. Following this overarching principle, various types of fairness can be defined depending on the granularity of the group, such as group fairness [40], individual fairness [41], and counterfactual fairness [42]. Fairness in classic machine learning under IID setting has been extensively studied for years [9, 43, 44]. However, classic fair machine learning techniques may not be directly applied because of the non-IID nature of graph data. And a re-examination of the implications of non-IID nature on measuring and mitigating the bias is required.

Many debiasing techniques on graphs have been developed so far. Group fairness is one of the most extensively studied fairness definitions. It aims to ensure disparity in statistical measures (e.g., acceptance rate, true positive/negative rate) for sensitive groups. To date, group fairness have been studied in PageRank [11, 45], spectral clustering [46], and node embedding [12, 13, 47, 48, 49, 50, 51, 52]. For example, fair PageRank guarantees a certain proportion of total PageRank mass is assigned to nodes in a specific demographic group [11, 45] by modifying the stochastic propagation matrix during the random walk process; fair spectral clustering considers the member distribution from each demographic group in a cluster [46] and encodes the fairness consideration as a linear constraint during model optimization. Moreover, many recent works study group fairness in node embedding. Typical solutions to learn fair node embedding includes adversarial learning [12, 47, 53], fairness-aware neighborhood propagation [48, 54] and aggregation [13, 55], embedding normalization [51], and topological structure modification [15, 50, 56].

Few efforts have been made on enforcing individual fairness and counterfactual fairness as well. For individual fairness, inspired by InFoRM [17], IF-D [57] ensures individual fairness on dynamic graph by optimizing a time-encoded individual fairness definition. GFairHint [58]

concatenates the node embedding extracted from the input graph and the fairness hint embedding learned via link prediction on the pairwise similarity graph. REDRESS [59] leverages learning-to-rank technique to avoid the distance comparison in the [17]. Recently, FairRankVis [60] designs a visual analytics system to discover the interaction between individual fairness and group fairness on graphs. Counterfactual fairness is another important fairness definition, whose goal is to remove the causal connection between the sensitive attribute and the mining results. NIFTY [61] applies contrastive learning to learn node embeddings that satisfy counterfactual fairness. GEAR [62] generates counterfactual graphs with a graph auto-encoder and utilizes the Siamese network [63] for fair representation learning.

Other than the aforementioned fairness definitions, few other fairness definitions have been studied as well. For example, degree fairness ensures comparable performance in the downstream tasks for nodes with different degrees [14, 64, 65, 66, 67]. Dyadic fairness studies edge-level fairness by ensuring comparable link prediction accuracy for edges of different types [15, 68]. Max-min fairness incorporates the Rawlsian difference principle [69] to optimize the performance of graph mining models on the worst-off group of nodes [10, 70]. For more related literature about fairness in graph mining, we refer to recent surveys [71, 72] and tutorials [73, 74].

2.3 GRAPH MINING – ROBUSTNESS

Robustness characterizes the stability of graph mining models against random or adversarial perturbations on graphs, both of which are commonplace in the real world. Thus, in order to enable the deployment of fair graph mining in the real world, robustness is the key properties in the safeguarding task. It is worth pointing out that the safeguarding challenge in the thesis will also study the robustness in terms of how fair a model could be (i.e., how to attack the mining model to make it biased), which is different from existing works that study the robustness in terms of the model utility.

Regarding the robustness aspect of graph mining, we briefly review literature related to both attacking and defense. Attacking aims to apply random or adversarial perturbations on the input graph to degrade the utility of graph mining models. Due to the notorious vulnerability of deep learning models [75], the major focus in this line of research is to attack node embedding algorithms, with few efforts made on the vulnerability of graph ranking algorithms [76, 77]. Representative approaches to attack node embedding algorithms include projected gradient descent [78, 79], bi-level optization [80, 81], reinforcement learning [82], edge flipping [83], and edge rewiring [84]. Other than attacking the utility of graph mining

models, FA-GNN [85] is the only method that attacks the fairness aspect of graph neural networks via edge injection to our best knowledge.

Defending against attacks/perturbations on the graph mining models is another important line of research. Spectral analysis is commonly applied to analyze the stability of graph ranking [86, 87, 88] and the robustness of graph connectivity [89, 90, 91, 92]. Besides, the robustness of node embedding techniques has drawn much research attention in recent years [93, 94, 95, 96, 97, 98, 99, 100]. More specifically, certified adversarial robustness of graph neural networks and label/feature propagation is provided in [94]. Adversarial training is also proven to be useful in robustifying graph neural networks [93]. Moreover, RGCN [95] introduces Gaussian hidden representations to absorb the effect of adversarial changes; GNN-Jaccard [96] disconnects dissimilar nodes; Pro-GNN [97] regularizes graph sparsity and feature smoothness; PA-GNN [98] enhances the robustness of graph neural networks via meta learning and transfer learning; GNN-SVD [99] exploits the low-rank singular components of graph topology; and GASOLINE [100] learns the optimal graph structure with maximal utility via meta learning. Recent surveys [101, 102] present comprehensive reviews for graph robustness measure and adversarial robustness of graph mining models, respectively.

2.4 GRAPH MINING – TRANSPARENCY

Transparency aims to make graph mining models accountable and explainable through auditing and generating explanations for the mining results. To develop fair graph mining models that can be trusted by non-expert end users, transparency should also be a key properties to consider in the safeguarding task. For auditing, NEAR [103, 104] audits the skip-gram based node embedding techniques and linear graph neural networks by quantifying the influence of edges. A visual analytics system [105] is designed to audit the sensitivity of graph ranking algorithms. The influence of edge and node removal in linear graph neural network is also derived in [106]. It is also important to enhance the interpretability and explainability of graph mining algorithms. Most efforts in this line have been made to node embedding algorithms [107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119]. For perturbation-based methods, GBP [108] generates the saliency map with gradient information; ExplaiNE [120] explains node embedding techiniques via the influence function of edges; GNNExplainer [107] and PGExplainer [112] both find the subgraph with highest mutual information to the model predictions; GraphMask [113] masks unnecessary edges that would not affect model predictions; SubgraphX [114] finds the subgraphs that give the highest Shapley value [121]; and CF-GNNExplainer [117] generates counterfactual explanations via graph sparsification. For generation-based explainability techniques, XGNN [109] generates

the graph pattern that maximizes the prediction probability for a specific class. For surrogate model-based techniques, GraphLIME [116] introduces LIME [122] to graph neural networks; PGM-Explainer [111] provides explanations using an interpretable Bayesian network. More comprehensive review of related literature can be found in [123, 124]. In recent years, few works focus on the intersection of fairness and explainability on graphs. Specifically, REFEREE [125] learns a mask matrix for the adjacency matrix to find out the edges that are most accountable for group fairness; BIND [126] finds the training node that contributes most to group fairness via influence function; and CFA [127] guarantees similar explanation quality across different groups.

CHAPTER 3: AUDITING GRAPH MINING

Auditing graph mining is a prelude in building an algorithmic foundation of fair graph mining. By studying the tensions between utility and transparency, it aims to understand how normal/confident a graph mining model functions and why a graph mining model outputs the specific graph mining results. In this chapter, we introduce our works on addressing the auditing challenge, including (1) a family of algorithms to audit PageRank efficiently with theoretical guarantees, (2) a generic algorithmic framework to audit a variety of graph mining models, and (3) a post-hoc uncertainty quantification method to understand the confidence of graph convolutional networks.

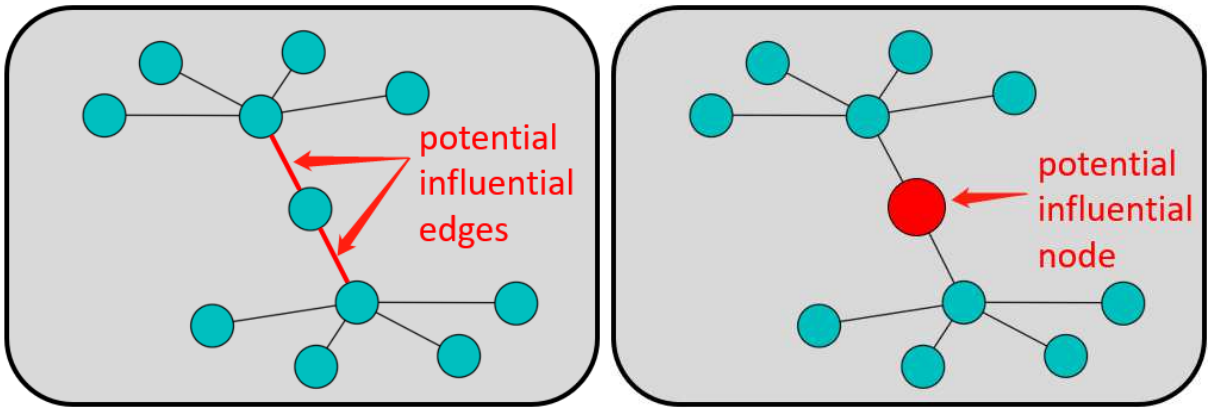
3.1 AUDITING PAGERANK ON GRAPHS

Ranking on graph data is a way to measure node importance and plays a fundamental role in many real-world applications, ranging from information retrieval [1], recommender systems [128], social networks [129], sports team management [130] to biology [131], and neuroscience [132]. Among others, PageRank [1], together with many of its random walk based variants, is one of the most well-known and widely used ones. Its mathematical elegance lies in that it only requires the topological structure and the associated edge weights as the input. Such a generality makes it applicable to networks[2] from many different application domains.

PageRank has a strong ability answering questions like *what* is the most important page in the World Wide Web, and *who* is the most influential person in a collaboration network. Despite its superior performance on graph ranking, PageRank lacks intuitive ways to give answers to questions like *why* the top-k returned webpages are the most important ones, and *why* the actor `John` ranks higher than the actor `Smith` in terms of their relevance with respect to a particular movie. *How* the ranking results are derived from the underlying graph structure has largely remained opaque to the end users, who are often not experts in data mining and mathematics.

To address this challenge, we aim to explain PageRank by finding the influential graph elements (e.g., edges, nodes, and subgraphs), which we formulate as the *PageRank auditing* problem. The key idea is to quantitatively understand how the ranking results would change if we perturb a specific graph element. To be specific, we measure the influence of each graph element by the rate of change in a certain loss function (e.g., L_p norm, etc.) defined over

[2]We use 'graph' and 'network' interchangeably.



(a) Potentially influential edges. (b) Potentially influential node.

Figure 3.1: Examples of potentially influential edges and node.

the ranking vector. We believe that auditing graph ranking can benefit many real-world applications. First, it can render the crucial explainability of such ranking algorithms, by identifying valuable information of influential graph elements. Thus, it can help answer questions like why a given node ranks on the top of the ranking list, and which link leads to the ranking vector in a certain way. Furthermore, users can use the auditing results to optimize the network topology. In addition, it may help identify the vulnerabilities in the network (e.g., links between two clusters, cutpoints of clusters as shown in Figure 3.1a and Figure 3.1b). With the auditing results, users may find several links or nodes that have the greatest influence on the ranking results. It can help users identify if there exist suspicious individuals that manipulate the ranking results by linking heavily with unrelated, off-query topics. Finally, with the knowledge of graph ranking auditing, we may design a more robust ranking algorithm that is hard to be manipulated by users with strategies like linking heavily [88].

The main contributions of this work are summarized as follows.

- *Problem definition.* We formally define the PageRank auditing problem and formulate it as an optimization problem, whose key idea is to measure the influence of different graph elements as the rate of change in the PageRank results.

- *Algorithms and analysis.* We develop fast approximation algorithms to solve the PageRank auditing problem. The algorithms achieve a $(1 - 1/e)$ approximation ratio with a linear complexity.

- *Empirical evaluations.* We perform extensive experiments on diverse, real-world datasets. The experimental results demonstrate that AURORA (a) provides reasonable and

Table 3.1: Table of symbols for AURORA.

Symbol	Definition
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$	the input network
(i, j)	edge from node i to node j
$\mathbf{A}$	adjacency matrix of the input network
$\mathbf{A}[i, j]$	the element at i-th row and j-th column
$\mathbf{A}[i, :]$	i-th row of matrix $\mathbf{A}$
$\mathbf{A}[:, j]$	j-th column of matrix $\mathbf{A}$
$\mathbf{A}^T$	transpose of the matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of the matrix $\mathbf{A}$
$\mathbf{e}$	the teleportation vector in PageRank
$\mathbf{r}$	PageRank of the input network
$\mathbf{r}[i]$	ranking score of i
$\mathrm{Tr}(\mathbf{A})$	Trace of the matrix $\mathbf{A}$
$f(\mathbf{r})$	a loss function over ranking vector $\mathbf{r}$
n	number of nodes in the input network
m	number of edges in the input network
c	damping factor in PageRank

intuitive information to find influential graph elements and (b) scales linearly with respect to the graph size.

3.1.1 Problem Definition

In this part, we first present a table of symbols (Table 3.1) that contains the main notations used throughout this work and then review PageRank for ranking nodes on graphs. Finally, we give a formal definition of the PageRank auditing problem.

We use bold upper-case letters for matrices (e.g., $\mathbf{A}$), bold lower-case letters for vectors (e.g., $\mathbf{a}$), calligraphic letters for sets (e.g., $\mathcal{S}$), and italic lower-case letters for scalars (e.g., c). For matrix indexing conventions, we use the rules similar to Numpy in Python that are shown as follows. We use $\mathbf{A}[i, j]$ to denote the entry of matrix $\mathbf{A}$ at i-th row and j-th column, $\mathbf{A}[i, :]$ to denote the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}[:, j]$ to denote the j-th column of matrix $\mathbf{A}$. We use the superscrit ^{T} to denote the transpose of matrix (i.e., $\mathbf{A}^T$ is the transpose of matrix $\mathbf{A}$).

Given a graph $\mathcal{G}$ with n nodes and m edges, PageRank essentially solves the following linear system,

$$\mathbf{r} = c\mathbf{A}\mathbf{r} + (1 - c)\,\mathbf{e} \tag{3.1}$$

where $\mathbf{e}$ is the teleportation vector with length n and $\mathbf{A}$ is the normalized adjacency matrix of

the input graph. In PageRank [1], $\mathbf{e}$ is chosen as the uniform distribution $\frac{1}{n}\mathbf{1}$; in personalized PageRank [21], $\mathbf{e}$ is a biased vector which reflects user preference (i.e., 'personalization') [21]; and in random walk with restart [22], all the probabilities are concentrated on a single node. A default choice for the normalized adjacency matrix $\mathbf{A}$ is the column-normalized matrix (i.e., the stochastic matrix). A popular alternative choice is the normalized graph Laplacian matrix. In fact, as long as the largest eigenvalue of $\mathbf{A}$ is less than $1/c$, a fix-point solution of the above linear system will converge to $\mathbf{r} = (1-c)\left(\mathbf{I} - c\mathbf{A}\right)^{-1}\mathbf{e}$ [133]. In this work, we use this general form for the adjacency matrix $\mathbf{A}$. For the ease of description, we also define $\mathbf{Q} = \left(\mathbf{I} - c\mathbf{A}\right)^{-1}$, and $\mathbf{r} = \mathrm{pg}\left(\mathbf{A}, \mathbf{e}, c\right)$ as the resulting PageRank vector with $\mathbf{A}$, $\mathbf{e}$, and c as the corresponding inputs.

Regarding explainable learning and mining techniques, Pang and Liang [134] propose a novel notation of influence functions to quantify the impact of each training example on the underlying learning system (e.g., a classifier). The key idea is to trace the model's prediction back to its training data, where the model parameters were derived. In this way, it learns how a perturbation of a single training data will affect the resulting model parameters and identifies the training examples that are most responsible for model predictions.

Our developed method to explain the PageRank results is built upon the principle outlined in [134]. To be specific, we aim to find a set of graph elements (e.g., edges, nodes, a subgraph) such that, when we perturb/remove them, the ranking vector will have the greatest change. Formally, we define the PageRank auditing problem as follows:

Problem 3.1. The PageRank auditing problem.

Given: (1) a graph with adjacency matrix $\mathbf{A}$, PageRank vector $\mathbf{r}$, (2) a loss function f over its PageRank vector, (3) a user-specific element type (e.g. edges vs. nodes vs. subgraphs), and (4) an integer budget k.
Find: a set of k influential graph elements that has the largest impact on the loss function over its PageRank vector $f\left(\mathbf{r}\right)$.

3.1.2 Auditing PageRank with Aurora

In this part, we present a family of algorithms (Aurora), to solve PageRank auditing problem (Problem 3.1), together with some analysis in terms of effectiveness and efficiency.
Formulation. The intuition behind Aurora is to find a set of key graph elements (e.g., edges, nodes, a subgraph) whose perturbation/removal from the graph would affect the PageRank results most. To be specific, let $\mathbf{r} = \mathrm{pg}(\mathbf{A}, \mathbf{e}, c)$ be the PageRank vector of the input graph $\mathbf{A}$, and $\mathbf{r}_{\mathcal{S}} = \mathrm{pg}(\mathbf{A}_{\mathcal{S}}, \mathbf{e}, c)$ be the new PageRank vector after removing the graph

elements in set $\mathcal{S}$. We formulate the PageRank auditing problem as the following optimization problem.

$$\max_{\mathcal{S}} \Delta f = \left(f\left(\mathbf{r}\right) - f\left(\mathbf{r}_{\mathcal{S}}\right)\right)^2 \quad \text{s.t. } |\mathcal{S}| = k \tag{3.2}$$

In order to solve the above optimization problem, we need to answer two key questions including (Q1) how to quantitatively measure the influence of an individual graph element with respect to the objective function and (Q2) how to collectively find a set of k graph elements with the maximal influence. We first present our solution for Q1 and then develop three different algorithms for Q2, depending on the specific type of graph elements (i.e., edges vs. nodes vs. subgraph).

Measuring graph element influence. To measure how $f\left(\mathbf{r}\right)$ will change if we perturb/remove a specific graph element, we define its influence as the rate of change in $f\left(\mathbf{r}\right)$.

Definition 3.1. (Graph element influence). The influence of an edge (i, j) is defined as the derivative of the loss function $f\left(\mathbf{r}\right)$ with respect to the edge, i.e., $\mathbb{I}\left[i, j\right] = \frac{df(\mathbf{r})}{d\mathbf{A}[i,j]}$. The influence of a node is defined as the aggregation of all inbound and outbound edges that connect to the node., i.e., $\mathbb{I}\left[i\right] = \sum_{j=1, j\neq i}^{n} \left[\mathbb{I}\left[i, j\right] + \mathbb{I}\left[j, i\right]\right] + \sum_{j=1, j=i}^{n} \mathbb{I}\left[i, j\right]$. And the influence of a subgraph is defined as the aggregation of all edges in the subgraph S, $\mathbb{I}\left(S\right) = \sum_{i,j\in S} \mathbb{I}\left[i, j\right]$.

We can see that the influence for both nodes and a subgraph can be naturally computed based on the edge influence. Therefore, we will focus on how to measure the edge influence. By the property of the derivative of matrices, we first re-write the influence $\frac{df(\mathbf{r})}{d\mathbf{A}[i,j]}$ as

$$\frac{df\left(\mathbf{r}\right)}{d\mathbf{A}} = \begin{cases} \frac{\partial f(\mathbf{r})}{\partial \mathbf{A}} + \left(\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}\right)^T - \operatorname{diag}\left(\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}\right), & \text{if undirected} \\ \frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}, & \text{if directed} \end{cases} \tag{3.3}$$

Directly calculating $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}[i,j]}$ is hard, and we resort to the chain rule:

$$\frac{\partial f\left(\mathbf{r}\right)}{\partial \mathbf{A}\left[i, j\right]} = \left(\frac{\partial f\left(\mathbf{r}\right)}{\partial \mathbf{r}}\right)^T \frac{\partial \mathbf{r}}{\partial \mathbf{A}\left[i, j\right]} \tag{3.4}$$

Next, we present the details on how to solve each partial derivative in Equation (3.4) one by one.

1 – Computing $\frac{\partial f(\mathbf{r})}{\partial \mathbf{r}}$. Here we discuss the choices of $f(\cdot)$ function. We choose $f(\cdot)$ to be squared L_2 norm for simplicity. However, it is worth mentioning that AURORA is applicable to a variety of other loss functions as well. We list some alternative choices and their corresponding derivatives in Table 3.2. In the table, L_p *norm* is the most commonly-used

Table 3.2: Choices of $f(\cdot)$ functions and their derivatives

Description	Function	Derivatives
L_p norm	$f(\mathbf{r}) = \|\mathbf{r}\|_p$	$\frac{\partial f}{\partial \mathbf{r}} = \frac{\mathbf{r} \circ \|\mathbf{r}\|^{p-2}}{\|\mathbf{r}\|_p^{p-1}}$
Soft maximum	$f(\mathbf{r}) = \log\left(\sum_{i=1}^{n} \exp(\mathbf{r}[i])\right)$	$\frac{\partial f}{\partial \mathbf{r}} = \left[\frac{\exp(\mathbf{r}[i])}{\sum_{i=1}^{n} \exp(\mathbf{r}[i])}\right]$
Energy norm	$f(\mathbf{r}) = \mathbf{r}^T\mathbf{M}\mathbf{r}$	$\frac{\partial f}{\partial \mathbf{r}} = (\mathbf{M} + \mathbf{M}^T)\mathbf{r}$

($\mathbf{M}$ in Energy Norm is a Hermitian positive definite matrix.)

vector norm that measures the overall size of a vector. Some commonly-used L_p norm includes L_1 norm and L_2 norm (also known as the Euclidean norm); *soft maximum* is used to approximate the maximum value of elements in a vector; *energy norm* is a measurement often used in system and control theory to measure the internal energy of a vector.

2 – Computing $\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]}$. Recall that $\mathbf{A}$ is the adjacency matrix of the input graph, and the PageRank solution can be written as solving the linear system in Equation (3.1). Then we have

$$\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]} = c\frac{\partial \mathbf{A}}{\partial \mathbf{A}[i,j]}\mathbf{r} + c\mathbf{A}\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]} \tag{3.5}$$

Moving the second term in Equation (3.5) to the left, we have that

$$(\mathbf{I} - c\mathbf{A})\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]} = c\frac{\partial \mathbf{A}}{\partial \mathbf{A}[i,j]}\mathbf{r} \tag{3.6}$$

$$\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]} = c(\mathbf{I} - c\mathbf{A})^{-1}\frac{\partial \mathbf{A}}{\partial \mathbf{A}[i,j]}\mathbf{r} \tag{3.7}$$

By the property of the first order derivative of matrix, we have that

$$\frac{\partial \mathbf{A}}{\partial \mathbf{A}[i,j]} = \begin{cases} \mathbf{S}^{ij} + \mathbf{S}^{ji} - \mathbf{S}^{ij}\mathbf{S}^{ji}, & \text{if undirected} \\ \mathbf{S}^{ij}, & \text{if directed} \end{cases} \tag{3.8}$$

where $\mathbf{S}^{ij}$ is the single-entry matrix with 1 at i-th row and j-th column and 0 elsewhere. Recall that $\mathbf{Q} = (\mathbf{I} - c\mathbf{A})^{-1}$, we can re-write Equation (3.7) as the following equation.

$$\frac{\partial \mathbf{r}}{\partial \mathbf{A}[i,j]} = c\mathbf{r}[j]\,\mathbf{Q}[:,i] \tag{3.9}$$

Combine everything together, we get the closed-form solution for calculating the influence of

an edge (i, j) as follows:

$$\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}[i,j]} = 2c\mathbf{r}[j] \operatorname{Tr}\left(\mathbf{r}^T \mathbf{Q}[:, i]\right) \tag{3.10}$$

Following Equation (3.10), we get the matrix of gradients for all edges as

$$\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}} = 2c(1-c)\mathbf{Q}^T \mathbf{r}\mathbf{e}^T \mathbf{Q}^T \tag{3.11}$$

Two major computational challenges in calculating $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ lie in (1) calculating $\mathbf{Q}$ with $O(n^3)$ time complexity and (2) $O(n^2)$ space complexity to save the matrix of gradients. We address both challenges by exploring the low-rank structure of $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$. From Equation (3.11), we can show that it can be re-written as the following low-rank form

$$\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}} = 2c(1-c)\mathbf{Q}^T \mathbf{r}\mathbf{e}^T \mathbf{Q}^T = 2c(\mathbf{Q}^T \mathbf{r})\mathbf{r}^T \tag{3.12}$$

where $\mathbf{Q}^T\mathbf{r}$ is an $n \times 1$ vector and $\mathbf{r}$ is an $n \times 1$ PageRank vector. Since $\mathbf{r} = (1-c)\mathbf{Q}\mathbf{e}$, we have that $\mathbf{Q}^T\mathbf{r}$ is a personalized PageRank on the reverse of the input graph with $\mathbf{r}$ as teleportation vector with a constant scaling. With this in mind, we do not need to calculate $\mathbf{Q}$ explicitly, or to save the entire matrix directly. Instead, we can use power method to calculate $\mathbf{r}$ and $\mathbf{Q}^T\mathbf{r}$, each with $O(m)$ time, and save these two vectors with $O(n)$ space. To extract the element of $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ at the i-th row and the j^{th} column, we simply calculate the product of the i^{th} element in $\mathbf{Q}^T\mathbf{r}$ and the j-th element in $\mathbf{r}$, and scale it by $2c$, which takes $O(1)$ time.

Auditing by edges: AURORA-E. Due to its combinatorial nature, straightforward methods for solving the optimization problem in Equation (3.2) are not feasible. The key behind AURORA-E is the diminishing returns property of Problem 3.1, which is summarized in Theorem 3.1

Theorem 3.1. (Diminishing returns property of Problem 3.1). For *any* loss function listed in Table 2, and for *any* set of graph elements $\mathcal{S}$, which could be either a set of edges, nodes or subgraphs, in the given graph, its influence measure $\mathbb{I}(\mathcal{S})$ defined in Definition 1 is (a) normalized; (b) monotonically non-decreasing; (c) submodular, where $\mathcal{S} \subseteq \mathcal{E}$.

Proof. We only prove the diminishing returns property in the edge case. The proofs for nodes and subgraphs are similar and thus is omitted due to the space limitation.

Let $\mathbb{I}(\mathcal{S}) = \sum_{[i,j] \in \mathcal{S}} \mathbb{I}[i,j]$. It is trivial that, if there is no edge selected, the influence is 0. Thus it is normalized.

Let $\mathcal{I}, \mathcal{J}, \mathcal{K}$ be three sets and $\mathcal{I} \subseteq \mathcal{J}$. We further define three sets $\mathcal{S}, \mathcal{T}, \mathcal{K}$ as follows:

$\mathcal{S} = \mathcal{I} \cup \mathcal{K}$, $\mathcal{T} = \mathcal{J} \cup \mathcal{K}$ and $\mathcal{R} = \mathcal{J} \setminus \mathcal{I}$, then we have

$$\begin{aligned} \mathbb{I}(\mathcal{J}) - \mathbb{I}(\mathcal{I}) &= \sum_{[i,j]\in\mathcal{J}} \mathbb{I}[i,j] - \sum_{[i,j]\in\mathcal{I}} \mathbb{I}[i,j] = \sum_{[i,j]\in\mathcal{J}\setminus\mathcal{I}} \mathbb{I}[i,j] = \sum_{[i,j]\in\mathcal{R}} \mathbb{I}[i,j] \\ &\geq 0 \end{aligned} \tag{3.13}$$

which proves that $\mathbb{I}(\mathcal{S})$ is monotonically non-decreasing.

Finally, we prove that it is submodular. Define $\mathcal{P} = \mathcal{T} \setminus \mathcal{S}$. We have that $\mathcal{P} = (\mathcal{J} \cup \mathcal{K}) \setminus (\mathcal{I} \cup \mathcal{K}) = \mathcal{R} \setminus (\mathcal{R} \cap \mathcal{K}) \subseteq \mathcal{R} = \mathcal{J} \setminus \mathcal{I}$. Then we have

$$\mathbb{I}(\mathcal{T}) - \mathbb{I}(\mathcal{S}) = \sum_{[i,j]\in\mathcal{P}} \mathbb{I}[i,j] \leq \mathbb{I}(\mathcal{J}) - \mathbb{I}(\mathcal{I}) \tag{3.14}$$

which proves the submodularity of the edge influence. QED.

The diminishing returns property naturally leads to a greedy algorithm to obtain a near-optimal solution for solving Problem 3.1. We first present the algorithm for auditing by edges. The algorithms for auditing by nodes and by subgraphs will be presented later.

With the diminishing returns property, we present the AURORA-E (Algorithm 3.1) algorithm to find top-k influential edges. The key idea of AURORA-E is to select one edge and update the gradient matrix at each of the k iterations.

Algorithm 3.1: AURORA-E

Input : the adjacency matrix $\mathbf{A}$, integer budget k.
Output : a set of k edges $\mathcal{S}$ with the highest influence

initialize $\mathcal{S} = \emptyset$;
initialize c (e.g., $c = 1/2 \max \text{eigenvalue}(\mathbf{A})$);
calculate PageRank $\mathbf{r} = \text{pg}(\mathbf{A}, \mathbf{e}, c)$;
calculate partial gradients $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12);
calculate gradients $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
while $|\mathcal{S}| \neq k$ **do**
    find $(i,j) = \underset{i,j}{\text{argmax}}\, \mathbb{I}[i,j]$ with Equation (3.3);
    add edge $[i,j]$ to $\mathcal{S}$;
    remove $[i,j]$, and remove $[j,i]$ if undirected;
    re-calculate $\mathbf{r}$, $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12), and $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
return $\mathcal{S}$;

The effectiveness and efficiency of AURORA-E are summarized in Lemma 3.1 and Lemma 3.2, respectively. We can see that AURORA-E finds a $(1 - 1/e)$ near-optimal solution with a linear complexity.

Lemma 3.1. (Approximation Ratio of AURORA-E). Let $\mathcal{S}_k = \{s_1, s_2, ..., s_k\}$ represents the set formed by AURORA-E, $\mathcal{O}$ is the optimal solution of Problem 3.1, $\mathbb{I}(\mathcal{S})$ is the influence defined in Definition 3.1.

$$\mathbb{I}(\mathcal{S}_k) \geq (1 - 1/e)\,\mathbb{I}(\mathcal{O}) \tag{3.15}$$

Proof. By diminishing returns property, $\forall\ i \leq k$, we have

$$\begin{aligned}
\mathbb{I}(\mathcal{O}) &\leq \mathbb{I}(\mathcal{O} \cup \mathcal{S}_i) = \mathbb{I}(\mathcal{S}_i) + \sum_{s \in \mathcal{O}} \Delta\left(s | \mathcal{S}_i \cup (\mathcal{O} \setminus \{s\})\right) \\
&\leq \mathbb{I}(\mathcal{S}_i) + \sum_{s \in \mathcal{O}} \Delta(s|\mathcal{S}_i) \\
&\leq \mathbb{I}(\mathcal{S}_i) + k\Delta(s_{\max}|\mathcal{S}_k)
\end{aligned} \tag{3.16}$$

where $s_{\max} = \text{argmax}_{s \in \mathcal{V} \setminus \mathcal{S}_i} \Delta(s|\mathcal{S}_i)$. Then we have

$$\Delta(s_{\max}|\mathcal{S}_k) = \mathbb{I}(\mathcal{S}_{i+1}) - \mathbb{I}(\mathcal{S}_i) \geq \frac{1}{k}\left(\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_i)\right) \tag{3.17}$$

After rearranging the terms, we have

$$\begin{aligned}
\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_{i+1}) &\leq \left(1 - \frac{1}{k}\right)\left(\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_i)\right) \\
\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_i) &\leq \left(1 - \frac{1}{k}\right)\left(\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_{i-1})\right) \\
&... \\
\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_1) &\leq \left(1 - \frac{1}{k}\right)\left(\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_0)\right)
\end{aligned}$$

Thus, by recursively applying the inequality, we have

$$\mathbb{I}(\mathcal{O}) - \mathbb{I}(\mathcal{S}_k) \leq \left(1 - \frac{1}{k}\right)^k \left(\mathbb{I}(\mathcal{O}) - f(\mathcal{S}_0)\right) = \left(1 - \frac{1}{k}\right)^k \mathbb{I}(\mathcal{O}) \leq \frac{1}{e}\mathbb{I}(\mathcal{O}) \tag{3.18}$$

Thus we have $(1 - 1/e)\,\mathbb{I}(\mathcal{O}) \leq \mathbb{I}(\mathcal{S}_{i+1})$. QED.

Lemma 3.2. (Time and space complexities of AURORA-E). Algorithm 1 is $O(mk)$ in time and $O(m + n)$ in space, where m and n are the numbers of edges and nodes in the input graph, and k is the budget.

Proof. It takes $O(m)$ time complexity to calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by applying power iterations. In the while-loop, we find the edge with the greatest influence by traversing all edges, which takes $O(m)$ time. Time spent to re-calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ remains the same as $O(m)$. Since

the body inside the loop will run k times, the overall time complexity is $O(mk)$. In Algorithm 1, it takes $O(m)$ space to save the sparse adjacency matrix $\mathbf{A}$ and $O(n)$ space to save the PageRank vector $\mathbf{r}$ and column vector $\mathbf{Q}^T\mathbf{r}$ in Equation (3.12). Therefore, it has $O(m+n)$ space complexity. QED.

Auditing by nodes: AURORA-N. By Theorem 3.1, the influence of nodes also enjoys the diminishing returns property. Following this, we design a greedy algorithm AURORA-N (Algorithm 3.2) to find a set of k influential nodes with $(1-1/e)$ approximation ratio with a linear complexity. The efficiency of AURORA-N is summarized in Lemma 3.3.

Algorithm 3.2: AURORA-N

Input : the adjacency matrix $\mathbf{A}$, integer budget k.
Output : a set of k nodes $\mathcal{S}$ with highest influence

initialize $\mathcal{S} = \emptyset$;
initialize c (e.g., $c = 1/2 \max \text{eigenvalue}(\mathbf{A})$);
calculate PageRank $\mathbf{r} = \text{pg}(\mathbf{A}, \mathbf{e}, c)$;
calculate partial gradients $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12);
calculate gradients $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
while $|\mathcal{S}| \neq k$ **do**
    find $v_i = \underset{i}{\text{argmax}}\, \mathbb{I}[i]$;
    add v_i to $\mathcal{S}$;
    remove all inbound and outbound edges of v_i;
    re-calculate $\mathbf{r}$, $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12), and $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
return $\mathcal{S}$;

Lemma 3.3. (Time and space complexities of AURORA-N). Algorithm 2 is $O(mk)$ in time and $O(m+n)$ in space, where m and n are the numbers of edges and nodes in the input graph, and k is the budget.

Proof. It takes $O(m)$ time complexity to calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by applying power iterations. In the while-loop, we calculate the influence of nodes and find the node with the greatest influence by traversing all edges, which takes $O(m)$ time. Time spent to re-calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ remains the same as $O(m)$. Since the body inside loop will run k times, the overall time complexity is $O(mk)$. In Algorithm 2, it takes $O(m)$ space to save the sparse adjacency matrix $\mathbf{A}$ and $O(n)$ space to save the PageRank vector $\mathbf{r}$ and column vector $\mathbf{Q}^T\mathbf{r}$ in Equation (3.12). Therefore, it has $O(m+n)$ space complexity. QED.

Auditing by subgraph: AURORA-S. Here, we discuss how to select an influential subgraph with k nodes, and we focus on the vertex-induced subgraph. With the diminishing returns

property (Theorem 3.1) in mind, we develop AURORA-S (Algorithm 3.3) to greedily identify the influential subgraph with $(1 - 1/e)$ approximation ratio with a linear complexity. The efficiency of AURORA-S is summarized in Lemma 3.4.

Algorithm 3.3: AURORA-S

Input : the adjacency matrix $\mathbf{A}$, output size k.
Output : a vertex-induced subgraph of k nodes $\mathcal{S}$ with highest influence

initialize $\mathcal{S} = \emptyset$;
initialize c (e.g., $c = 1/2 \max \text{eigenvalue}(\mathbf{A})$);
calculate PageRank $\mathbf{r} = \text{pg}(\mathbf{A}, \mathbf{e}, c)$;
calculate partial gradients $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12);
calculate gradients $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
while $|\mathcal{S}| \neq k$ **do**
    find $(i, j) = \underset{i,j}{\text{argmax}} \, \mathbb{I}[i, j]$;
    if $|\mathcal{S}| + 2 \leq k$ **then**
        add v_i and v_j to $\mathcal{S}$;
    else
        find the endpoint v with higher influence;
        if $v \notin \mathcal{S}$ **then**
            add v to $\mathcal{S}$;
        else
            add the other endpoint to $\mathcal{S}$;
    remove all edges in $\mathcal{S}$;
    re-calculate $\mathbf{r}$, $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by Equation (3.12), and $\frac{df(\mathbf{r})}{d\mathbf{A}}$ by Equation (3.3);
return $\mathcal{S}$;

Lemma 3.4. (Time and space complexities of AURORA-S). Algorithm 3 is $O(mk)$ in time and $O(m + n)$ in space, where m and n are the numbers of edges and nodes in the input graph, and k is the budget.

Proof. It takes $O(m)$ time complexity to calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ by applying power iterations. In the while-loop, we find the edge with the greatest influence by traversing all edges, which takes $O(m)$ time. Time spent to re-calculate $\mathbf{r}$ and $\frac{\partial f(\mathbf{r})}{\partial \mathbf{A}}$ remains the same as $O(m)$. Since the body inside loop will run k times, the overall time complexity is $O(mk)$. In Algorithm 3, it takes $O(m)$ space to save the sparse adjacency matrix $\mathbf{A}$ and $O(n)$ space to save the PageRank vector $\mathbf{r}$ and column vector $\mathbf{Q}^T\mathbf{r}$ in Equation (3.12). Therefore, it has $O(m + n)$ space complexity. QED.

Generalization and variants. The family of AURORA algorithms assumes the input graph

is a plain network. However, it is worth pointing out that AURORA algorithms also work on different types of networks and other random walk-based techniques.

A – AURORA *on normalized PageRank.* Recall that in Section 3.1.1, we remove the constraint of $\mathbf{A}$ being a normalized adjacency matrix and use the general form instead. The effect of removing that constraint will cause the L_1 norm of PageRank to be not equal to 1. We show that AURORA is also able to work on L_1 normalized PageRank. Let $S(\mathbf{r}) = \sum_i \mathbf{r}[i]$, we have

$$\frac{\partial f(r)}{\partial \mathbf{A}} = c\mathbf{Q}^T \left(-\frac{2f(\mathbf{r})}{S(\mathbf{r})}\mathbf{1} + \frac{2}{S(\mathbf{r})}\mathbf{r} \right) \mathbf{r}^T \tag{3.19}$$

Then we can apply AURORA algorithms by replacing Equation (3.12) with Equation (3.19).

B – AURORA *on Network of Networks.* Network of Networks (NoN) is a type of networks first introduced in [135] with the ability to leverage the within-network smoothness in the domain-specific network and the cross-network consistency through the main network. An NoN is usually defined as the triplet $\mathcal{R} = \langle \mathcal{G}, \mathcal{A}, \theta \rangle$, where $\mathcal{G}$ is the main network, $\mathcal{A}$ is a set of domain-specific networks, and θ is a mapping function to map the main node to the corresponding domain-specific network. In [135], CROSSRANK and CROSSQUERY are two ranking algorithms proposed to solve ranking on NoN. The authors have proved that they are actually equivalent to the well-known PageRank and random walk with restart on the integrated graph. Thus, AURORA algorithms also have the ability to audit CROSSRANK and CROSSQUERY on Network of Networks.

C – AURORA *on attributed networks.* Given a large attributed network, it is important to learn the most influential node-attribute or edge-attribute with respect to a query node. We show that AURORA algorithms have the ability to find top-k influential edge attributes and node attributes on attributed networks. The central idea is to treat attributes as *attribute nodes* and form an augmented graph with those attribute nodes. To support node attributes, let $\mathbf{A}$ be the $a \times a$ node-to-node adjacency matrix, and $\mathbf{W}$ be the $w \times a$ node-to-attribute matrix, then we can form an augmented graph $\mathcal{G} = \left(\begin{smallmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{W} & \mathbf{0} \end{smallmatrix}\right)$. To support edge attributes, similar to [136], we embed an edge-node for each edge in the input graph and define a mapping function ψ that maps each edge-node to edge attribute in the original graph. We assume $\mathbf{A}$ is the $a \times a$ node-to-node adjacency matrix, and b is the number of different edge attribute values. By embedding edge-node, it creates a $(a+b) \times (a+b)$ augmented graph $\mathbf{Y}$. Then, to find the top-k influential node attributes and edge attributes, we can easily run AURORA-N on the augmented attributed graphs $\mathcal{G}$ and $\mathbf{Y}$, respectively.

Table 3.3: Statistics of the datasets used to evaluate AURORA.

Domain	Network	Type	# Nodes	# Edges
Social	Karate	U	34	78
	Dolphins	U	62	159
	WikiVote	D	7,115	103,689
	Pokec	D	1,632,803	30,622,564
Collaboration	GrQc	U	5,242	14,496
	DBLP	U	42,252	420,640
	NBA	U	3,923	71,581
	cit-DBLP	D	12,591	49,743
	cit-HepTh	D	27,770	352,807
	cit-HepPh	D	34,546	421,578
Physical	Airport	D	1,128	18,736
Others	Lesmis	U	77	254
	Amazon	D	262,111	1,234,877

(In Type, U means undirected graph; D means directed graph.)

3.1.3 Experimental Evaluation

In this part, we evaluate AURORA. All experiments are designed to answer the following two questions:

- *Effectiveness.* How effective are AURORA algorithms in identifying key graph elements with respect to the PageRank results?
- *Efficiency.* How efficient and scalable are AURORA algorithms?

Experimental settings. Here, we discuss the detailed experimental settings to evaluate AURORA.

A – Hardware and software specifications. All datasets are publicly available. All experiments are performed on a virtual machine with 4 Intel i7-8700 CPU cores at 3.4GHz and 32GB RAM. The operating system is Windows 10. All codes are written in Python 3.6. The source code of AURORA can be downloaded at `https://jiank2.github.io/files/bigdata18/aurora-v2.zip`.

B – Dataset descriptions. We test our algorithms on a diverse set of real-world network datasets. The statistics of these datasets are listed in Table 3.3.

- *Social networks.* Here, nodes are users, and edges indicate social relationships. Among them, *Karate* [137] is a well-known network dataset of a university karate club collected in 1977. *Dolphins* [138] is an undirected social network of frequent associations between dolphins in a community living off Doubtful Sound, New Zealand. *WikiVote* [139] is

generated by Wikipedia voting data from the inception of Wikipedia till January 2008. *Pokec* [140] is a popular online social network in Slovakia.

- *Collaboration networks.* Here, nodes are individuals, and two people are connected if they have collaborated together. We use the collaboration network in the field of General Relativity and Quantum Cosmology (*GrQc*) in Physics from arXiv preprint archive[3]. *DBLP*[4] is a co-authorship network from DBLP computer science bibliography. And *NBA* [141] is a collaboration network of NBA players from 1946 to 2009. *cit-DBLP* [142] is the citation network of DBLP, a database of scientific publications such as papers and books. Each node in the network is a publication, and each edge represents a citation of a publication by another publication. *cit-HepTh* [143] is an ArXiv HEP-TH (High Energy Physics – Theory) citation network. The data covers papers from January 1993 to April 2003. If a paper i cites paper j, there is a directed edge from i to j. *cit-HepPh* [143] is an ArXiv HEP-PH (High Energy Physics – Phenomenology) citation network. The data covers papers from January 1993 to April 2003. If a paper i cites paper j, there is a directed edge from i to j.

- *Physical infrastructure networks.* This category refers to the networks of physical infrastructure entities. Nodes in them correspond to physical infrastructure, and edges are connections. *Airport*[5] is a dataset of airline traffic. Each node represents an airport in the United States, and an edge (i, j) represents the airline from i to j while the edge weight stands for the normalized number of passengers.

- *Others.* This category contains networks that do not fit into the above categories. *Lesmis* [144] is a network of co-appearances of characters in Victor Hugo's novel *"Les Miserables"*. A node represents a character and an edge connects a pair of characters if they both appear in the same chapter of the book. *Amazon* [145] is a co-purchasing network collected by crawling Amazon website. It is based on the *Customers Who Bought This Item Also Bought* feature.

C – Baseline methods. We compare AURORA with several baseline methods, which are summarized as follows.

- *Random selection (random).* Randomly select k elements and calculate the change by removing them.

[3] `https://arxiv.org/`
[4] `http://dblp.uni-trier.de/`
[5] `https://www.transtats.bts.gov/`

- *Top-k degrees (degree).* We first define the degree of an edge (u, v) as follows,

$$\deg(u,v) = \begin{cases} (\deg(u) \times \deg(v)) \times \max\limits_{i \in \{u,v\}} \deg(i), & \text{if undirected} \\ (\deg(u) \times \deg(v)) \times \deg(u), & \text{if directed} \end{cases} \tag{3.20}$$

  where $\deg(u)$ represents the degree of node u. To audit by graph elements, we select k elements with the highest degrees. For edges, we select k edges with the highest edge degrees defined above; for nodes, we select k nodes with the highest node degrees; for subgraphs, we form a vertex-induced subgraph from k nodes with the highest degrees.

- *PageRank.* We first define the PageRank score of an edge (u, v) as follows,

$$\mathrm{r}(u,v) = \begin{cases} (\mathbf{r}[u] \times \mathbf{r}[v]) \times \max\limits_{i \in \{u,v\}} \mathbf{r}[i], & \text{if undirected} \\ (\mathbf{r}[u] \times \mathbf{r}[v]) \times \mathbf{r}[u], & \text{if directed} \end{cases} \tag{3.21}$$

  where $\mathbf{r}[u]$ is the PageRank score of node u. To audit by graph elements, we select k elements with the highest PageRank scores. That is, for edges, we select k edges with the highest PageRank scores defined above; for nodes, we select k nodes with highest PageRank scores; for subgraphs, we form a vertex-induced subgraph from k nodes with the highest PageRank scores.

- *HITS.* We first define HITS score of an edge (u, v) and node u as follows,

$$\mathrm{HITS}(u,v) = \mathrm{hub}(u) \times \mathrm{hub}(v) + \mathrm{auth}(u) \times \mathrm{auth}(v) \tag{3.22}$$

$$\mathrm{HITS}(u) = \mathrm{hub}(u) + \mathrm{auth}(u) \tag{3.23}$$

  where $\mathrm{hub}(u)$ and $\mathrm{auth}(u)$ represent the hub score and authority score of node u, respectively. To audit by graph elements, we select k elements with the highest HITS scores. That is, for edges, we select k edges with the highest HITS scores defined above; for nodes, we select k nodes with the highest HITS scores; for subgraphs, we form a vertex-induced subgraph from k nodes with the highest HITS scores.

D – Evaluation metrics. Here, we choose the loss function to be squared L_2 norm. We quantify the performance of auditing by the goodness score Δf of the graph elements $\mathcal{S}$

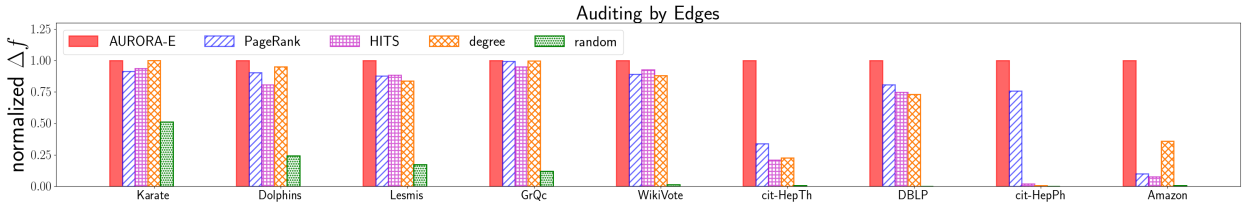


Figure 3.2: Auditing results by edges. Budget $k = 10$. Higher is better. Best viewed in color.

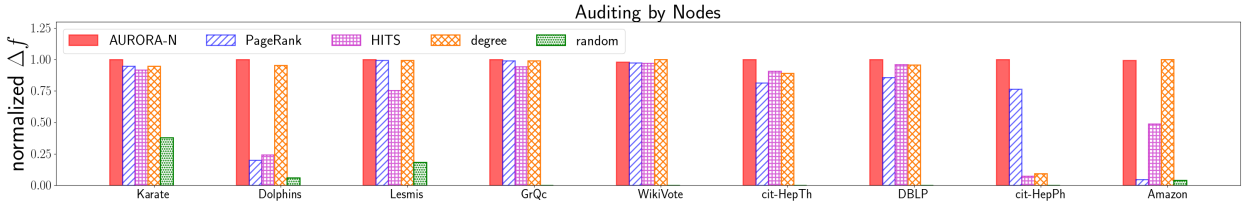


Figure 3.3: Auditing results by nodes. Budget $k = 10$. Higher is better. Best viewed in color.

found by the corresponding algorithms. The goodness score Δf is defined as

$$\Delta f = \left| f\left(\mathbf{r}/\sum_{i=1}^{n}\mathbf{r}\left[i\right]\right) - f\left(\mathbf{r}_{\mathcal{S}}/\sum_{i=1}^{n}\mathbf{r}_{\mathcal{S}}\left[i\right]\right)\right| \tag{3.24}$$

Effectiveness results. The effectiveness results of AURORA are as follows.

A – Quantitative comparison. We perform effectiveness experiments with the baseline methods. We set k from 1 to 10 and find k influential edges and nodes, respectively. We set k only from 2 to 10 to find an influential subgraph of size-k, respectively. This is because a vertex-induced subgraph with only 1 node does not contain any edge and therefore is meaningless for the purpose of PageRank auditing. On large graphs, searching a ground-truth with k most influential elements by brute force is prohibitively expensive due to its combinatorial nature. For example, even if we use the small *Lesmis* dataset, it will take over a day to find ground-truth with $k = 5$. Thus, we do not report ground-truth results for all datasets.

The results of quantitative comparison across 9 different datasets are shown from Figure 3.2 to Figure 3.4. From those figures, we have observed that our family of AURORA algorithms consistently outperform other baseline methods on all datasets. Figure 3.5 and Figure 3.6 shows the effect of k on auditing results. We can observe the following findings from those figures. (1) Our family of AURORA algorithms incrementally find influential graph elements with respect to the budget k; (2) AURORA algorithms consistently outperform baseline methods on different budgets.

B – Case study on the Airport dataset. A natural use case of our AURORA algorithms is to find influential edges and nodes in a given graph. To demonstrate that our algorithms are

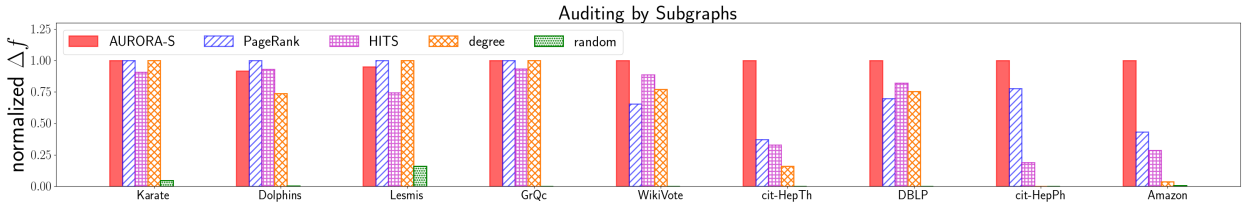
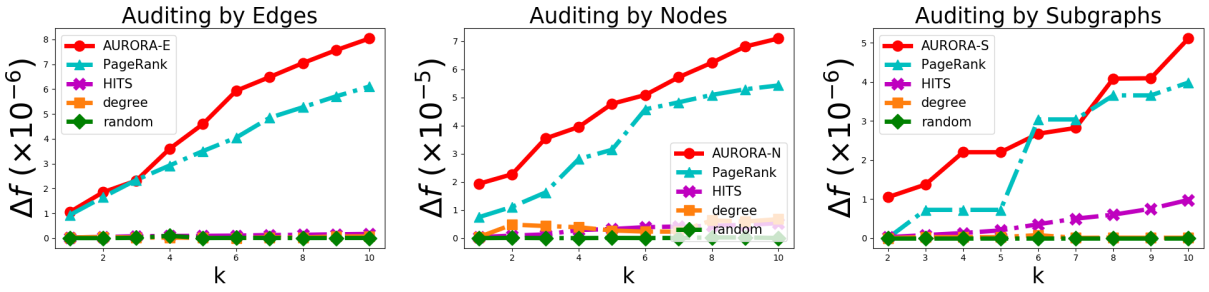


Figure 3.4: Auditing results by subgraphs. Budget $k = 10$. Higher is better. Best viewed in color.



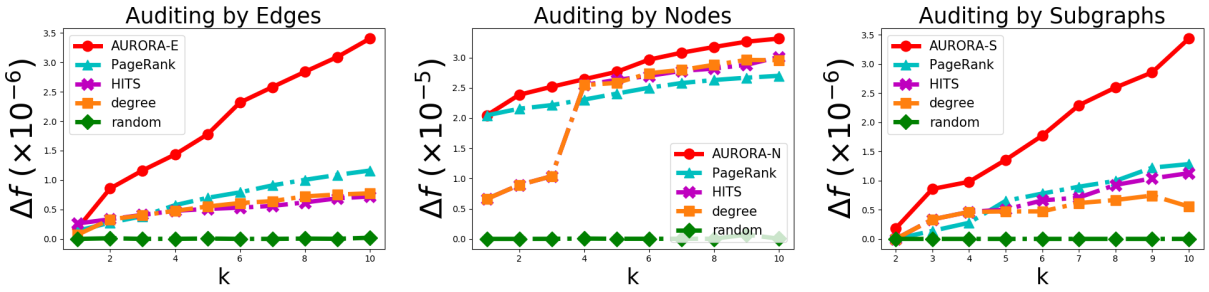
(a) Auditing by edges. (b) Auditing by nodes. (c) Auditing by subgraphs.

Figure 3.5: Effect of k on auditing results (cit-HepPh Dataset). Higher is better. Best viewed in color.

indeed able to provide intuitive information, we test our algorithms on the *Airport* dataset. This dataset was manually created from the commercial airline traffic data in 2017, which is provided by the United States Department of Transportation. We perform AURORA-E and AURORA-N to find the most influential airlines (edges in the graph) and airports (nodes in the graph) across United States with $k = 7$.

Edges selected by AURORA-E are `ATL-LAX`, `LAX-ATL`, `ATL-ORD`, `ORD-ATL`, `ATL-DEN`, `DEN-ATL` and `LAX-ORD`. In contrast, PageRank selects *ATL-LAS* instead of `DEN-ATL` and *ATL-DFW* instead of `LAX-ORD`. `DEN-ATL` plays a more important role in determining the centrality (e.g., PageRank) of other airports. This is because `DEN` serves as one of the busiest hub airports that connects West coast and East coast; while *ATL-LAS* is less important in that regard, considering the existence of `ATL-LAX` and `ATL-PHX`. Comparing `LAX-ORD` and *ATL-DFW*, `LAX-ORD` directly connects Los Angeles and Chicago, both of them are largest cities in the United States.

In the scenario of node-auditing, AURORA-N selects `ATL`, `LAX`, `ORD`, `DFW`, `DEN`, `LAS` and `CLT`. In contrast, PageRank selects *SFO* instead of `CLT`. `CLT` seems to be a more reasonable choice because it serves as a major hub airport, the 6$^{\text{th}}$ busiest airport by FAA statistics, to connect many regional airports around states like North Carolina, South Carolina, Virginia, West Virginia, etc. Compared with `CLT`, *SFO* is less influential in that regard, mainly due to the



(a) Auditing by edges. (b) Auditing by nodes. (c) Auditing by subgraphs.

Figure 3.6: Effect of k on auditing results (cit-HepTh Dataset). Higher is better. Best viewed in color.

following two reasons. (1) It ranks after `CLT` (7^{th} vs. 6^{th}) in the list of busiest airports by the FAA statistics; (2) due to the location proximity of *SFO* to `LAX` and `SJC`, even if this node is perturbed (i.e., absent), many surrounding airports (especially regional airports in California) could still be connected via `LAX` and `SJC`.

C – Case study on the DBLP dataset. Another interesting use case of AURORA algorithms is sense-making in graph proximity. We construct a co-authorship network from DBLP computer science bibliography to test our algorithms. We perform AURORA-N and PageRank with $k = 6$. Different from the previous case study, here we use a personalized PageRank with the query node being `Christos Faloutsos`. In this case, the top-ranked scholars in the resulting ranking vector $\mathbf{r}$ form the proximity (i.e., 'neighborhood') of the query node (i.e., who are most relevant to `Christos Faloutsos`). Consequently, the nodes selected by an auditing algorithm indicate those important nodes in terms of making/maintaining the neighborhood of the query node. Comparing the results by AURORA-N and PageRank, 5 of them are the same while AURORA-N selects `Jure Leskovec` instead of *Yannis Ioannidis*. This result is consistent with the intuition, since `Jure Leskovec`, as the former student of `Christos Faloutsos` with lots of joint publications, plays a more prominent role in the neighborhood of `Christos Faloutsos` by sharing more common collaborators.

D – Case study on the NBA dataset. In a collaboration network, a subgraph can be naturally viewed as a team (e.g., sports team). From this perspective, AURORA-S has the potential to find teammates of a player. We set the query node as `Tracy McGrady`. Since there are average 14 players for each team in NBA, we set the budget $k = 14$. Comparing the results by AURORA-N and PageRank, 13 of them are the same. However, AURORA-N selects `Rafer Alston` instead of `Steven Hunter`. As `Tracy`'s teammate, we believe `Rafer Alston` is a more important collaborator and teammate mainly because of the following two reasons. (1) `Rafer Alston` played more seasons with `Tracy McGrady` than `Steven Hunter` (4 seasons,

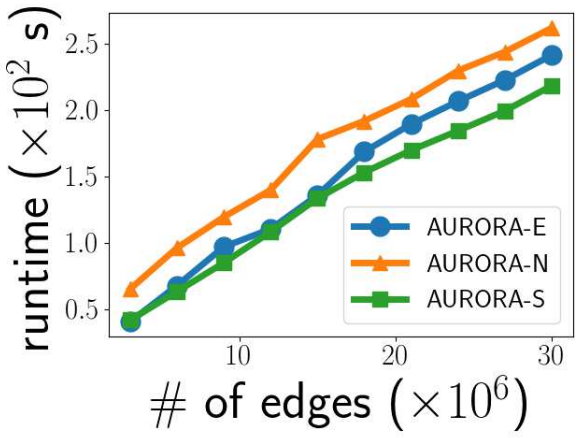


Figure 3.7: Running time vs. the number of edges on Pokec dataset.

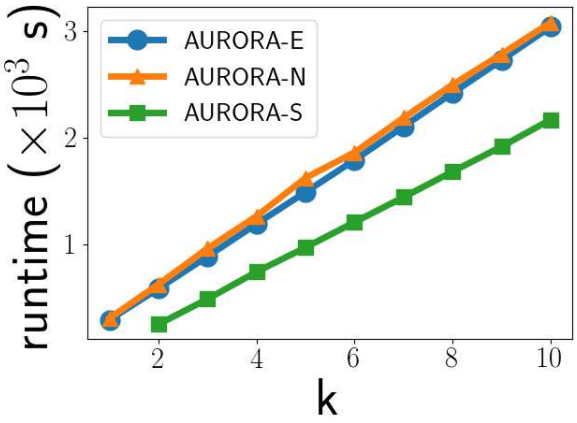


Figure 3.8: Running time vs. number of k on Pokec dataset.

191 games vs. 3 seasons, 129 games); and (2) he played more games in the starting lineups with `Tracy McGrady` than `Steven Hunter` (187 games vs. 47 games).

Efficiency results of AURORA. We show the running time vs. the number of edges m and budget size k on *Pokec* dataset in Figure 3.7 and Figure 3.8. We can see that AURORA algorithms scale linearly with respect to m and k, respectively. This is consistent with our complexity analysis that the family of AURORA algorithms is linear with respect to the number of edges and the budget.

Visualization. To better understand the auditing results, we developed a prototype system with D3.js to represent the influence of graph elements visually. In the system, we use the strength of line to represent the gradient of an edge and use the size and color for the gradient of nodes. An example of visualizing hand-crafted toy graph is shown in Figure 3.9. It is easy to see in the figure that Node 5 is the most influential node, and edges around Node 5 is more influential than other edges, both of which are consistent with our intuition.

3.2 NETWORK DERIVATIVE MINING

Network mining plays a pivotal role in many important real-world applications, including information retrieval [1, 2], healthcare [131], social network analysis [146], security [80], and recommender systems [128]. Throughout the years, researchers have developed many network mining algorithms for various mining tasks. To name a few, HITS [2] is a well-known and widely used ranking algorithm to measure node importance by considering the network structure; spectral clustering [3] is a popular technique for community detection and image segmentation; and matrix factorization-based completion [20] on a bipartite network is a key

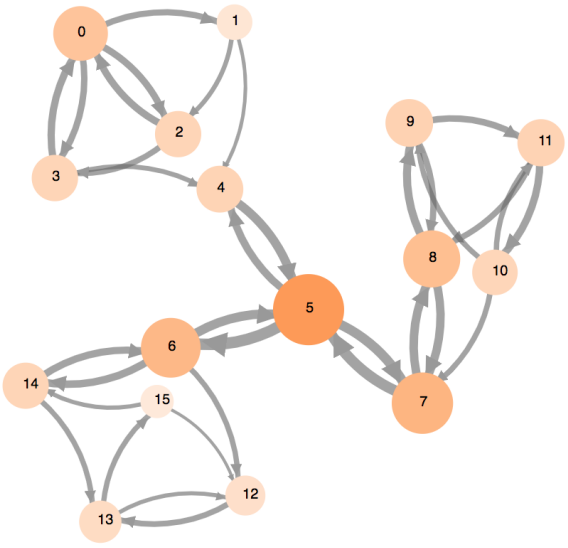


Figure 3.9: Visualization of AURORA results on the toy graph. Best viewed in color.

enabling technology for modern recommender systems.

State-of-the-art network mining algorithms have been widely adopted in various real-world applications, which often deliver a strong empirical performance in finding interesting patterns, e.g., which webpages are the most important, who are grouped into the same online community and which movies best suit users' tastes, etc. Despite the tremendous progress, it remains opaque on *how the results of a given mining algorithm relate to the underlying network structure*. Consequently, it is often hard to answer questions like *how* the ranking results for webpages might be manipulated by malicious link farms; *why* two seemingly different users are grouped into the same online community; *how* sensitive the recommendation results are due to the random noisy or fake ratings; and *what* would have happened to the epidemic dynamics *if* we had distributed vaccines in a different way, etc.

To tackle this issue, we introduce a paradigm shift of network mining and introduce the *network derivative mining* (N2N) problem. To be specific, given an input network and a mining algorithm, it aims to find a derivative network, each of whose edges provides a quantitative measurement of the influence of the corresponding edge of the input network on the given mining algorithm. In detail, we define the influence of edges as the rate of change of a function over the mining results induced by the given mining algorithm. We envision that network derivative mining will benefit a variety of aspects. First, it is directly applicable to adversarial network mining, where users can identify potential edges which, if attacked, will drastically affect network mining results. Second, it will render the crucial explainability of the network mining model by identifying the most responsible/relevant edges for the mining

results. It can further help answer questions like why a node belongs or does not belong to a certain cluster. Third, the derivative network can be used as a quantitative reference for sensitivity analysis on network structure. Fourth, the network derivative mining has great potential in active learning where edges with the high influence act as the most valuable data points to query the oracle. Fifth, it will offer an effective way to encode side information to boost some network mining tasks, e.g., to learn an optimal network based on user feedback [133]. Finally, it allows the end users to quickly examine how the mining results would differ should the underlying network have changed, and thus it naturally fits for counterfactual learning on networks.

Besides the problem definition, the main contributions of this work are summarized as follows.

- *Algorithmic framework.* We formulate the network derivative mining problem as an optimization problem and introduce a generic algorithmic framework. Its key idea is to measure the influence as the rate of change of a scalar valued function over the given network mining task.
- *Instantiations and computation.* We instantiate the developed framework by three classic network mining tasks, including ranking, clustering, and completion. For each task, we develop effective and efficient algorithm to construct the corresponding derivative network with a linear complexity in both time and space.
- *Empirical evaluations.* We perform extensive experiments on diverse, real-world datasets. The experimental results demonstrate that our method (a) is effective in adversarial network mining for different instantiations and (b) scales linearly with respect to the number of nodes and edges in the network.

3.2.1 Problem Definition

In this part, we first present a table of symbols used throughout this work (Table 3.4). Then we review the general procedure of classic network mining tasks. Finally, we formally define the network derivative mining problem.

In this paper, we denote matrices with bold upper-case letters (e.g., $\mathbf{A}$), vectors with bold lower-case letters (e.g., $\mathbf{x}$), sets with calligraphic fonts (e.g., $\mathcal{S}$), and scalars with italic lower-case letters (e.g., c). For matrix indexing conventions, we use the rules similar to Numpy in Python as follows. We use $\mathbf{A}[i, j]$ to denote the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column, $\mathbf{A}[i, :]$ to denote the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}[:, j]$ to denote the

Table 3.4: Table of symbols for N2N.

Symbol	Definition
$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$	the input network
(i, j)	edge from node i to node j
$\mathbf{A}$	a matrix
$\mathbf{A}[i, j]$	the element at the i-th row and the j-th column
$\mathbf{A}[i, :]$	the i-th row of matrix $\mathbf{A}$
$\mathbf{A}[:, j]$	the j-th column of matrix $\mathbf{A}$
$\mathbf{A}^T$	transpose of the matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of the matrix $\mathbf{A}$
$\mathbf{u}$	a vector
$\mathbf{u}[i]$	the i-th element of vector $\mathbf{u}$
$\mathbb{I}[i, j]$	the influence of edge $[i, j]$
$L(\cdot)$	the loss function for a mining task
$\mathcal{Y}^*$	the optimal model output
θ	a set of parameters
$f(\mathcal{Y}^*)$	a scalar function over the mining results
n	number of nodes
m	number of edges

j-th column of matrix $\mathbf{A}$. We use superscript ^{T} to denote the transpose of matrix (i.e., $\mathbf{A}^T$ is the transpose of matrix $\mathbf{A}$).

Generally speaking, given a network $\mathbf{A}$ with n nodes and m edges, a network mining algorithm aims to learn mining results $\mathcal{Y}^*$ by optimizing a loss function $L(\mathbf{A}, \mathcal{Y}, \theta)$, where $\mathcal{Y}^* = \text{argmin}_{\mathcal{Y}} L(\mathbf{A}, \mathcal{Y}, \theta)$ is the optimal model output, and θ is a set of additional parameters that corresponds to a specific mining task. Let us illustrate this using three classic examples. Table 3.5 presents a summary.

The first example is HITS [2], which is a widely-used ranking algorithm that measures the importance of nodes with hub scores $\mathbf{u}$ and authority scores $\mathbf{v}$ for network structure analysis. It solves the linear system $\mathbf{u} = \mathbf{A}\mathbf{v}, \quad \mathbf{v} = \mathbf{A}^T\mathbf{u}$, which can be naturally formulated as the following optimization problem,

$$\min_{\mathbf{u},\mathbf{v}} ||\mathbf{A} - \mathbf{u}\mathbf{v}^T||_F^2 \tag{3.25}$$

The second example is spectral clustering, which aims to find a matrix $\mathbf{U}$ with r orthonormal column vectors by the following optimization problem,

$$\min_{\mathbf{U}} \text{Tr}\left(\mathbf{U}^T\mathbf{L}\mathbf{U}\right) \quad \text{s.t. } \mathbf{U}^T\mathbf{U} = \mathbf{I} \tag{3.26}$$

where $\mathbf{L}$ is the Laplacian matrix of adjacency matrix $\mathbf{A}$, and r is the number of clusters. It is well-known that $\mathbf{U}^T\mathbf{L}\mathbf{U}$ is essentially the diagonal matrix of the r smallest eigenvalues of $\mathbf{L}$, and columns in $\mathbf{U}$ are the associated eigenvectors.

The third example is matrix-factorization based completion, where we are given a bipartite network $\mathbf{A}$ with n_1 users, n_2 items, and m observations. We denote $\mathbf{A}[i,j]$ as the rating of the j-th item made by the i-th user. With the low-rank assumption, matrix factorization-based completion finds two low-rank matrices with r latent factors, namely $\mathbf{U}$ and $\mathbf{V}$, such that

$$\min_{\mathbf{U},\mathbf{V}} ||\text{proj}_{\Omega}(\mathbf{A} - \mathbf{U}\mathbf{V}^T)||_F^2 + \lambda_u||\mathbf{U}||_F^2 + \lambda_v||\mathbf{V}||_F^2 \tag{3.27}$$

where $\Omega = \{(i,j) : \mathbf{A}[i,j] \text{ is observed}\}$, and λ_u and λ_v are two hyperparameters for regularization. Each row of the factorized $n_1 \times r$ matrix $\mathbf{U}$ and each row of the $n_2 \times r$ matrix $\mathbf{V}$ represent a latent vector for the corresponding user and item, respectively.

Though these network mining algorithms have achieved a remarkable empirical performance in finding various patterns for the corresponding network mining tasks, they often lack effective and efficient ways to characterize how such results relate to the input network's structure. Following an overarching principle laid in [134, 147, 148], we adopt the influence functions to quantify the impact of network structure (e.g., edges) when perturbed. Based on that, we go an extra mile to further construct a *derivative network*, where each edge measures the influence of the corresponding edge of the input network on the given mining algorithm. Formally, we define the network derivative mining problem as follows.

Problem 3.2. The network derivative mining problem (N2N).

Input: (1) an input network with adjacency matrix $\mathbf{A}$ and (2) a network mining algorithm represented as $L(\mathbf{A}, \mathcal{Y}, \theta)$, where $L(\cdot)$ is the loss function, $\mathcal{Y}^* = \text{argmin}_{\mathcal{Y}} L(\mathbf{A}, \mathcal{Y}, \theta)$ is the model output, and θ contains all the additional parameters.
Output: a derivative network $\mathbf{B}$, which has the same node set as the input network $\mathbf{A}$, where $\mathbf{B}[i,j]$ measures the influence of edge $\mathbf{A}[i,j]$ on $\mathcal{Y}^*$, and $\mathbf{B}[i,j] = 0$ if $\mathbf{A}[i,j]$ does not exist.

Remarks. In this work, we focus on the derivatives of existing edges (i.e., $\mathbf{A}[i,j] = 1$). Nonetheless, the developed technique for computing the influence $\mathbf{B}[i,j]$ naturally applies to non-existing edges (i.e., $\mathbf{A}[i,j] = 0$).

Table 3.5: Examples of network mining algorithms studied in N2N.

Mining Task	**Loss Function** L	**Mining Results** $\mathcal{Y}$	**Parameters** θ	[6]**Scalar Function** $f(\cdot)$
Ranking by HITS	$\min_{\mathbf{u},\mathbf{v}} \ \|\mathbf{A}-\mathbf{u}\mathbf{v}^T\|_F^2$	hub vector $\mathbf{u}$ authority vector $\mathbf{v}$	none	$f(\mathcal{Y}^*)=\lambda_1-\lambda_2$
Spectral Clustering	$\min_{\mathbf{U}} \ \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L}\mathbf{U}\right)$ s.t. $\mathbf{U}^T\mathbf{U}=\mathbf{I}$	matrix $\mathbf{U}$	number of clusters r	$f(\mathcal{Y}^*)=\sum_{i=1}^{r}\lambda_i$
Matrix Completion	$\min_{\mathbf{U},\mathbf{V}} \ \|\mathrm{proj}_\Omega(\mathbf{A}-\mathbf{U}\mathbf{V}^T\|_F^2$ $+\lambda_u\|\mathbf{U}\|_F^2+\lambda_v\|\mathbf{V}\|_F^2$	user matrix $\mathbf{U}$ item matrix $\mathbf{V}$	latent dimensions r regularization parameters λ_u, λ_v	$f(\mathcal{Y}^*)=\|\mathbf{U}\mathbf{V}^T\|_F^2$

3.2.2 N2N Algorithmic Framework

In this part, we present a generic algorithmic framework for the network derivative mining problem. We first define the influence of edges and then formulate the network derivative mining problem from the optimization perspective, followed by a generic algorithmic framework to solve it. Formally, we define the influence as the rate of change in $f(\mathcal{Y}^*)$ for different edges.

Definition 3.2. (Edge influence). Let $\mathbf{B}$ be the derivative network, $\mathcal{Y}^*$ be the optimal result of a network mining task, and $f(\cdot)$ be a scalar function defined over the mining result $\mathcal{Y}^*$, the influence of an edge $[i,j]$ is defined as the derivative of $f(\mathcal{Y}^*)$ with respect to the corresponding edge in the input network $\mathbf{A}$, i.e., $\mathbb{I}[i,j]=\mathbf{B}[i,j]=\frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}[i,j]}$.

Based on Definition. 3.2, the network derivative mining problem can be naturally formulated as the following optimization problem

$$\mathbf{B}=\frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}} \quad \text{s.t. } \mathcal{Y}^*\in\operatorname*{argmin}_{\mathcal{Y}} L(\mathbf{A},\mathcal{Y},\theta) \tag{3.28}$$

where $L(\cdot)$ is the loss function of a network mining task with θ being the additional parameters from Table 3.5. To be specific, we have that

$$\mathbf{B}=\frac{\mathrm{d}f(\mathcal{Y}^*)}{\mathrm{d}\mathbf{A}}=\begin{cases}\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}+\left(\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}\right)^T-\mathrm{diag}\left(\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}\right), & \text{if undirected}\\ \frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}, & \text{if directed}\end{cases} \tag{3.29}$$

We can see that for both directed and undirected networks, the key to constructing the derivative network $\mathbf{B}$ is $\frac{\partial f(\mathcal{Y}^*)}{\partial\mathbf{A}}$. Therefore, in the remaining of this work, we will mainly focus

[6]It is worth mentioning that the N2N framework is applicable to other choices of loss function $f(\cdot)$, as long as it is a differentiable or subdifferentiable scalar valued function, e.g., L_p norm of a vector, $L_{1,1}$ or Frobenius norm of a matrix, soft maximum to approximate the largest entry value in the vector, etc. It would be an interesting future direction to study (1) how to automatically choose the 'best' loss function for a specific mining task and (2) how to generalize the N2N framework for non-differentiable loss function.

on effective and efficient computation of this quantity (i.e., $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$). Based on Equation (3.29), we develop a generic algorithmic framework to solve Problem 3.2, which is summarized in Algorithm 3.4. The key idea is to generate the derivative network by applying Equation (3.28) and (3.29). In Algorithm 3.4, it first runs the network mining algorithm and get the optimal model output $\mathcal{Y}^*$ (step 1). Based on the output $\mathcal{Y}^*$ and the corresponding scalar function f, it calculates the partial derivatives of $f(\mathcal{Y}^*)$ with respect to the network adjacency matrix $\mathbf{A}$ (step 2), and then uses it to generate the derivative network finally (step 3).

Algorithm 3.4: N2N Algorithm Framework

Input : the adjacency matrix $\mathbf{A}$, a mining algorithm $L(\mathbf{A}, \mathcal{Y}, \theta)$, and a scalar function $f(\cdot)$.

Output : the derivative network $\mathbf{B}$.

1 calculate $\mathcal{Y}^* = \underset{\mathcal{Y}}{\operatorname{argmin}}\ L(\mathbf{A}, \mathcal{Y}, \theta)$;

2 calculate partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$;

3 generate derivative network $\mathbf{B} = \frac{df(\mathcal{Y}^*)}{d\mathbf{A}}$ by Equation (3.29) ;

4 **return** $\mathbf{B}$;

There are two key challenges remained in Algorithm 3.4.

- *(C1) How to compute Equation* (3.28) *to generate the derivative network?* The key step to computing Equation (3.28) requires the partial derivative of the optimal mining result $\mathcal{Y}^*$ with respect to the entire input network $\mathbf{A}$, and the optimal mining result $\mathcal{Y}^*$ itself involves a potentially complicated optimization problem.
- *(C2) How to scale up the computation to large networks?* Even if we can compute the influence of each edge with a reasonable time complexity (e.g., linear complexity with respect to the input network size), the entire Algorithm 3.4, which iterates over *every* edge and calculates the corresponding influence, could still bear a superlinear complexity in both time and space, which makes it hard to scale up to large networks.

In the next part, we instantiate the N2N framework using three classic network mining tasks with effective and efficient algorithms to address these two challenges.

3.2.3 N2N Instantiation and Computation

In this part, we provide the instantiations of the N2N framework for three classic network mining tasks shown in Table 3.5. For each mining task, we start with the specific choice of $f(\cdot)$ function, then present the mathematical details on how to compute the derivative

network $\mathbf{B}$ (i.e., C1 challenge), and finally design efficient ways to scale-up the computation (i.e., C2 challenge).

Instantiation #1: Ranking by HITS. Here, we discuss how to construct the derivative network for HITS ranking. We discuss how to choose the scalar function $f(\cdot)$, how to calculate the derivative network, as well as how to scale up the computation.

A – Choice of $f(\cdot)$ function. Given an input network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathbf{A}$ being the adjacency matrix, and let $\mathbf{u}$ and $\mathbf{v}$ be its associated hub vector and authority vector, respectively, HITS algorithm iteratively solves the linear equations $\mathbf{u} = \mathbf{A}\mathbf{v}$, $\mathbf{v} = \mathbf{A}^T\mathbf{u}$ until convergence to find the final hub and authority vectors. It is well-known that the hub vector $\mathbf{u}$ is the principal eigenvector associated with the leading eigenvalue of $\mathbf{A}\mathbf{A}^T$, while the authority vector $\mathbf{v}$ is the principal eigenvector associated with the leading eigenvalue of $\mathbf{A}^T\mathbf{A}$.

Our choice of $f(\cdot)$ function for HITS is inspired by [23], which proves that hubs and authorities are actually sensitive to the eigengap of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$. Furthermore, it is known that the eigenvalues of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ are the same. Therefore, we choose the scalar function over the mining results $f(\cdot)$ to be the eigengap between first and second largest eigenvalues to reflect the stability of the ranking results. That is, $f(\mathcal{Y}^*) = \lambda_1 - \lambda_2$, where λ_1 and λ_2 are the first and second largest eigenvalues of $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$.

B – Calculating the derivative network $\mathbf{B}$. A key step to generating the derivative network $\mathbf{B}$ is to calculate the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = \frac{\partial \lambda_1}{\partial \mathbf{A}} - \frac{\partial \lambda_2}{\partial \mathbf{A}}$, which is summarized in Lemma 3.5.

Lemma 3.5. (N2N for HITS). For a given input network with adjacency matrix $\mathbf{A}$, the partial derivative of eigengap with respect to the adjacency matrix is $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = 2\left(\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} - \mathbf{u}_2\mathbf{u}_2^T\mathbf{A}\right)$, where $\mathbf{u}_1$ and $\mathbf{u}_2$ are the eigenvectors associated with λ_1 and λ_2, respectively.

Proof. We mainly show how to calculate $\frac{\partial \lambda_1}{\partial \mathbf{A}}$, since $\frac{\partial \lambda_2}{\partial \mathbf{A}}$ can be calculated in a very similar way. The key idea is to obtain the representation of $\frac{\partial \lambda_1}{\partial \mathbf{A}[i,j]}$, which is the partial derivative with respect to each element in $\mathbf{A}$. By the chain rule of matrix derivative, we have that $\frac{\partial \lambda_1}{\partial \mathbf{A}[i,j]} = \mathrm{Tr}\left[\left(\frac{\partial \lambda_1}{\partial \mathbf{A}\mathbf{A}^T}\right)^T \frac{\partial \mathbf{A}\mathbf{A}^T}{\partial \mathbf{A}[i,j]}\right]$.

We follow a similar strategy to calculate the first term in the chain rule by computing $\frac{\partial \lambda_1}{\partial (\mathbf{A}\mathbf{A}^T)[i,j]}$. Since $\mathbf{A}\mathbf{A}^T$ is a real symmetric matrix, the partial derivative of its eigenvalue can be written as

$$\frac{\partial \lambda_1}{\partial \left(\mathbf{A}\mathbf{A}^T\right)[i,j]} = \mathbf{u}_1^T \frac{\partial \mathbf{A}\mathbf{A}^T}{\partial \left(\mathbf{A}\mathbf{A}^T\right)[i,j]} \mathbf{u}_1 = \mathbf{u}_1^T[i]\,\mathbf{u}_1[j] \tag{3.30}$$

where $\mathbf{u}_1$ is the eigenvector associated with λ_1. By Equation (3.30), we can write out its matrix form solution as follows, where each element is the derivative of eigenvalue with

respect to the corresponding element in $\mathbf{A}\mathbf{A}^T$

$$\frac{\partial \lambda_1}{\partial \mathbf{A}\mathbf{A}^T} = \mathbf{u}_1 \mathbf{u}_1^T \tag{3.31}$$

Then we show how to calculate the second term in chain rule $\frac{\partial \mathbf{A}\mathbf{A}^T}{\partial \mathbf{A}[i,j]}$. By the property of derivative of matrix multiplication, we have that

$$\frac{\partial \mathbf{A}\mathbf{A}^T}{\partial \mathbf{A}\,[i,j]} = \frac{\partial \mathbf{A}}{\mathbf{A}\,[i,j]}\mathbf{A}^T + \mathbf{A}\frac{\partial \mathbf{A}^T}{\mathbf{A}\,[i,j]} = \mathbf{S}^{ij}\mathbf{A}^T + \mathbf{A}\mathbf{S}^{ji} \tag{3.32}$$

where $\mathbf{S}^{ij}$ is a single-entry matrix with 1 at the i-th row and the j-th column and 0 elsewhere. Putting Equation (3.31) and Equation (3.32) together, we have that $\frac{\partial \lambda_1}{\partial \mathbf{A}[i,j]} = 2\left[\mathbf{u}_1\,[i]\left(\mathbf{u}_1^T\mathbf{A}\right)[j]\right]$. With that in mind, we obtain its matrix form solution as

$$\frac{\partial \lambda_1}{\partial \mathbf{A}} = 2\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} \tag{3.33}$$

Following the same strategy for the second largest eigenvalue λ_2, we can write out $\frac{\partial \lambda_2}{\partial \mathbf{A}[i,j]} = 2[\mathbf{u}_2\,[i]\,(\mathbf{u}_2^T\mathbf{A})\,[j]]$ with the matrix form solution to be

$$\frac{\partial \lambda_2}{\partial \mathbf{A}} = 2\mathbf{u}_2\mathbf{u}_2^T\mathbf{A} \tag{3.34}$$

Combining Equation (3.33) and Equation (3.34) together, we obtain the matrix form solution to complete the proof, i.e.,

$$\frac{\partial f\left(\mathcal{Y}^*\right)}{\partial \mathbf{A}} = \frac{\partial \lambda_1}{\partial \mathbf{A}} - \frac{\partial \lambda_2}{\partial \mathbf{A}} = 2\left(\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} - \mathbf{u}_2\mathbf{u}_2^T\mathbf{A}\right) \tag{3.35}$$

where each element in the matrix is its partial derivative with respect to corresponding element in the adjacency matrix $\mathbf{A}$. QED.

Based on Lemma 3.5, the derivative network $\mathbf{B}$ can be calculated by applying Equation (3.35) and Equation (3.29).

C – Scale-up Computation. We have shown in Lemma 3.5 how to calculate $\mathbf{B}$. However, directly calculating $\mathbf{u}_1\mathbf{u}_1^T\mathbf{A}$ and $\mathbf{u}_2\mathbf{u}_2^T\mathbf{A}$ in Equation (3.35) requires matrix-matrix multiplication for $n \times n$ matrices, which would impose an $\Omega(n^2)$ lower-bound on the complexity. To address this issue, we explore the low-rank structure of Equation (3.35), where $\mathbf{u}_1$ is the $n \times 1$ eigenvector and $\mathbf{u}_1^T\mathbf{A}$ is a $1 \times n$ row vector. Similar properties also hold for $\mathbf{u}_2$ and $\mathbf{u}_2^T\mathbf{A}$. Furthermore, since $\mathbf{u}_1$ is the eigenvector of $\mathbf{A}\mathbf{A}^T$ which is actually the first left singular vector associated with the largest singular value of $\mathbf{A}$, we can show in Lemma 3.6 that $\mathbf{u}_1\mathbf{u}_1^T\mathbf{A}$

can be computed by truncated singular value decomposition (SVD) on $\mathbf{A}$ in a much more compact way.

Lemma 3.6. (Efficient computation of N2N for HITS). For a given input network with adjacency matrix $\mathbf{A}$, the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = 2\left(\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} - \mathbf{u}_2\mathbf{u}_2^T\mathbf{A}\right)$ can be calculated by a rank-2 truncated SVD on $\mathbf{A}$.

Proof. Let δ_1 and δ_2 be the first and second largest singular values associated with the left singular vectors $\mathbf{u}_1$ and $\mathbf{u}_2$, and let $\mathbf{v}_1$ and $\mathbf{v}_2$ be the corresponding right singular vectors. By truncated SVD on $\mathbf{A}$, we have

$$\mathbf{A} = [\mathbf{u}_1\ \mathbf{u}_2] \begin{bmatrix} \delta_1 & \\ & \delta_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix} + \Delta \tag{3.36}$$

where $\Delta = \sum_{3 \le i \le n} \mathbf{u}_i \delta_i \mathbf{v}_i^T$ is defined as the residual matrix. Note that the left singular vectors are unitary, i.e. $\mathbf{u}_i^T\mathbf{u}_j = 0$ if $i \neq j$, then

$$\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} = \mathbf{u}_1\mathbf{u}_1^T\mathbf{u}_1\delta_1\mathbf{v}_1^T + \mathbf{u}_1\mathbf{u}_1^T\mathbf{u}_2\delta_2\mathbf{v}_2^T + \mathbf{u}_1\mathbf{u}_1^T\Delta = \mathbf{u}_1\delta_1\mathbf{v}_1^T \tag{3.37}$$

Similarly, for $\mathbf{u}_2\mathbf{u}_2^T\mathbf{A}$, we have $\mathbf{u}_2\mathbf{u}_2^T\mathbf{A} = \mathbf{u}_2\delta_2\mathbf{v}_2^T$. Combining it with Equation (3.37), we have that

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = 2\mathbf{u}_1\mathbf{u}_1^T\mathbf{A} - \mathbf{u}_2\mathbf{u}_2^T\mathbf{A} = 2\mathbf{u}_1\delta_1\mathbf{v}_1^T - \mathbf{u}_2\delta_2\mathbf{v}_2^T = 2[\mathbf{u}_1\ \mathbf{u}_2] \begin{bmatrix} \delta_1 & \\ & -\delta_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix} \tag{3.38}$$

which is equivalent to truncated SVD on the adjacency matrix $\mathbf{A}$ after *reversing* the second largest singular value δ_2. QED.

To be specific, based on Lemma 3.6, for each edge $\mathbf{A}[i,j]$, we compute the corresponding edge weight in the derivative network $\mathbf{B}$ as $\mathbf{B}[i,j] = 2\delta_1\mathbf{u}_1[i]\mathbf{v}_1[j] - 2\delta_2\mathbf{u}_2[i]\mathbf{v}_2[j]$. Then the derivative network $\mathbf{B}$ can be easily computed by applying Equation (3.29) with a linear complexity in both time and space.

Lemma 3.7. (Time and space complexities of N2N for HITS). It takes $O(m+n)$ in time and $O(m+n)$ in space to generate the derivative network $\mathbf{B}$ for HITS on the input network $\mathbf{A}$, where n is the number of nodes and m is the number of edges.

Proof. It takes $O(r(m+n) + r^2 n)$ time complexity to perform the rank-r truncated SVD. Here, we strictly let $r = 2$, which reveals a $O(m+n)$ time complexity. Computing the partial derivatives of all edges takes $O(m)$ time in total. And it takes $O(m)$ time to calculate

the influence of all edges and generate the derivative network $\mathbf{B}$. Thus, the overall time complexity is $O(m+n)$. For space complexity, it takes $O(m)$ space complexity to save the adjacency matrix $\mathbf{A}$ and the derivative network $\mathbf{B}$ in sparse format. It also takes $O(n)$ space to save the rank-2 SVD results. Therefore, it has $O(m+n)$ space complexity. QED.

Instantiation #2: Spectral clustering. Here, we discuss how to construct the derivative network for spectral clustering, including how to choose the scalar function $f(\cdot)$, how to calculate the derivative network, as well as how to scale up the computation.

A – Choice of $f(\cdot)$ function. Here, we have an undirected network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with $\mathbf{A}$ being the adjacency matrix. We have the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$. Spectral clustering aims to find a matrix $\mathbf{U}$ by solving the optimization problem in Equation (3.26), which maximizes the intra-cluster connectivity while minimizing the inter-cluster connectivity. Naturally, we have $\mathcal{Y}^* = \mathbf{U}$ and define $f(\mathcal{Y}^*) = \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L}\mathbf{U}\right)$, which aligns with the objective function of the spectral clustering.

B – Calculating the derivative network $\mathbf{B}$. With our choice of $f(\cdot)$ function shown above, key steps to calculating the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ are summarized in Lemma 3.8.

Lemma 3.8. (N2N for spectral clustering). For a given input undirected network with adjacency matrix $\mathbf{A}$, the partial derivative of the sum of the r smallest eigenvalues with respect to the adjacency matrix is $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = \mathrm{diag}\left(\mathbf{U}\mathbf{U}^T\right)\mathbf{1}_{n\times n} - \mathbf{U}\mathbf{U}^T$, where columns in $\mathbf{U}$ are the associated eigenvectors, and $\mathbf{1}_{n\times n}$ is an $n \times n$ matrix with all entries equal to 1.

Proof. Since $\mathbf{U}$ is the eigenvectors associated with the r smallest eigenvalues, we let λ_i be the i-th smallest eigenvalue and re-write the objective as $f(\mathcal{Y}^*) = \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L}\mathbf{U}\right) = \sum_{i=1}^{r}\lambda_i$. We first apply chain rule to get the influence of each edge (i,j) as

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}[i,j]} = \mathrm{Tr}[(\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}})^T \frac{\partial \mathbf{L}}{\partial \mathbf{A}[i,j]}] \tag{3.39}$$

To calculate $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}}$, we have the following

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}[i,j]} = \sum_{l=1}^{r}\left(\frac{\partial \lambda_l}{\partial \mathbf{L}[i,j]}\right) = \sum_{l=1}^{r}\mathbf{U}[:,l]^T\mathbf{S}^{ij}\mathbf{U}[:,l] \tag{3.40}$$

where $\mathbf{S}^{ij}$ is a single-entry matrix with 1 at the i-th row and the j-th column and 0 elsewhere. With this, the matrix form solution of $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}}$ can be written as

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{L}} = \left[\sum_{l=1}^{r}\mathbf{U}[:,l]^T\mathbf{S}^{ij}\mathbf{U}[:,l]\right]_{1\le i\le n, 1\le j\le n} = \mathbf{U}\mathbf{U}^T \tag{3.41}$$

Since $\mathbf{L} = \mathbf{D} - \mathbf{A}$, we can calculate $\frac{\partial \mathbf{L}}{\partial \mathbf{A}[i,j]}$ as $\frac{\partial \mathbf{L}}{\partial \mathbf{A}[i,j]} = \mathbf{S}^{ii} - \mathbf{S}^{ij}$.

Putting everything together, we have the partial derivative of $f(\mathcal{Y}^*)$ with respect to $\mathbf{A}$ as follows

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = \operatorname{diag}\left(\mathbf{U}\mathbf{U}^T\right)\mathbf{1}_{n\times n} - \mathbf{U}\mathbf{U}^T \tag{3.42}$$

where $\mathbf{1}_{n\times n}$ is an $n \times n$ matrix with all entries equal to 1. QED.

Then we can apply Equation (3.29) to calculate its corresponding derivative network $\mathbf{B}$.

C – Scale-up computation. Major computational challenges in Equation (3.42) lie in the matrix multiplication of $\operatorname{diag}\left(\mathbf{U}\mathbf{U}^T\right)\mathbf{1}_{n\times n}$ and $\mathbf{U}\mathbf{U}^T$, which, if computed in a straightforward way, would require $O(n^3)$ complexity in time and $O(n^2)$ in space. We address the computational challenges by exploring the low-rank structure of $\operatorname{diag}\left(\mathbf{U}\mathbf{U}^T\right)\mathbf{1}_{n\times n} - \mathbf{U}\mathbf{U}^T$.

Let us denote the i-th row of $\mathbf{U}$ by $\mathbf{u}_i^T$, use $\mathbf{1}_{n\times 1}$ to denote the $n \times 1$ column vector filled by $1s$, and use $\mathbf{1}_{1\times n}$ to denote the $1 \times n$ row vector filled by $1s$. Then $\operatorname{diag}(\mathbf{U}\mathbf{U}^T)\mathbf{1}_{n\times n}$ can be re-written as follows

$$\operatorname{diag}(\mathbf{U}\mathbf{U}^T)\mathbf{1}_{n\times n} = \operatorname{diag}(\mathbf{U}\mathbf{U}^T)\mathbf{1}_{n\times 1}\mathbf{1}_{1\times n} = \begin{bmatrix} \mathbf{u}_1^T\mathbf{u}_1 \\ \mathbf{u}_2^T\mathbf{u}_2 \\ \cdots \\ \mathbf{u}_n^T\mathbf{u}_n \end{bmatrix} \mathbf{1}_{1\times n} \tag{3.43}$$

such that the column vector is of size $n \times 1$, and $\mathbf{1}_{1\times n}$ is a $1 \times n$ row vector. Then $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ can be represented in the following low-rank form

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}} = \begin{bmatrix} \mathbf{u}_1^T\mathbf{u}_1 \\ \mathbf{u}_2^T\mathbf{u}_2 \\ \cdots \\ \mathbf{u}_n^T\mathbf{u}_n \end{bmatrix} \mathbf{1}_{1\times n} - \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \cdots \\ \mathbf{u}_n^T \end{bmatrix} [\mathbf{u}_1\ \mathbf{u}_2\ \ldots\ \mathbf{u}_n] \tag{3.44}$$

With the low-rank structure, to get the partial derivative of edge (i, j) (i.e., element at the i-th row and the j-th column in $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$), we can simply calculate it as $\mathbf{u}_i^T\mathbf{u}_i - \mathbf{u}_i^T\mathbf{u}_j$. Then the influence of an edge (i, j) can be calculated as $\mathbf{B}[i, j] = \mathbf{u}_i^T\mathbf{u}_i + \mathbf{u}_j^T\mathbf{u}_j - \mathbf{u}_i^T\mathbf{u}_j - \mathbf{u}_j^T\mathbf{u}_i$, which takes $O(r)$ in time. In this way, we achieve linear time and space complexities with respect to the input network size, as stated in the following lemma.

Lemma 3.9. (Time and space complexities of N2N for spectral clustering). It takes $O(r(m+n) + r^2 n)$ complexity in time and $O(rn + m)$ in space to generate the derivative network $\mathbf{B}$ for the task of spectral clustering, where n and m are the numbers of nodes and edges of the input network respectively, and r is the number of clusters.

Proof. It takes $O(n)$ time to get the Laplacian matrix $\mathbf{L}$ and $O(r(m+n)+r^2n)$ to perform a rank-r eigen-decomposition on $\mathbf{L}$. Precomputing $\mathbf{u}_i^T\mathbf{u}_i$ for $1 \leq i \leq n$ in Equation (3.44) takes $O(rn)$ time complexity. Then it takes $O(rm)$ time to calculate the partial derivatives and the influence of all edges in the derivative network. Thus, the overall time complexity to calculate the derivative network $\mathbf{B}$ is $O(r(m+n)+r^2n)$. Regarding the space complexity, it takes $O(m)$ space to save the adjacency matrix $\mathbf{A}$ and the derivative network $\mathbf{B}$ in sparse format. Also, it takes $O(n)$ space to store the low-rank form of $\operatorname{diag}\left(\mathbf{U}\mathbf{U}^T\right)\mathbf{1}$ and $O(rn)$ space to save the matrix $\mathbf{U}$. Hence, the space complexity is $O(rn+m)$. QED.

Instantiation #3: Matrix completion. Now we discuss how to construct the derivative network for matrix factorization-based completion. The following discussions include how to choose the scalar function $f(\cdot)$, how to calculate the derivative network, as well as how to scale up the computation.

A – Choice of $f(\cdot)$ function. For matrix factorization-based completion, it solves the optimization problem in Equation (3.27) to find two low-rank matrices $\mathbf{U}$ and $\mathbf{V}$ with r latent factors. Here, the input is a user-item bipartite network $\mathbf{A}$ with n_1 users, n_2 items, and m observations, where $\mathbf{A}[i,j]$ is the rating of the j-th item made by the i-th user. Since $\hat{\mathbf{A}} = \mathbf{U}\mathbf{V}^T$ is often used to complete/infer the missing entries in the observed rating matrix $\mathbf{A}$, we choose $f(\mathcal{Y}^*) = \|\mathbf{U}\mathbf{V}^T\|_F^2$.

B – Calculating the derivative network $\mathbf{B}$. Since the bipartite network $\mathbf{A}$ is often represented as an asymmetric $n_1 \times n_2$ matrix, we have $\mathbf{B} = \frac{df(\mathcal{Y}^*)}{d\mathbf{A}} = \frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}}$ by Equation (3.29). Similar to the previous two instantiations, we aim to get the representation of $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}[i,j]}$ and then write out the matrix form solution. We first denote $\mathbf{X} = \mathbf{U}\mathbf{V}^T$ and then apply chain rule to get

$$\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{A}[i,j]} = \sum_{l=1}^{n_1}\sum_{t=1}^{n_2} \frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}[l,t]} \frac{\partial \mathbf{X}[l,t]}{\partial \mathbf{A}[i,j]} \tag{3.45}$$

However, it is non-trivial to calculate Equation (3.45). We present an accurate and efficient solution to calculate $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}[l,t]}$ and $\frac{\partial \mathbf{X}[l,t]}{\partial \mathbf{A}[i,j]}$ in Lemma 3.10.

Lemma 3.10. (N2N for matrix completion). For a given bipartite network $\mathbf{A}$, with $\mathbf{U}$ and $\mathbf{V}$ being the optimal model output of Equation (3.27), the influence of $[i,j]$ rating is $\mathbb{I}[i,j] = 2\mathbf{U}[i,:]\mathbf{V}^T\mathbf{V}\mathbf{C}_i^{-1}\mathbf{V}[j,:]^T + 2\mathbf{V}[j,:]\mathbf{U}^T\mathbf{U}\mathbf{D}_j^{-1}\mathbf{U}[i,:]^T$, where $\mathbf{C}_i = \lambda_u\mathbf{I} + \sum_{k\in\Omega_i}\mathbf{V}[k,:]^T\mathbf{V}[k,:]$, $\mathbf{D}_j = \lambda_v\mathbf{I} + \sum_{k\in\Omega_j}\mathbf{U}[k,:]^T\mathbf{U}[k,:]$, and Ω_i and Ω_j are sets of indices for non-zero entries of user i and item j, respectively.

Proof. We solve the two terms in Equation (3.45) one by one. First, we show how to compute $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}[l,t]}$. Recall that we define $f(\mathcal{Y}^*) = \|\mathbf{U}\mathbf{V}^T\|_F^2$ and $\mathbf{X} = \mathbf{U}\mathbf{V}^T$. By the derivative of matrix

norm, we have $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}} = 2\mathbf{X} = 2\mathbf{U}\mathbf{V}^T$. Thus, the partial derivative $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}[l,t]}$ can be denoted as

$$\frac{\partial f\left(\mathcal{Y}^*\right)}{\partial \mathbf{X}\left[l,t\right]} = 2\mathbf{X}\left[l,t\right] \tag{3.46}$$

This completes the calculation of the first term $\frac{\partial f(\mathcal{Y}^*)}{\partial \mathbf{X}[l,t]}$ in Equation (3.45).

Next, we show how to compute $\frac{\partial \mathbf{X}[l,t]}{\partial \mathbf{A}[i,j]}$. Since $\mathbf{X} = \mathbf{U}\mathbf{V}^T$, we have $\mathbf{X}\left[l,t\right] = \mathbf{U}\left[l,:\right]\mathbf{V}\left[t,:\right]^T$. Then the second term in Equation (3.45) $\frac{\partial \mathbf{X}[l,t]}{\partial \mathbf{A}[i,j]}$ can be re-written as $\frac{\partial \mathbf{X}[l,t]}{\partial \mathbf{A}[i,j]} = \frac{\partial \mathbf{U}[l,:]}{\partial \mathbf{A}[i,j]}\mathbf{V}\left[t,:\right]^T + \mathbf{U}\left[l,:\right](\frac{\partial \mathbf{V}[t,:]}{\partial \mathbf{A}[i,j]})^T$. However, it is hard to directly calculate $\frac{\partial \mathbf{U}[l,:]}{\partial \mathbf{A}[i,j]}$ and $\frac{\partial \mathbf{V}[t,:]}{\partial \mathbf{A}[i,j]}$, since there is no straightforward closed-form solution for $\mathbf{U}$ and $\mathbf{V}$ with respect to $\mathbf{A}\left[i,j\right]$. To solve this problem, we follow [78, 149] and consider the KKT conditions of Alternating Least Square (ALS) method, which are shown as follows

$$\begin{aligned}
\lambda_u \mathbf{U}\left[l,:\right] &= \sum_{k\in\Omega_l}(\mathbf{A}\left[l,k\right] - \mathbf{U}\left[l,:\right]\mathbf{V}\left[k,:\right]^T)\mathbf{V}\left[k,:\right] \\
\lambda_v \mathbf{V}\left[t,:\right] &= \sum_{k\in\Omega_t}(\mathbf{A}\left[k,t\right] - \mathbf{U}\left[k,:\right]\mathbf{V}\left[t,:\right]^T)\mathbf{U}\left[k,:\right]
\end{aligned} \tag{3.47}$$

where Ω_l and Ω_t are sets of indices for non-zero entries of user l and item t, respectively. Following the equations in Equation (3.47), we obtain the following partial derivatives,

$$\begin{aligned}
\frac{\partial \mathbf{U}\left[l,:\right]}{\partial \mathbf{A}\left[i,j\right]} &= \begin{cases} \mathbf{V}\left[j,:\right]\left[\lambda_u \mathbf{I} + \sum\limits_{k\in\Omega_l}\mathbf{V}\left[k,:\right]^T\mathbf{V}\left[k,:\right]\right]^{-1}, & \text{if } i = l \\ 0, & \text{otherwise} \end{cases} \\
\frac{\partial \mathbf{V}\left[t,:\right]}{\partial \mathbf{A}\left[i,j\right]} &= \begin{cases} \mathbf{U}\left[i,:\right]\left[\lambda_v \mathbf{I} + \sum\limits_{k\in\Omega_t}\mathbf{U}\left[k,:\right]^T\mathbf{U}\left[k,:\right]\right]^{-1}, & \text{if } j = t \\ 0, & \text{otherwise} \end{cases}
\end{aligned} \tag{3.48}$$

Since $\mathbf{C}_l = \lambda_u \mathbf{I} + \sum\limits_{k\in\Omega_l}\mathbf{V}\left[k,:\right]^T\mathbf{V}\left[k,:\right]$, $\mathbf{D}_t = \lambda_v \mathbf{I} + \sum\limits_{k\in\Omega_t}\mathbf{U}\left[k,:\right]^T\mathbf{U}\left[k,:\right]$, we have the following equation

$$\frac{\partial \mathbf{X}\left[l,t\right]}{\partial \mathbf{A}\left[i,j\right]} = \mathbf{V}\left[j,:\right]\mathbf{C}_i^{-1}\mathbf{V}\left[t,:\right]^T + \mathbf{U}\left[l,:\right]\mathbf{D}_j^{-1}\mathbf{U}\left[i,:\right]^T \tag{3.49}$$

Combining Equation (3.46) and Equation (3.49), we re-write Equation (3.45) as

$$\begin{aligned}\frac{\partial f\left(\mathcal{Y}^{*}\right)}{\partial \mathbf{A}\left[i,j\right]} &= 2\sum_{t=1}^{m}\mathbf{U}\left[i,:\right]\mathbf{V}\left[t,:\right]^{T}\frac{\partial \mathbf{U}\left[i,:\right]}{\partial \mathbf{A}\left[i,j\right]}\mathbf{V}\left[t,:\right]^{T} + 2\sum_{l=1}^{n}\mathbf{U}\left[l,:\right]\mathbf{V}\left[j,:\right]^{T}\mathbf{U}\left[l,:\right]\left(\frac{\partial \mathbf{V}\left[j,:\right]}{\partial \mathbf{A}\left[i,j\right]}\right)^{T} \\ &= 2\sum_{t=1}^{m}\mathbf{U}\left[i,:\right]\mathbf{V}\left[t,:\right]^{T}\mathbf{V}\left[j,:\right]\mathbf{C}_{i}^{-1}\mathbf{V}\left[t,:\right]^{T} + 2\sum_{l=1}^{n}\mathbf{U}\left[l,:\right]\mathbf{V}\left[j,:\right]^{T}\mathbf{U}\left[l,:\right]\mathbf{D}_{j}^{-1}\mathbf{U}\left[i,:\right]^{T} \\ &= 2\mathbf{U}\left[i,:\right]\mathbf{V}^{T}\mathbf{V}\mathbf{C}_{i}^{-1}\mathbf{V}\left[j,:\right]^{T} + 2\mathbf{V}\left[j,:\right]\mathbf{U}^{T}\mathbf{U}\mathbf{D}_{j}^{-1}\mathbf{U}\left[i,:\right]^{T}\end{aligned} \tag{3.50}$$

which completes the proof. QED.

C − Scale-up computation. Simply calculating the influence by Equation (3.50) will take $O\left(r^{2}\left(n_{1}+n_{2}\right)\right)$ time complexity for *each* rating. Therefore, the amortized time complexity will still be superlinear with respect to the network size ($O\left(r^{2}\left(n_{1}+n_{2}\right)m\right)$), which makes it not scalable to large networks. However, if we take one more look at Equation (3.50), we can find out that $\mathbf{C}_{i}^{-1}$ for each user i, $\mathbf{D}_{j}^{-1}$ for each item j, $\mathbf{U}^{T}\mathbf{U}$, and $\mathbf{V}^{T}\mathbf{V}$ are globally shared by all elements in $\mathbf{A}$. Thus, we can precompute them for all users and items once we have trained the optimal model output $\mathbf{U}$ and $\mathbf{V}$. With the help of precomputation, for each edge $\mathbf{A}\left[i,j\right]$ in the input bipartite network $\mathbf{A}$, we can calculate its influence as $\mathbf{B}\left[i,j\right] = 2\mathbf{U}\left[i,:\right]\mathbf{V}^{T}\mathbf{V}\mathbf{C}_{i}^{-1}\mathbf{V}\left[j,:\right]^{T} + 2\mathbf{V}\left[j,:\right]\mathbf{U}^{T}\mathbf{U}\mathbf{D}_{j}^{-1}\mathbf{U}\left[i,:\right]^{T}$ in $O\left(r\right)$ time. Since we only compute $\mathbf{B}\left[i,j\right]$ for observed rating $\mathbf{A}\left[i,j\right]$, we efficiently reduce the overall time complexity to be linear with respect to the input network size.

Lemma 3.11. (Time and space complexities of N2N for matrix completion.). The amortized time complexity to generate the derivative network $\mathbf{B}$ is $O\left(r^{3}\left(n_{1}+n_{2}\right)+r^{2}m\right)$, where n_{1} and n_{2} are numbers of users and items, r is the dimension of latent factors, and m is the total number of observed ratings. It takes $O\left(r^{2}\left(n_{1}+n_{2}\right)+m\right)$ complexity in space.

Proof. It takes $O\left(r^{3}\left(n_{1}+n_{2}\right)+r^{2}m\right)$ to train the model by ALS. It takes $O\left(r^{2}|\Omega_{i}|\right)$ and $O\left(r^{2}|\Omega_{j}|\right)$ to precompute $\mathbf{C}_{i}$ and $\mathbf{D}_{j}$, where $|\Omega_{i}|$ and $|\Omega_{j}|$ are the number of observed feedback for user i and item j, respectively. And it takes $O\left(r^{3}\right)$ complexity to inverse each $\mathbf{C}_{i}$ and $\mathbf{D}_{j}$ for all $1 \leq i \leq n_{1}$ and $1 \leq j \leq n_{2}$. Thus, the amortized complexity in the precomputation stage is $O\left(r^{3}\left(n_{1}+n_{2}\right)+r^{2}m\right)$. And the time complexities to precompute $\mathbf{U}^{T}\mathbf{U}$ and $\mathbf{V}^{T}\mathbf{V}$ are $O\left(r^{2}n\right)$ and $O\left(r^{2}m\right)$, respectively. Then, to calculate the influence for one element $\mathbf{A}\left[i,j\right]$, it takes $O\left(r^{2}\right)$ time complexity. Thus, the overall amortized time complexity to calculate the influence for all observed element in $\mathbf{A}$ is $O\left(r^{3}\left(n_{1}+n_{2}\right)+r^{2}m\right)$. Regarding space complexity, it takes $O\left(m\right)$ space to save the bipartite network $\mathbf{A}$. Each $\mathbf{C}_{i}$ and $\mathbf{D}_{j}$ for any $1 \leq i \leq n_{1}$ and $1 \leq j \leq n_{2}$ takes $O\left(r^{2}\right)$ space. And we also need to save the precomputed $\mathbf{U}^{T}\mathbf{U}$ and

$\mathbf{V}^T\mathbf{V}$, which also requires $O(r^2)$ space. Thus, it takes $O(r^2(n_1+n_2)+m)$ complexity in space. QED.

3.2.4 Experimental Evaluation

In this part, we perform empirical evaluations on the developed N2N framework. All experiments are designed to answer the following questions:

- *Effectiveness.* How effective is the developed N2N framework with respect to the corresponding mining task?
- *Efficiency.* How efficient and scalable is the developed N2N framework to generate the derivative network?

3.2.5 Experimental setting.

The detailed experimental settings to evaluate N2N are as follows.
Hardware and software specifications. All experiments are performed on a Windows PC with 3.8GHz Intel Core i7-9800X CPU and 64 GB RAM. All datasets are publicly available. The methods are programmed in Python 3.6. The source code of N2N can be downloaded at `https://jiank2.github.io/files/cikm19/n2n.zip`.
A – Dataset descriptions. We test our algorithms on a diverse set of real-world datasets. The statistics of these datasets are summarized in Table 3.6.

There are three types of datasets, including directed uni-partite networks ('D') for the ranking task, undirected uni-partite networks ('U') for the spectral clustering task, and bipartite networks ('B') for the matrix completion task. Among them, the three largest ones (i.e., *patent*, *douban* and *ml-20m*) are used for scalability experiments. These datasets come from a variety of application domains, including social networks (*Social*), citation networks (*Cit*), collaboration networks (*CA*), physical infrastructure networks (*Infra*), and rating networks (*Rating*). The detailed descriptions of these datasets are as follows.

- *Social networks.* Here, nodes are users, and edges indicate social relationships. Among them, *gplus* [150] is a directed network of Google+ user-user links, which is a social network by Google. A directed edge indicates that one user has the other user in his/her circle. *epinions* [151] is a directed who-trust-whom network from consumer reviews website Epinions[7]. *hamster* [150] is a directed network of friendship among

[7] `http://www.epinions.com/`

Table 3.6: Statistics of datasets used to evaluate N2N.

Task	Domain	Type	Dataset	# Nodes (Users/Items)	# Edges
HITS	*Cit*	D	cit-hepth	27,770	352,807
		D	cit-dblp	12,590	49,759
		D	cora	23,166	91,500
		D	patent	3,774,768	16,518,948
	Social	D	gplus	23,628	39,242
		D	epinions	75,879	508,837
	Infra	D	gnutella	8,114	26,013
Spectral clustering	*CA*	U	astroph	18,772	198,110
		U	condmat	23,133	93,497
		U	grqc	5,242	14,496
		U	ca-hepph	12,008	118,521
	Social	U	hamster	1,858	12,534
		U	douban	154,908	327,162
	Infra	U	twin	14,274	20,573
Matrix completion	*Rating*	B	lastfm	1,892/17,632	92,834
		B	delicious	1,867/69,223	437,593
		B	movielens	610/9,724	100,836
		B	ml-20m	138,493/26,744	20,000,264

(In Type, D: directed; U: undirected; B: bipartite.)

users of the website `hamsterster.com`. *douban* [150] is the social network of a Chinese online recommendation site Douban[8].

- *Citation networks.* Here, each node is a research paper. If a paper i cites paper j, there is a directed edge from node i to node j. *cit-hepph* [151] is an ArXiv HEP-PH (High Energy Physics – Phenomenology) citation network. The data covers papers from January 1993 to April 2003. *cit-dblp* [150] is a directed network of citation data on DBLP[9], a database of computer science bibliography. *cora* [150] is the CORA citation network. *patent* [151] is a directed network of all the utility patents granted from 1963 to 1999.

- *Collaboration networks.* Here, nodes are researchers, and two individuals are connected if they have collaborated together. We use four collaboration networks in the field of Physics from arXiv preprint archive[10]: Astro Physics (*astroph*), Condense Matter Physics (*condmat*), General Relativity and Quantum Cosmology (*grqc*), and High Energy Physics – Phenomenology (*ca-hepph*).

[8] `https://www.douban.com`
[9] `http://dblp.uni-trier.de/`
[10] `https://arxiv.org/`

- *Physical infrastructure networks.* This domain refers to the networks of physical infrastructure entities. Nodes correspond to physical infrastructure, and edges are connections. *gnutella* [151] is a snapshots of Gnutella peer-to-peer file sharing network collected on August 9, 2002. *twin* [150] is an undirected network of cities connected by sister city relationships. The dataset is extracted from WikiData[11].

- *Rating networks.* It is a collection of bipartite networks that consist of feedback given to items by users, weighted by a rating value. Four different rating networks are used. Among them, *lastfm* is extracted from the music streaming service Last.fm[12]. If a user i listened to a song by an artist j, its corresponding feedback $\mathbf{A}[i,j] = 1$, otherwise it is 0. *delicious* is extracted from the social bookmark sharing service website Delicious[13]. If a user i bookmarked a particular URL j, the feedback $\mathbf{A}[i,j] = 1$, otherwise it is 0. *movielens* and *ml-20m* are two rating networks of users to movies provided by GroupLens Research[14]. An edge between a user and a movie represents a rating of the movie by the user. Each rating ranges from 0.5 to 5.0.

C – Baseline methods. We compare N2N (Algorithm 3.4) with several baseline methods. We briefly summarize the baseline methods as follows.

- *Top Degrees (Degree).* We define the degree score of an edge (u, v) as follows.

$$\deg(u,v) = \begin{cases} \deg(u) + \deg(v), & \text{if undirected} \\ (\mathrm{d}(u) + \deg(v)) \times \deg(u), & \text{if directed} \end{cases} \tag{3.51}$$

  where $\deg(u)$ represents the degree of node u.

- *Top Eigenvector Centrality (EigenCentrality).* We define the eigenvector centrality score of an edge (u, v) as follows.

$$\mathrm{eig}(u,v) = \begin{cases} \mathrm{eig}(u) + \mathrm{eig}(v), & \text{if undirected} \\ (\mathrm{eig}(u) + \mathrm{eig}(v)) \times \mathrm{eig}(u), & \text{if directed} \end{cases} \tag{3.52}$$

  where $\mathrm{eig}(u)$ is the eigenvector centrality score of node u.

[11] `https://www.wikidata.org`
[12] `https://www.last.fm`
[13] `https://del.icio.us`
[14] `https://grouplens.org/datasets/movielens/`

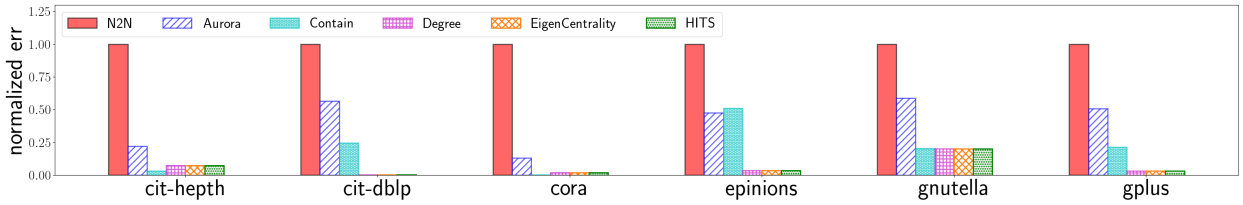


Figure 3.10: HITS ranking attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

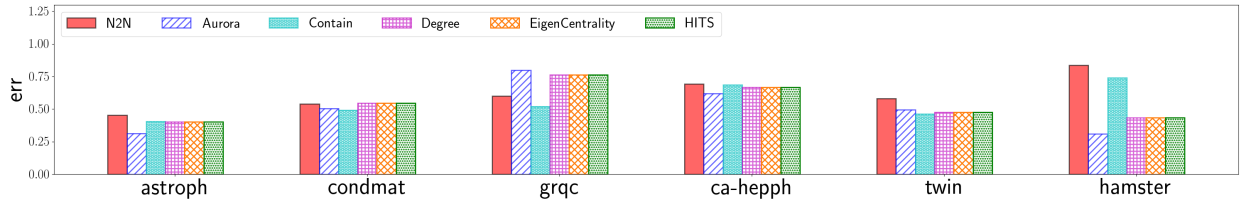


Figure 3.11: Spectral clustering attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

- *HITS.* We define HITS score of an edge (u, v) as follows.

$$\text{HITS}\,(u, v) = \text{hub}\,(u) + \text{hub}\,(v) + \text{auth}\,(u) + \text{auth}\,(v) \tag{3.53}$$

  where $\text{hub}\,(u)$ and $\text{auth}\,(u)$ represent the hub score and authority score of node u, respectively.

- *Contain.* Contain [152] is an algorithm to optimize the network connectivity. It iteratively selects a network element (e.g., a node or an edge) with the highest impact score on a user-defined connectivity measurement.

- *Aurora.* Aurora [147, 148] is an algorithm for PageRank auditing problem. It iteratively selects a network element (e.g., a node, an edge, or a subgraph) with the highest influence on the PageRank ranking vector of a network.

Algorithm 3.5 describes the procedures to select edges for our developed N2N method and baseline methods (including top degrees, top eigenvector centrality and HITS). The edge scoring function are defined as follows: $\text{C}\,(u, v) = \mathbb{I}\,(u, v)$ for N2N, $\text{C}\,(u, v) = \deg\,(u, v)$ for top degrees, $\text{C}\,(u, v) = \text{eig}\,(u, v)$ for top eigenvector centrality, and $C(u, v) = \text{HITS}\,(u, v)$ for HITS.

D – Evaluation metrics. Generally speaking, we want to measure how the mining results would change if we perturb the network elements (e.g., nodes, edges) for each mining task (i.e., HITS, spectral clustering, matrix completion).

Figure 3.12: Matrix completion attacking results. Higher is better (i.e., more effective attacking). Best viewed in color.

Algorithm 3.5: Description of Baseline Methods in N2N

Input : the adjacency matrix $\mathbf{A}$, an edge scoring function $C(\cdot,\cdot)$, integer budget k.
Output : a set of k edges $\mathcal{S}$ with highest edge scores.

initialize $\mathcal{S} = \emptyset$;
let $\tilde{\mathbf{A}} = \mathbf{A}$;
while $|\mathcal{S}| \neq k$ **do**
    find edge $(u, v) = \underset{[u,v]\in\tilde{\mathbf{A}}}{\text{argmax}}\ \mathrm{C}(u, v)$;
    add edge (u, v) to $\mathcal{S}$;
    remove edge (u, v) from $\tilde{\mathbf{A}}$;
return $\mathcal{S}$;

More specifically, for HITS, we measure how the hub and authority rankings change in total if we perturb the set of network elements. We define the distortion error metric for HITS as

$$\text{err} = \|\mathbf{u} - \tilde{\mathbf{u}}\|_2 + \|\mathbf{v} - \tilde{\mathbf{v}}\|_2 \tag{3.54}$$

where $\mathbf{u}$ and $\mathbf{v}$ are the original hub and authority vectors, while $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{v}}$ are the hub and authority vectors after perturbation.

For spectral clustering, we measure changes in cluster assignments after we perturb the set of network elements. We use the normalized mutual information (NMI) to measure the agreement between two cluster assignments before and after the perturbation. We define the distortion error metric as

$$\text{err} = 1 - \text{NMI}\left(\mathcal{C}, \tilde{\mathcal{C}}\right) = 1 - \frac{2\text{MI}\left(\mathcal{C}, \tilde{\mathcal{C}}\right)}{H\left(\mathcal{C}\right) + H\left(\tilde{\mathcal{C}}\right)} \tag{3.55}$$

where $\mathcal{C}$ and $\tilde{\mathcal{C}}$ are the cluster assignments before and after perturbation, $\text{MI}\left(\mathcal{C}, \tilde{\mathcal{C}}\right)$ is the mutual information between $\mathcal{C}$ and $\tilde{\mathcal{C}}$, and $H\left(\mathcal{C}\right)$ is the entropy of assignment $\mathcal{C}$. Notice that

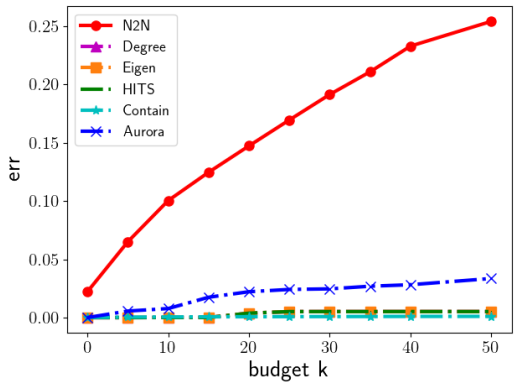


Figure 3.13: HITS ranking attacking results on different budget size. Higher is better (i.e., more effective attacking). Best viewed in color.

a larger err means a bigger difference between the clustering assignment before and after perturbation, which implies more effective attacking on clustering.

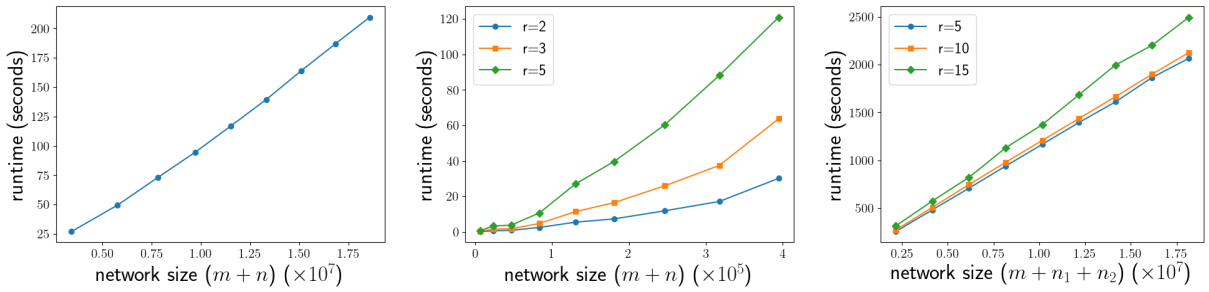
Regarding matrix completion, we measure how the model prediction changes after we perturb the set of network elements. We measure its change as follows,

$$\text{RMSE} = \sqrt{\sum_{[i,j]\in\Omega^C} \left(\hat{\mathbf{A}}\left[i,j\right] - \tilde{\mathbf{A}}\left[i,j\right]\right)^2 / \left|\Omega^C\right|} \tag{3.56}$$

where $\hat{\mathbf{A}}\left[i,j\right] = \mathbf{U}\left[i,:\right]\mathbf{V}\left[j,:\right]^T$ is the prediction made by the original model output, $\tilde{\mathbf{A}}\left[i,j\right] = \tilde{\mathbf{U}}\left[i,:\right]\tilde{\mathbf{V}}\left[j,:\right]^T$ is the prediction made by the model after perturbation, and $\left|\Omega^C\right|$ is the cardinality of the complementary set of Ω. Notice that a larger RMSE means a bigger difference in predicting a user's preference, which implies more effective attacking on the recommender system.

E – Detailed parameter settings. Regarding detailed parameter settings in the experimental evaluations, we set the budget size $k = 50$ for ranking and spectral clustering, and $k = 10$ for matrix completion. For *Contain* method, we use the leading eigenvalue as the connectivity measurement and set its rank parameter to 80 (see details in [152]). For spectral clustering, the number of clusters is set to 5. Regarding matrix completion, we set the latent dimension $r = 10$, the regularization parameters $\lambda_u = 0.5$ and $\lambda_v = 0.5$, and the number of training iterations to 10. Regarding the random initialization of cluster centroids in spectral clustering and low-rank matrices $\mathbf{U}$, $\mathbf{V}$ in matrix completion, the random seed is uniformly selected from 1 to 50 for each dataset.

Effectiveness results. We perform quantitative effectiveness comparison with baseline methods. The experiments are designed to explore the potential of the developed N2N frame-



(a) HITS ranking. (b) Spectral clustering. (c) Matrix Completion.

Figure 3.14: The scalability of the developed N2N on different network mining tasks.

work in attacking the corresponding network mining task, i.e., the mining results could be distorted greatly by removing a set of influential network edges in the derivative network.

Quantitative comparison. The quantitative comparison results for HITS, spectral clustering, and matrix completion across 15 different datasets are shown from Figure 3.10 to Figure 3.12, respectively. From those figures, we can see that the developed N2N framework (the leftmost red solid bar in Figures 3.10, 3.11, and 3.12) consistently outperforms other baseline methods in all datasets, which indicates that the derivative network generated by our developed N2N framework can indeed effectively attacking the network mining tasks by a few edge deletion.

Effect of budget size k. We explore the power of N2N on different budget size. Note that we only show the results of HITS ranking here. We set the budget size k from 1 to 50. From the results shown in Figure 3.13, we can observe that our developed N2N method (red solid line in Figure 3.13) outperforms other baseline methods on different budget size k.

Efficiency results. We show the running time vs. the input network size for HITS, spectral clustering, and matrix completion in Figure 3.14. We can see that the developed N2N framework scales linearly with respect to the input network size $m+n$ in all three instantiations. These results are consistent with our complexity analysis in Lemmas 3.7, 3.9, and 3.11, which states that the derivative network $\mathbf{B}$ can be computed in linear time with respect to the number of edges m and the number of nodes n.

3.3 UNCERTAINTY QUANTIFICATION ON GRAPH CONVOLUTIONAL NETWORK

Graph Convolutional Network (GCN) has become a prevalent learning paradigm in many real-world applications, including financial fraud detection [153], drug discovery [154], and traffic prediction [155]. To date, the vast majority of existing works do not take into account the uncertainty of a GCN regarding its prediction, which is alarming especially in high-stake

scenarios. For example, in automated financial fraud detection, it is vital to let expert banker to take controls if a GCN-based detector is highly uncertain about its predictions in order to prevent wrong decisions on suspending banking account(s).

A well established study on uncertainty quantification of GCN could bring several crucial benefits. First, it is a cornerstone in trustworthy graph mining. Uncertainty quantification aims to understand to what extent the model is likely to be incorrect, thus providing natural remedy to questions like *how uncertain is a GCN in its own predictions?* Second, an accurate quantification of GCN uncertainty could potentially answer *how to improve GCN predictions by leveraging its uncertainty* in many graph mining tasks. For example, in active learning on graphs, nodes with high uncertainty could be selected as the most valuable node to query the oracle; in node classification, the node uncertainty could help calibrate the confidence of GCN predictions, thereby improving the overall classification accuracy.

Important as it could be, very few studies on uncertainty quantification of GCN exist, which mainly focuses on two different directions: Bayesian-based approaches and deterministic quantification-based approaches. Regarding Bayesian-based approaches [156, 157], they either drops edge(s) with certain sampling strategies or leverages random graph model (e.g., stochastic block model) to assign edge probabilities for training. However, these models fall short in explicitly quantifying the uncertainty on model predictions. Another type of methods, i.e., deterministic quantification-based approaches [158, 159, 160], directly quantifies uncertainty by parameterizing a Dirichlet distribution as prior to estimate the posterior distribution under a Bayesian framework. Nevertheless, it changes the training procedures of a graph neural network by introducing additional parameters (e.g., parameters for Dirichlet distribution) or additional architectures (e.g., teacher network) in order to precisely estimate the uncertainty.

To address the aforementioned limitations, we provide the first study on frequentist-based analysis of the GCN uncertainty, which we term as the JuryGCN problem. Building upon the general principle of jackknife (leave-one-out) resampling [161], the jackknife uncertainty of a node is defined as the width of confidence interval constructed by a jackknife estimator when leaving the corresponding node out. In order to estimate the GCN parameters without exhaustively re-training GCN, we leverage influence functions [134] to quantify the change in GCN parameters by infinitesimally upweighting the loss of a training node. Compared with existing works, our method brings several advantages. First, our method provides deterministic uncertainty quantification, which is not available in Bayesian-based approaches [156, 157]. Second, different from existing works on deterministic uncertainty quantification [158, 159, 160], our method does not introduce any additional parameters or components in the GCN architecture. Third, our method can provide *post-hoc* uncertainty

quantification. As long as the input graph and a GCN are provided, our method can *always* quantify node uncertainty without any epoch(s) of model training.

The major contributions of this work are summarized as follows.

- *Problem definition.* To our best knowledge, we provide the first frequentist-based analysis of GCN uncertainty and formally define the JuryGCN problem.
- *Algorithm and analysis.* We develop JuryGCN to quantify jackknife uncertainty on GCN. The key idea is to leverage a jackknife estimator to construct a leave-one-out predictive confidence interval for each node, where the leave-one-out predictions are estimated using the influence functions with respect to model parameters.
- *Experimental evaluations.* We demonstrate the effectiveness of JuryGCN through extensive experiments on real-world graphs in active learning on node classification and semi-supervised node classification.

3.3.1 Problem Definition

In this part, we first introduce preliminary knowledge on the Graph Convolutional Network (GCN), predictive uncertainty, and jackknife resampling. Then we formally define the problem of jackknife uncertainty quantification on GCN (JuryGCN).

In this work, unless otherwise specified, we use bold upper-case letters for matrices (e.g., $\mathbf{A}$), bold lower-case letters for vectors (e.g., $\mathbf{x}$), calligraphic letters for sets (e.g., $\mathcal{G}$), and fraktur font for high-dimensional tensors ($\mathfrak{H}$). We use supercript ^{T} for matrix transpose and superscript $^{-1}$ for matrix inversion, i.e., $\mathbf{A}^T$ and $\mathbf{A}^{-1}$ are the transpose and inverse of $\mathbf{A}$, respectively. We use conventions similar to Numpy in Python for indexing. For example, $\mathbf{A}[i, j]$ represents the entry of $\mathbf{A}$ at the i-th row and j-th column; $\mathbf{A}[i, :]$ and $\mathbf{A}[:, j]$ demonstrate the i-th row and j-th column of $\mathbf{A}$, respectively.

Preliminaries. We briefly review preliminary knowledge on graph convolutional network (GCN), uncertainty quantification, and jackknife resampling.

A – Graph Convolutional Network (GCN). Let $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ denote a graph whose node set is $\mathcal{V}$, adjacency matrix is $\mathbf{A}$, and node feature matrix is $\mathbf{X}$. For the l-th hidden layer in an L-layer GCN, we assume $\mathbf{E}^{(l)}$ is the output node embeddings (where $\mathbf{E}^{(0)} = \mathbf{X}$), and $\mathbf{W}^{(l)}$ is the weight matrix. Mathematically, the graph convolution at the l-th hidden layer can be represented by $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$, where σ is the activation, and $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the degree matrix of $(\mathbf{A} + \mathbf{I})$.

B – Uncertainty quantification is one of the cornerstones in safe-critical applications. It provides accurate quantification on how confident a mining model is towards its predictions.

In general, uncertainty can be divided into two types: *aleatoric* uncertainty and *epistemic* uncertainty [162]. Aleatoric uncertainty (or data uncertainty) refers to the variability in mining results due to the inherent randomness in input data, which is irreducible due to complexity of input data (e.g., noise); whereas epistemic uncertainty (or model uncertainty) measures how well the mining model fits the training data due to the lack of knowledge on the optimal model parameters, which is reducible by increasing the size of training data.

C – Jackknife resampling is a classic method to estimate the bias and variance of a population [163]. It often relies on a jackknife estimator, which is built by leaving out an observation from the entire population (i.e., leave-one-out) and evaluating the error of the model re-trained on the held-out population. Suppose we have (1) a set of n data points $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|i = 1, \ldots, n\}$, (2) a test point $(\mathbf{x}_{\text{test}}, y_{\text{test}})$, (3) a mining model $f_\theta()$ parameterized by θ (e.g., a neural network) where $f_\theta(\mathbf{x})$ is the prediction of input feature $\mathbf{x}$, and (4) a target coverage level $(1-\alpha)$ such that the label y is covered by the predictive confidence interval with probability $(1-\alpha)$. Mathematically, the confidence interval constructed by the naive jackknife [164] is upper bounded by $\mathbb{C}^+(\mathbf{x}_{\text{test}}) = Q_{1-\alpha}(\mathcal{R}^+)$ and lower bounded by $\mathbb{C}^-(\mathbf{x}_{\text{test}}) = Q_\alpha(\mathcal{R}^-)$, where Q_α finds the α quantile of a set and $\mathcal{R}^\gamma = \{f_\theta(\mathbf{x}_{\text{test}}) + \gamma \cdot |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})| | i = 1, \ldots, n\}$ for $\gamma \in \{-, +\}$ and $|y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ is the error residual of the re-trained model on the dataset $\mathcal{D} \setminus \{(\mathbf{x}_i, y_i)\}$ (i.e., parameterized by θ_{-i}).[15] Hence, $\mathcal{R}^+$ and $\mathcal{R}^-$ represent the sets of upper and lower uncertainty bounds on the original model prediction (i.e., $f_\theta(\mathbf{x}_{\text{test}})$). Furthermore, jackknife+ [165] constructs the predictive confidence interval for exchangeable data as

$$\mathbb{C}^+(\mathbf{x}_{\text{test}}) = Q_{1-\alpha}\left(\mathcal{P}^+\right) \qquad \mathbb{C}^-(\mathbf{x}_{\text{test}}) = Q_\alpha\left(\mathcal{P}^-\right) \tag{3.57}$$

where $\mathcal{P}^\gamma$ for $\gamma \in \{-, +\}$ is defined as $\mathcal{P}^\gamma = \left\{f_{\theta_{-i}}(\mathbf{x}_{\text{test}}) + \gamma \cdot |y_i - f_{\theta_{-i}}(\mathbf{x}_i)| | i = 1, \ldots, n\right\}$. Similarly, $\mathcal{P}^-$ and $\mathcal{P}^+$ represent the sets of the lower and upper uncertainty bounds of the leave-one-out prediction $f_{\theta_{-i}}(\mathbf{x}_{\text{test}})$, respectively. With the assumption on data exchangeability, it yields a $(1-2\alpha)$ coverage rate theoretically.

Problem definition. Existing works on deterministic uncertainty quantification of a graph neural network (GNN) mainly rely on changing the training procedures of a vanilla GNN (i.e., graph neural network without consideration of uncertainty) [159, 160]. As such, given a well-trained GNN, it requires epoch(s) of re-training to quantify its uncertainty. Nevertheless, it would cost a lot of computational resources to re-train it, especially when the model has already been deployed in an operational environment. Additionally, it remains a challenging

[15] We use γ to represent the symbol before the leave-one-out error, i.e., $f_\theta(\mathbf{x}_{\text{test}}) - |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ when $\gamma = -$, or $f_\theta(\mathbf{x}_{\text{test}}) + |y_{\text{test}} - f_{\theta_{-i}}(\mathbf{x}_{\text{test}})|$ otherwise.

problem to further comprehend the predictive results of GNN from the perspective of uncertainty and to answer the following question: *to what extent the GNN model is confident of the current prediction?* Therefore, it is essential to investigate the uncertainty quantification in a post-hoc manner, i.e., quantifying uncertainty without further (re-)training on the model.

Regarding post-hoc uncertainty quantification for IID (i.e., non-graph) data, Alaa and van der Schaar [166] propose a frequentist-based method inspired by jackknife resampling. It uses high-order influence functions to quantify the impact of a data point on the underlying neural network. Given that the parameters of a neural network are derived by learning with data points, high-order influence functions are capable of understanding how much a data point will affect the model parameters. Then the change in model parameters can be used to infer the uncertainty of the corresponding data point on the neural network by a jackknife estimator [165]. Under mild assumption (such as the algorithmic stability assumption and the IID/exchangebility of data), the naive jackknife estimator [164] and its variants [165] bear strong theoretical guarantee in terms of the coverage such that the confidence interval will cover the true model parameters with a high probability.

Building upon the jackknife resampling [161] and the general principle outlined in [166], we seek to bridge the gap between frequentist-based uncertainty quantification and graph neural networks. To be specific, given an input graph and a GCN, we aim to estimate the uncertainty of a node as the impact of leaving out its loss when computing the overall loss. Formally, we define the problem of jackknife uncertainty quantification on GCN, which is referred to as JuryGCN problem.

Problem 3.3. JuryGCN: Jackknife uncertainty quantification on Graph Convolutional Network.

Given: (1) an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$, (2) an L-layer GCN with the set of weights Θ, and (3) a task-specific loss function $R(\mathcal{G}, \mathcal{Y}, \Theta)$ where $\mathcal{Y}$ is the set of node labels.
Find: an uncertainty score $\mathbb{U}_\Theta(u)$ for any node u in graph $\mathcal{G}$ with respect to the GCN parameters Θ and the task-specific loss function $R(\mathcal{G}, \mathcal{Y}, \Theta)$.

3.3.2 JuryGCN: Measure and Computation

In this part, we start by discussing the general strategy of quantifying jackknife uncertainty with influence functions and formally define the node jackknife uncertainty. After that, we present mathematical analysis on the influence function computation for GCN.

Jackknife uncertainty of GCN. In this work, we consider an L-layer GCN with ReLU activation for node-level tasks (e.g., node classification). We further assume that the nodes

of the input graph $\mathcal{G}$ are exchangeable data.[16]

We first observe that the loss function of many node-level tasks can often be decomposed into a set of node-specific subproblems. Mathematically, it can be written as

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}}\, R\left(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta\right) = \underset{\Theta}{\operatorname{argmin}} \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{v \in \mathcal{V}_{\text{train}}} r\left(v, \mathbf{y}_v, \Theta\right) \tag{3.58}$$

where $\mathcal{V}_{\text{train}} \subseteq \mathcal{V}$ is the set of training nodes, $|\mathcal{V}_{\text{train}}|$ is the number of training nodes, $\mathcal{Y}_{\text{train}} = \{\mathbf{y}_v | v \in \mathcal{V}_{\text{train}}\}$ is the set of ground-truth training labels, and $\mathbf{y}_v$ is the label of node v. In this case, the overall loss function $R\left(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta\right)$ is decomposed into several subproblems, each of which minimizes the node-specific loss function $r\left(v, \mathbf{y}_v, \Theta\right)$ for a node v. An example is the cross entropy with node-specific loss as $r\left(v, \mathbf{y}_v, \Theta\right) = -\sum_{i=1}^{c} \mathbf{y}_v\left[c\right] \log\left(\text{GCN}\left(v, \Theta\right)\left[c\right]\right)$, where c is the number of classes, and $\text{GCN}\left(v, \Theta\right)$ is the vector of predicted probabilities for each class using the GCN with parameters Θ. If we upweight the importance of optimizing the loss of a node i with some small constant ϵ, we have the following loss function.

$$\Theta^*_{\epsilon,i} = \underset{\Theta}{\operatorname{argmin}}\, \epsilon r\left(i, \mathbf{y}_i, \Theta\right) + \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{v \in \mathcal{V}_{\text{train}}} r\left(v, \mathbf{y}_v, \Theta\right) \tag{3.59}$$

Influence function is a powerful approach to evaluate the dependence of the estimator on the value of the data examples [134, 170]. In order to obtain $\Theta^*_{\epsilon,i}$ without re-training the GCN, we leverage the influence function [134], which is essentially the Taylor expansion over the model parameters.

$$\Theta^*_{\epsilon,i} \approx \Theta^* + \epsilon \mathbb{I}_{\Theta^*}\left(i\right) \tag{3.60}$$

where $\mathbb{I}_{\Theta^*}\left(i\right) = \frac{d\Theta^*_{\epsilon,i}}{d\epsilon}|_{\epsilon=0}$ is the influence function with respect to node i. The influence function $\mathbb{I}_{\Theta^*}\left(i\right)$ can be further computed using the classical result in [171] as

$$\mathbb{I}_{\Theta^*}\left(i\right) = \mathbf{H}^{-1}_{\Theta^*} \nabla_{\Theta} r\left(i, \mathbf{y}_i, \Theta^*\right) \tag{3.61}$$

where $\mathbf{H}_{\Theta^*} = \frac{1}{|\mathcal{V}_{\text{train}}|} \nabla^2_{\Theta} R\left(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta^*\right)$ is the Hessian matrix with respect to model parameters Θ^*. For a training node i, by setting $\epsilon = -\frac{1}{|\mathcal{V}_{\text{train}}|}$, Equation (3.60) efficiently estimates the leave-one-out (LOO) parameters $\Theta^*_{\epsilon,i}$ if leaving out the loss of node i. After that, by simply switching the original model parameters to $\Theta^*_{\epsilon,i}$, we estimate the leave-one-out error err_i of node i as follows.

$$err_i = \|\mathbf{y}_i - \text{GCN}\left(i, \Theta^*_{\epsilon,i}\right)\|_2 \tag{3.62}$$

[16] A sequence of random variables is exchangeable if and only if the joint distribution of the random variables remains unchanged regardless of their ordering in the sequence [167]. This assumption is commonly used in random graph models, e.g., Erdős-Rényi model [168], stochastic block model [169], etc.

where $\mathrm{GCN}\left(u, \Theta^*_{\epsilon,i}\right)$ represents the output of node u using the GCN with leave-one-out parameters $\Theta^*_{\epsilon,i}$.

With Equation (3.62), we use jackknife+ [165], which requires the data to be exchangeable instead of IID, to construct the confidence interval of node u. Mathematically, the lower bound $\mathbb{C}^-_{\Theta}(u)$ and upper bound $\mathbb{C}^+_{\Theta}(u)$ of the predictive confidence interval of node u are

$$\begin{aligned} \mathbb{C}^-_{\Theta^*}(u) &= Q_\alpha(\{\|\mathrm{GCN}\left(u, \Theta^*_{\epsilon,i}\right)\|_2 - err_i | \forall i \in \mathcal{V}_{\text{train}} \setminus \{u\}\}) \\ \mathbb{C}^+_{\Theta^*}(u) &= Q_{1-\alpha}\left(\left\{\|\mathrm{GCN}\left(u, \Theta^*_{\epsilon,i}\right)\|_2 + err_i | \forall i \in \mathcal{V}_{\text{train}} \setminus \{u\}\right\}\right) \end{aligned} \tag{3.63}$$

where Q_α and $Q_{1-\alpha}$ are the α and $(1-\alpha)$ quantile of a set. Since a wide confidence interval of node u means that the model is less confident with respect to node u, it implies that node u has high uncertainty. Following this intuition, the uncertainty of node u can be naturally quantified by the width of the corresponding confidence interval (Equation (3.63)). Since the uncertainty is quantified using the confidence interval constructed by a jackknife estimator, we term it as *jackknife uncertainty*, which is formally defined in Definition 3.3.

Definition 3.3. (Node jackknife uncertainty). Given an input graph $\mathcal{G}$ with node set $\mathcal{V}$, a set of training nodes $\mathcal{V}_{\text{train}} \subseteq \mathcal{V}$, and an L-layer GCN with parameters Θ, $\forall i \in \mathcal{V}_{\text{train}}$ and $\forall u \in \mathcal{V}$, we assume (1) the nodes are exchangeable, (2) the LOO parameters are denoted as $\Theta_{\epsilon,i}$, (3) the error is defined as Equation (3.62), and (4) the lower bound $\mathbb{C}^-_{\Theta}(u)$ and the upper bound $\mathbb{C}^+_{\Theta}(u)$ of predictive confidence interval are defined as Equation (3.63). Then the jackknife uncertainty of node u is

$$\mathbb{U}_{\Theta}(u) = \mathbb{C}^+_{\Theta}(u) - \mathbb{C}^-_{\Theta}(u) \tag{3.64}$$

We note that Alaa and van Der Schaar [166] leverage high-order influence functions to quantify the jackknife uncertainty for IID data. Though Equation (3.60) shares the same form as in [166] when the order is up to 1, our work bears three subtle differences. First, [166] views the model parameters as statistical functionals of data distribution and exploits von Mises expansion over the data distribution to estimate the LOO parameters,[17] which is fundamentally different from our Taylor expansion-based estimation. Specifically, von Mises expansion requires that the perturbed data distribution should be in a convex set of the original data distribution and all possible empirical distributions [172]. Since the node distribution of a graph is often unknown, the basic assumption of von Mises expansion might not hold on graph data. However, our definition relies on the Taylor expansion over model parameters which are often drawn independently from Gaussian distribution(s). Thus,

[17] A statistical functional is a map that maps a distribution to a real number.

our method is able to generalize on graphs. Second, [166] works for regression or binary classification tasks by default, whereas we target more general learning settings on graphs (e.g., multi-class node classification). Third, jackknife uncertainty is always able to quantify aleatoric uncertainty and epistemic uncertainty simultaneously on IID data. Nevertheless, as shown in Proposition 3.1, it requires additional assumption to quantify both types of uncertainty on GCN simultaneously for a node u.

Proposition 3.1. (Necessary condition of aleatoric and epistemic uncertainty quantification on GCN). Given an input graph $\mathcal{G}$ whose node set is $\mathcal{V}$, a node $u \in \mathcal{V}$, a set of training nodes $\mathcal{V}_{\text{train}}$, and an L-layer GCN, jackknife uncertainty quantifies the aleatoric uncertainty and the epistemic uncertainty as long as u is outside the L-hop neighborhood of an arbitrary training node $v \in \mathcal{V}_{\text{train}} \setminus \{u\}$.

Proof. For an arbitrary node $v \in \mathcal{V}$ in the graph, its receptive field after L graph convolution layers is the set of all neighbors within L hops. Then if the node u, whose uncertainty is going to be quantified, is not within the L-hop neighborhood of any training node $v \in \mathcal{V}_{\text{train}} \setminus \{u\}$, whether to leave out the loss of node u during training will have no impact on hidden node representations and final predictions of v, which is equivalent to the leave-one-out settings for IID data. QED.

Remark. For GCN, jackknife uncertainty cannot always measure the aleatoric uncertainty and epistemic uncertainty simultaneously. In fact, if a node u is one of the neighbors within L hops with respect to any training node $v \in \mathcal{V}_{\text{train}} \setminus \{u\}$, jackknife uncertainty only quantifies the epistemic uncertainty due to lack of knowledge on the loss of node u. In this case, jackknife uncertainty cannot quantify aleatoric uncertainty because leaving out the loss of node u does not necessarily remove node u in the graph. More specifically, the aleatoric uncertainty of node u can still be transferred to its neighbors through neighborhood aggregation in graph convolution.

The influence functions of GCN. In order to quantify jackknife uncertainty (Equation (3.64)), we need to compute the influence functions to estimate the leave-one-out parameters by Equation (3.60). Given a GCN with Θ being the set of model parameters, to compute the influence functions of node i (Equation (3.61)), we need to compute two key terms, including (1) first-order derivative $\nabla_{\Theta} r\left(i, \mathbf{y}_i, \Theta\right)$ and (2) second-order derivative $\mathbf{H}_{\Theta}$. We first give the following proposition for the computation of first-order derivative in Proposition 3.2.[18] Based on that, we present the main results for computing the second-order

[18]The method to compute the first-order derivative was first proposed in [18] for a different purpose, i.e., ensuring degree-related fairness in GCN.

derivative in Theorem 3.2. Finally, we show details of influence function computation in Algorithm 3.6.

Proposition 3.2. (First-order derivative of GCN [18]). Given an L-layer GCN whose parameters are Θ, an input graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$, a node i with its label $\mathbf{y}_i$, and a node-wise loss function $r(i, \mathbf{y}_i, \Theta)$ for node i, the first-order derivative of loss function $r(i, \mathbf{y}_i, \Theta)$ with respect to the parameters $\mathbf{W}^{(l)}$ in the l-th graph convolution layer is

$$\nabla_{\mathbf{W}^{(l)}} r(i, \mathbf{y}_i, \Theta) = \left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\right)^T \left(\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} \circ \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)\right) \tag{3.65}$$

where $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the degree matrix of $\mathbf{A}+\mathbf{I}$, σ' is the derivative of the activation function σ, $\circ$ is the element-wise product, and $\frac{\partial r(i,\mathbf{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}}$ can be iteratively calculated by

$$\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} = \hat{\mathbf{A}}^T \left(\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l-1)}} \circ \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l-1)}\right)\right)\left(\mathbf{W}^{(l-1)}\right)^T \tag{3.66}$$

Proof. To prove it, we derive the element-wise computation of $\nabla_{\mathbf{W}^{(l)}} r(i, \mathbf{y}_i, \Theta)$ and then write out the matrix form. We first apply the chain rule and get

$$\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{W}^{(l)}[a, b]} = \sum_{c=1}^{n}\sum_{d=1}^{h_l} \frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}[c, d]} \frac{\partial \mathbf{E}^{(l)}[c, d]}{\partial \mathbf{W}^{(l)}[a, b]} \tag{3.67}$$

where h_l is the hidden dimension of l-th layer. Regarding the computation of $\frac{\partial \mathbf{E}^{(l)}[c,d]}{\partial \mathbf{W}^{(l)}[a,b]}$, since $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$, we have

$$\begin{aligned} \frac{\partial \mathbf{E}^{(l)}[c, d]}{\partial \mathbf{W}^{(l)}[a, b]} &= \frac{\partial \sigma\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)[c, d]}{\partial\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)[c, d]} \frac{\partial\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)[c, d]}{\partial \mathbf{W}^{(l)}[a, b]} \\ &= \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)[c, d]\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\right)[c, a]\,\mathbf{I}[d, b] \end{aligned} \tag{3.68}$$

where σ' is the first-order derivative of the activation function σ. Combining Equations (3.67) and (3.68) together, we have the element-wise computation of $\nabla_{\mathbf{W}^{(l)}} r(i, \mathbf{y}_i, \Theta)$ as follows.

$$\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{W}^{(l)}[a, b]} = \left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\right)^T[a, :]\left(\frac{\partial r(i, \mathbf{y}_i, \Theta)}{\partial \mathbf{E}^{(l)}} \circ \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)\right)[:, b] \tag{3.69}$$

where $\circ$ is the element-wise product. Finally, we get Equation (3.65) by writing Equation (3.69) into matrix form.

Regarding the computation of $\frac{\partial r(i,\mathbf{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}}$, the key idea is to write out a recursive function with respect to $\frac{\partial r(i,\mathbf{y}_i,\Theta)}{\partial \mathbf{E}^{(l)}}$ based on the chain rule. More specifically, we first consider the element-wise computation and apply the chain rule as follows.

$$\frac{\partial r\left(i,\mathbf{y}_i,\Theta\right)}{\partial \mathbf{E}^{(l)}\left[c,d\right]} = \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}} \frac{\partial r\left(i,\mathbf{y}_i,\Theta\right)}{\partial \mathbf{E}^{(l+1)}\left[e,f\right]} \frac{\partial \mathbf{E}^{(l+1)}\left[e,f\right]}{\partial \mathbf{E}^{(l)}\left[c,d\right]} \tag{3.70}$$

We take derivative on both sides of $\mathbf{E}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)})$ and get

$$\frac{\partial \mathbf{E}^{(l+1)}\left[e,f\right]}{\partial \mathbf{E}^{(l)}\left[c,d\right]} = \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\right)\left[e,f\right]\hat{\mathbf{A}}\left[e,c\right]\mathbf{W}^{(l+1)}\left[d,f\right] \tag{3.71}$$

Combining Equations (3.70) and (3.71) together, we get the following element-wise first-order derivative.

$$\frac{\partial r\left(i,\mathbf{y}_i,\Theta\right)}{\partial \mathbf{E}^{(l)}\left[c,d\right]} = \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}} \hat{\mathbf{A}}^T\left[c,e\right] \cdot \left(\frac{\partial r\left(i,\mathbf{y}_i,\Theta\right)}{\partial \mathbf{E}^{(l+1)}}\sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\right)\right)\left[e,f\right] \cdot \left(\mathbf{W}^{(l+1)}\right)^T\left[f,d\right] \tag{3.72}$$

We complete the proof by writing Equation (3.72) into matrix form. QED.

Since the parameters are often represented as matrices in the hidden layers, the second-order derivative will be a 4-dimensional tensor (i.e., a Hessian tensor). Building upon the results in Proposition 3.2, we first present the computation of the Hessian tensor in Theorem 3.2. Then we discuss efficient computation of influence function, which is summarized in Algorithm 3.6.

Theorem 3.2. (The Hessian tensor of GCN). Following the settings of Proposition 3.2, denoting the overall loss $R(\mathcal{G},\mathcal{Y}_{\text{train}},\Theta)$ as R and σ_l' as $\sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l-1)}\mathbf{W}^{(l)}\right)$, the Hessian tensor $\mathfrak{H}_{l,i} = \frac{\partial^2 R}{\partial \mathbf{W}^{(l)}\partial \mathbf{W}^{(i)}}$ of R with respect to $\mathbf{W}^{(l)}$ and $\mathbf{W}^{(i)}$ has the following forms.

- **Case 1.** $i = l$, $\mathfrak{H}_{l,i} = 0$
- **Case 2.** $i = l-1$

$$\mathfrak{H}_{l,i}\left[:,:,c,d\right] = \left(\hat{\mathbf{A}}\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}\left[c,d\right]}\right)^T\left(\frac{\partial R}{\partial \mathbf{E}^{(l)}} \circ \sigma_l'\right) \tag{3.73}$$

where $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$ is the matrix whose entry at the a-th row and the b-th column is

$$\frac{\partial \mathbf{E}^{(l-1)}\left[a,b\right]}{\partial \mathbf{W}^{(l-1)}\left[c,d\right]} = \sigma_{l-1}'\left[a,b\right]\left(\hat{\mathbf{A}}\mathbf{E}^{(l-2)}\right)\left[a,c\right]\mathbf{I}\left[b,d\right] \tag{3.74}$$

- **Case 3.** $i < l-1$
  - Apply Equation (3.74) for the i-th hidden layer.
  - Forward to the $(l-1)$-th layer iteratively with

$$\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]} = \sigma'_{l-1} \circ \left(\hat{\mathbf{A}} \frac{\partial \mathbf{E}^{(l-2)}}{\partial \mathbf{W}^{(i)}[c,d]} \mathbf{W}^{(l-1)} \right) \tag{3.75}$$

  - Apply Equation (3.73).

- **Case 4.** $i = l+1$

$$\mathfrak{H}_{l,i}[:,:,c,d] = (\hat{\mathbf{A}}\mathbf{E}^{(l-1)})^T \left(\frac{\partial^2 R}{\partial \mathbf{E}^{(l)} \partial \mathbf{W}^{(i)}[c,d]} \circ \sigma'_l \right) \tag{3.76}$$

  where $\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b] \partial \mathbf{W}^{(l-1)}[c,d]} = \mathbf{I}[b,c] \left[\hat{\mathbf{A}}^T \left(\frac{\partial R}{\partial \mathbf{E}^{(l-1)}} \circ \sigma'_{l+1} \right) \right][a,d]$

- **Case 5.** $i > l+1$
  - Compute $\frac{\partial^2 R}{\partial \mathbf{E}^{(i-1)} \partial \mathbf{W}^{(i)}[c,d]}$ whose (a,b)-th entry has the form $\frac{\partial^2 R}{\partial \mathbf{E}^{(i-1)}[a,b] \partial \mathbf{W}^{(i)}[c,d]} = \mathbf{I}[b,c] \left(\hat{\mathbf{A}}^T \left(\frac{\partial R}{\partial \mathbf{E}^{(i)}} \circ \sigma'_i \right) \right)[a,d]$
  - Backward to $(l-1)$-th layer iteratively with

$$\frac{\partial^2 R}{\partial \mathbf{E}^{(l)} \partial \mathbf{W}^{(i)}[c,d]} = \hat{\mathbf{A}}^T \left(\frac{\partial^2 R}{\partial \mathbf{E}^{(l-1)} \partial \mathbf{W}^{(i)}[c,d]} \circ \sigma'_{l+1} \right) \left(\mathbf{W}^{(l-1)} \right)^T \tag{3.77}$$

  - Apply Equation (3.76).

Proof. We prove case by case.

- **Case 1.** When $i = l$, since the activation function σ is the ReLU function, it is trivial that the subgradient of its second-order derivative is always 0, since the first-order derivative is the indicator function. Thus, $\mathfrak{H}_{l,l} = 0$.

- **Case 2.** When $i = l-1$, to get Equation (3.73), it is trivial to prove by taking derivative on both sides of Equation (3.65). See the proof of Proposition 3.2 for the proof of Equation (3.74).

- **Case 3.** When $i < l-1$, we first take derivative on both sides of Equation (3.65) in i-th hidden layer. Then we have

$$\mathfrak{H}_{l,i}[:,:,c,d] = \left(\hat{\mathbf{A}} \frac{\partial \mathbf{E}^{(i-1)}}{\partial \mathbf{W}^{(i)}[c,d]} \right)^T \left(\frac{\partial R}{\partial \mathbf{E}^{(i)}} \circ \sigma'_l \right) \tag{3.78}$$

To compute $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$, we first consider an arbitrary (a,b)-th element in $\frac{\partial \mathbf{E}^{(l-1)}}{\partial \mathbf{W}^{(i)}[c,d]}$, i.e., $\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]}$. Then we take the derivative on both sides of $\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]}$, which gives us

$$\begin{aligned}\frac{\partial \mathbf{E}^{(l-1)}[a,b]}{\partial \mathbf{W}^{(i)}[c,d]} &= \sigma'_{l-1}[a,b] \cdot \sum_{e=1}^{n}\sum_{f=1}^{h_i} \hat{\mathbf{A}}[a,e] \frac{\partial \mathbf{E}^{(l-2)}[e,f]}{\partial \mathbf{W}^{(i)}[c,d]} \mathbf{W}^{(l-1)}[f,b] \\ &= \left(\sigma'_{l-1} \circ \left(\hat{\mathbf{A}} \frac{\partial \mathbf{E}^{(l-2)}}{\partial \mathbf{W}^{(i)}[c,d]} \mathbf{W}^{(l-1)}\right)\right)[a,b]\end{aligned} \tag{3.79}$$

It is trivial to get Equation (3.75) by writing out the matrix form of Equation (3.79).

- **Case 4.** When $i = l+1$, to get Equation (3.76), it is trivial to prove by taking derivative on both sides of Equation (3.65). Regarding the computation of $\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b] \partial \mathbf{W}^{(l+1)}[c,d]}$, by Equation (3.66), we have

$$\frac{\partial R}{\partial \mathbf{E}^{(l)}[a,b]} = \left(\hat{\mathbf{A}}^T \left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}} \circ \sigma'\left(\hat{\mathbf{A}}\mathbf{E}^{(l)}\mathbf{W}^{(l+1)}\right)\right)\left(\mathbf{W}^{(l+1)}\right)^T\right)[a,b] \tag{3.80}$$

Then we take derivative on both sides and get

$$\begin{aligned}\frac{\partial^2 R}{\partial \mathbf{E}^{(l)}[a,b]\, \partial \mathbf{W}^{(l+1)}[c,d]} &= \sum_{e=1}^{n}\sum_{f=1}^{h_{l+1}} \hat{\mathbf{A}}^T[a,e] \cdot \left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}} \sigma'_{l+1}\right)[e,f] \cdot \frac{\partial \mathbf{W}^{(l+1)}[b,f]}{\partial \mathbf{W}^{(l+1)}[c,d]} \\ &= \mathbf{I}[b,c] \sum_{e=1}^{n} \hat{\mathbf{A}}^T[a,e] \cdot \left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}} \sigma'_{l+1}\right)[e,d] \\ &= \mathbf{I}[b,c] \left(\hat{\mathbf{A}}^T \left(\frac{\partial R}{\partial \mathbf{E}^{(l+1)}} \circ \sigma'_{l+1}\right)\right)[a,d]\end{aligned} \tag{3.81}$$

- **Case 5.** When $i > l+1$, we get Equation (3.77) by taking derivative on both sides of Equation (3.66).

Putting everything (Cases 1 – 5) together, we complete the proof. QED.

Even with Proposition 3.2 and Theorem 3.2, it is still non-trivial to compute the influence of node u due to (C1) the high-dimensional nature of the Hessian tensor and (C2) the high computational cost of Equation (3.61) due to the inverse operation. Regarding the first challenge (C1), for any node u, we observe that each element in the first-order derivative $\nabla_{\mathbf{W}^{(l)}} R$ is the element-wise first-order derivative, i.e., $\nabla_{\mathbf{W}^{(l)}} R[a,b] = \frac{\partial R}{\partial \mathbf{W}^{(l)}[a,b]}$. Likewise, for the Hessian tensor, we have $\mathfrak{H}_{l,i}[a,b,c,d] = \frac{\partial^2 R}{\partial \mathbf{W}^{(l)}[a,b] \partial \mathbf{W}^{i}[c,d]}$.[19] Thus, the key idea to solving

[19]We use R to represent $R(\mathcal{G}, \mathcal{Y}_{\text{train}}, \Theta)$ for notational simplicity.

the first challenge (C1) is to vectorize the first-order derivative into a column vector and compute the element-wise second-order derivatives accordingly, which naturally flatten the Hessian tensor into a Hessian matrix. More specifically, we first vectorize the first-order derivatives of R with respect to $\mathbf{W}^{(l)}, \forall l \in \{1, \ldots, L\}$ into column vectors and stack them vertically as follows,

$$\mathbf{f}_R = \begin{bmatrix} \text{vec}\left(\nabla_{\mathbf{W}^{(1)}} R\right) = \text{vec}\left(\frac{\partial R}{\partial \mathbf{W}^{(1)}}\right) \\ \vdots \\ \text{vec}\left(\nabla_{\mathbf{W}^{(l)}} R\right) = \text{vec}\left(\frac{\partial R}{\partial \mathbf{W}^{(l)}}\right) \end{bmatrix} \tag{3.82}$$

where $\text{vec}(\cdot)$ vectorizes a matrix to a column vector. Then the flattened Hessian matrix is a matrix $\mathbf{H}_{\text{flat}}$ whose rows are of the form

$$\mathbf{H}_{\text{flat}}\left[(i \cdot c + d), :\right] = \left(\frac{\partial \mathbf{f}_R}{\partial \mathbf{W}^{(i)}[c, d]}\right)^T = \text{vec}\left(\mathfrak{H}_{l,i}[:, :, c, d]\right)^T \tag{3.83}$$

Finally, we follow the strategy to compute the influence of node u: (1) compute $\nabla_{\mathbf{W}^{(l)}} r\left(u, \mathbf{y}_u, \Theta\right)$ for all l-th hidden layer; (2) vectorize $\nabla_{\mathbf{W}^{(l)}} r\left(u, \mathbf{y}_u, \Theta\right)$ and stack to column vector $\mathbf{f}_u$ as shown in Equation (3.82); and (3) compute the influence function $\mathbb{I}\left(u\right) = \mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u$.

Regarding the second challenge (C2), the key idea is to apply Hessian-vector product (Algorithm 3.6) [134, 166], which approximates $\mathbb{I}\left(u\right) = \mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u$ using the power method. Mathematically, it treats $\mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u$ as one vector and iteratively computes

$$\mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u = \mathbf{f}_u + \left(\mathbf{I} - \hat{\mathbf{H}}_{\text{flat}}\right)\left(\mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u\right) \tag{3.84}$$

where $\hat{\mathbf{H}}_{\text{flat}}\left(\mathbf{H}_{\text{flat}}^{-1} \mathbf{f}_u\right)$ is viewed as a vector, and $\hat{\mathbf{H}}_{\text{flat}}$ is the flattened Hessian matrix with respect to a set of sampled nodes at the current iteration. The workflow of the Hessian-vector product is presented in Algorithm 3.6. For any l-th hidden layer (step 2), we first compute the first-order derivative $\nabla_{\mathbf{W}^{(l)}} r\left(u, \mathbf{y}_u, \Theta\right)$ with respect to $\mathbf{W}^{(l)}$ (step 3). Then we vectorize it to a column vector and stack it to $\mathbf{f}_u$ that stores the first-order derivatives of all hidden layers (step 4). After all first-order derivatives are computed, we apply the power method to compute the Hessian-vector product. In each iteration, we first sample a batch of t training nodes, which helps reduce both noise and running time, and then compute the empirical loss over these nodes (steps 7 – 8). After that, we compute the second-order derivatives with Theorem 3.2 and flatten it to a matrix with the strategy shown in Equation (3.83) (step 9). We finish this iteration by computing Equation (3.84) (step 10). The power method (steps 7 – 10) iterates until the maximum number of iteration is reached to ensure the convergence. Consequently, Algorithm 3.6 offers a computationally friendly way to approximate influence

Algorithm 3.6: Hessian-Vector Product

Input : an input graph $\mathcal{G}$, training nodes $\mathcal{V}_{\text{train}}$, ground-truth labels $\mathcal{Y}_{\text{train}}$, node u with label $\mathbf{y}_u$, an L-layer GCN with parameters Θ, a node-wise loss function r, sampling batch size t, #iterations m.

Output : the influence $\mathbb{I}_{\Theta}(u)$ of node u.

initialize $\mathbf{f}_u = []$ as an empty column vector;
for $l = 1 \to L$ **do**
    compute $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ by Equation (3.65);
    vectorize $\nabla_{\mathbf{W}^{(l)}} r(u, \mathbf{y}_u, \Theta)$ and stack it to $\mathbf{f}_u$ as Equation (3.82);
initialize $\left(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u\right)_0 \leftarrow \mathbf{f}_u$;
for $iter = 1 \to m$ **do**
    uniformly sample t training nodes and get $\mathcal{V}_{\text{s}}$;
    compute empirical loss $R_s \leftarrow \frac{1}{|\mathcal{V}_{\text{s}}|} \sum_{i \in \mathcal{V}_{\text{s}}} r(i, \mathbf{y}_i, \Theta)$;
    compute $\hat{\mathbf{H}}_{\text{flat}}$ of R_s with Theorem 3.2 and Equation (3.83);
    compute $\left(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u\right)_{iter} \leftarrow \mathbf{f}_u + \left(\mathbf{I} - \hat{\mathbf{H}}_{\text{flat}}\right)\left(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u\right)_{iter-1}$
return $\left(\mathbf{H}_{\text{flat}}^{-1}\mathbf{f}_u\right)_m$;

functions without involving both tensor-level operations and the computationally expensive matrix inversion.

Remark. We observe that both the first-order derivative (Proposition 3.2) and the second-order derivative (Theorem 3.2) can be computed in the style of neighborhood aggregation. Due to the well-known over-smoothness of GCN and the homophily nature of neighborhood aggregation, the resulting influence functions by Algorithm 3.6 may follow the homophily principle as well. For two nodes under homophily, due to similarity between their influences, their corresponding LOO parameters and LOO errors could be similar as well, which in turn could cause similar uncertainty scores by Definition 3.3. The potential homophily phenomenon in jackknife uncertainty is consistent with the homophily assumption with respect to uncertainty/confidence in existing works [159, 160, 173].

3.3.3 JuryGCN: Algorithm and Applications

In this part, we present JuryGCN to quantify node jackknife uncertainty (Algorithm 3.7) followed by discussions on applications and generalizations of JuryGCN.

JuryGCN algorithm. With Algorithm 3.6, the LOO parameters of each node can be efficiently computed by proper initialization on the perturbation coefficient (ϵ in Equation (3.61)). After that, the LOO predictions and LOO errors can be efficiently inferred by simply switching the original parameters to the LOO parameters, resulting in efficient

jackknife uncertainty quantification.

Based on that, Algorithm 3.7 presents the general workflow of JuryGCN to quantify the jackknife uncertainty of a node. In detail, with proper initialization (step 1), we loop through each training node i to quantify their influences (step 2). For each training node i, it estimates the LOO parameters by leaving out training node i (steps 3 – 6), outputs the LOO predictions of nodes i and u (step 7), and computes the LOO error of each training node i (step 8). After the LOO predictions of node u and the LOO errors of all training nodes are obtained, we compute the lower bound and upper bound of the predictive confidence interval (steps 9 – 10). Finally, the uncertainty of the node is computed as the width of the predictive confidence interval (step 11).

Algorithm 3.7: JuryGCN: Jackknife Uncertainty Quantification

Input : an input graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ with training nodes $\mathcal{V}_{\text{train}}$, a node u, a GCN with parameters Θ, a node-wise loss function r, a coverage parameter α.

Output : the uncertainty $\mathbb{U}_\Theta(u)$ of node u.

1 initialize $\epsilon \leftarrow -\frac{1}{|\mathcal{V}_{\text{train}}|}$;

2 **for** $i \in \mathcal{V}_{train}$ **do**

3 compute node-wise loss $r_{i,\Theta} \leftarrow r(i, \mathbf{y}_i, \Theta)$;

4 compute node-wise derivative $\nabla_\Theta r_{i,\Theta}$;

5 compute $\mathbb{I}_\Theta(i) \leftarrow \mathbf{H}_\Theta^{-1} \nabla_\Theta r_{i,\Theta}$ using Algorithm 3.6;

6 estimate LOO model parameters $\Theta_{\epsilon,i} \leftarrow \Theta + \epsilon \mathbb{I}_\Theta(i)$;

7 output LOO predictions $\text{GCN}(i, \Theta_{\epsilon,i})$ and $\text{GCN}(u, \Theta_{\epsilon,i})$;

8 compute LOO error $err_i \leftarrow \|\mathbf{y}_i - \text{GCN}(i, \Theta_{\epsilon,i})\|_2$;

9 compute lower bound $\mathbb{C}_\Theta^-(u) \leftarrow Q_\alpha(\{\|\text{GCN}(u, \Theta_{\epsilon,i})\|_2 - err_i | i \in \mathcal{V}_{\text{train}} \setminus \{u\}\})$;

10 compute upper bound $\mathbb{C}_\Theta^+(u) \leftarrow Q_{1-\alpha}(\{\|\text{GCN}(u, \Theta_{\epsilon,i})\|_2 + err_i | i \in \mathcal{V}_{\text{train}} \setminus \{u\}\})$;

11 **return** $\mathbb{U}_\Theta(u) \leftarrow \mathbb{C}_\Theta^+(u) - \mathbb{C}_\Theta^-(u)$;

JuryGCN applications. After quantifying the jackknife uncertainty of each node, we utilize the node uncertainty in (1) active learning on node classification and (2) semi-supervised node classification. The details of uncertainty-aware active learning and node classification are as follows.

A – Application #1: Active learning on node classification. In general, active learning sequentially selects a subset of data points to query according to an acquisition function, which is designed to identify the most informative samples, and hence improves the model performance from the obtained labels. In active learning on node classification, we are given (1) an unlabelled training set of nodes (i.e., $\mathcal{V}_{\text{train}}$), (2) a node classifier (e.g., GCN), (3) step size b, and (4) the query budget K. At each query step, according to the acquisition function, we select b nodes from the remaining unlabelled nodes in the training set $\mathcal{V}_{\text{train}}$ to

query and then re-train the classifier. The query step is repeated until the query budget K is exhausted. Intuitively, a node with high predictive uncertainty is a better query candidate compared to the one with certain prediction. From Algorithm 3.7, we can obtain the jackknife uncertainty of each node in $\mathcal{V}_{\text{train}}$. Hence, we define the acquisition function as follows, $\text{Acq}(\mathcal{V}_{\text{train}}) = \text{argmax}_{u \in \mathcal{V}_{\text{train}}} \mathbb{U}_\Theta(u)$, where $\mathbb{U}_\Theta(u)$ is the jackknife uncertainty of node u from the remaining unlabelled nodes in $\mathcal{V}_{\text{train}}$ and is computed in Equation (3.64). Therefore, at each step, we select b unlabelled nodes with the top-b jackknife uncertainty.

B – Application #2: Semi-supervised node classification. In training a GCN-based node classification model, existing approaches treat each training node equally and compute the mean of loss from all training nodes, i.e., $R = \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i \in \mathcal{V}_{\text{train}}} r(i, \mathbf{y}_i, \Theta)$ where the node-specific loss (i.e., $r(\cdot)$) is factored by an identical weight (i.e., $\frac{1}{|\mathcal{V}_{\text{train}}|}$). Intuitively, training nodes have various levels of uncertainty during training, the easily predicted samples (i.e., small uncertainty) may comprise the majority of the total loss and hence dominate the gradient, which makes the training inefficient. To address this problem, based on the jackknife uncertainty estimation, we introduce a dynamic scale factor, β, to adjust the importance for nodes with different levels of uncertainty. Specifically, given the cross-entropy loss that is utilized in semi-supervised node classification, we define the uncertainty-aware node-specific loss as $r_u = -\beta_u^\tau \log\left(p_u^{(i)}\right)$, where $p_u^{(i)}$ is the predictive probability of the i-th class from GCN, and τ is a hyperparameter. The scale factor of node u is computed by normalizing the uncertainty over all training nodes, i.e., $\beta_u = \frac{|\mathbb{U}_\Theta(u)|}{\sqrt{\sum_{i \in \mathcal{V}_{\text{train}}} |\mathbb{U}_\Theta(i)|^2}}$. By introducing β_u, the loss of node with larger uncertainty (i.e., $\mathbb{U}_\Theta$) would be upweighted in the total loss, and hence the training is guided toward nodes with high uncertainty. Therefore, when training a node classification model, we leverage Algorithm 3.7 to estimate the jackknife uncertainty of the training nodes and then apply the obtained uncertainty results to update the loss every few number of epochs.

In addition to Applications #1 and #2, JuryGCN is generalizable to other learning tasks and graph mining models. Here, we only present brief descriptions of each generalization direction, which could be a future direction of JuryGCN.

C – Applications beyond node classification. JuryGCN can be applied to a variety of learning tasks. For example, it can estimate the confidence interval of the predictive dependent variable in a regression task by replacing the cross-entropy loss for node classification with mean squared error (MSE) [166]. Besides active learning, reinforcement learning (RL) is extensively explored to model the interaction between agent and environment. JuryGCN is applicable to RL in the following two aspects. First, in the early stage of training an RL model, the uncertainty quantification results can be utilized to guide the exploration. Second,

Table 3.7: Statistics of datasets to evaluate JuryGCN.

Dataset	# Nodes	# Edges	# Features	# Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3
Reddit	232,965	114,615,892	602	41

through effectively quantifying the uncertainty of the reward after taking certain actions, JuryGCN can also benefit the exploitation. Specifically, as a classic topic of RL, multi-armed bandit can also be a potential application for JuryGCN where uncertainty is highly correlated with decision making.

D – Beyond JuryGCN*: Generalizations to other GNN models.* JuryGCN is able to be generalized to other graph mining models with the help of automatic differentiation in many existing packages, e.g., PyTorch, TensorFlow. In this way, only model parameters and the loss function are required in computing the first-order and second-order derivatives for influence function computation (Algorithm 3.6), which is further used in Algorithm 3.7 for jackknife uncertainty quantification.

3.3.4 Experimental Evaluation

In this part, we conduct experiments to answer the following research questions:

- *RQ1.* How effective is JuryGCN in improving GCN predictions?
- *RQ2.* How efficient is JuryGCN in terms of time and space?
- *RQ3.* How sensitive is JuryGCN to hyperparameters?

Experimental settings. Here, we provide the detailed experimental settings to evaluate JuryGCN.

A – Hardware and software specifications. All datasets are publicly available. All codes are programmed in Python 3.6.9 and PyTorch 1.4.0. All experiments are performed on a Linux server with 2 Intel Xeon Gold 6240R CPUs and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory. The source code of JuryGCN can be accessed at `https://github.com/bluewhalezhou/jurygcn_uq`.

B – Dataset descriptions. We adopt four widely used benchmark datasets, including Cora [174], Citeseer [174], Pubmed [175], and Reddit [36]. Table 3.7 summarizes the statistics of the datasets.

C – Baseline methods. We present the detailed descriptions of all baseline methods that are used in the experiments. For the application of active learning on node classification, we have the following methods.

- *AGE* [176] measures the informativeness of each node by considering the following perspectives, including (1) the entropy of the prediction results, (2) the centrality score, and (3) the distance between the corresponding representation and its nearest cluster center. At each step of query, nodes with the highest scores are selected.
- *ANRMAB* [177] leverages the same selection criterion as in AGE. Additionally, ANRMAB proposes a multi-armed bandit framework to dynamically adjust weights for the three perspectives. ANRMAB utilizes the performance score of the previous query steps as the rewards to learn the optimal combination of weights during the query process.
- *Coreset* [178] is originally proposed for Convolutional Neural networks and performs k-means clustering on the vector representations from the last hidden layer. We follow Hu et al. [179] and apply Coreset on the node representations obtained by GCN. At each query step, we select the node closet to the cluster centroid to label.
- *Centrality* selects the nodes with the highest scores of betweenness centrality at each query step.
- *Degree* queries the node with the highest degree.
- *Random* annotates node randomly at each query step.
- *SOPT-GCN* [180] utilizes Σ-optimal (SOPT) acquisition function as the active learner [181], which requires the graph Laplacian and the indices of labeled nodes. We follow the same setting as in [180] to conduct the experiments.

For semi-supervised node classification, we compare JuryGCN with the following approaches.

- *S-GNN* [159] is an uncertainty-aware estimation framework, which leverages a graph-based kernel Dirichlet distribution to estimate different types of uncertainty associated with the prediction results. We utilize the obtained node-level representations to perform classification in the experiments.
- *GPN* [160] derives three axioms for characterizing the predictive uncertainty and performs Bayesian posterior updates over the predictions based on density estimation and diffusion. Similarly, we utilize the node representations for the classification task.

- *GCN* [7] learns node-level representations by stacking multiple layers of spectral graph convolution.
- *GAT* [8] computes the representation of each node by introducing the learnable attention weights from its neighbors.

D – Evaluation metrics. We use the micro F1 score (Micro-F1) to evaluate the effectiveness. In terms of efficiency, we compare the running time (in seconds) and the memory usage (in MB).

E – Implementation details. In the experiments, we conduct empirical evaluations in two applications, including (1) active learning on node classification and (2) semi-supervised node classification, as described in Section 3.3.3.

- In active learning on node classification, we evaluate all methods on a randomly constructed test set of $1,000$ nodes for *Cora*, *Citeseer*, and *Pubmed*, and $139,779$ nodes for *Reddit*. The validation sets for the first three citation networks contains 500 nodes, and $23,296$ is the size of validation set for *Reddit*. The remaining nodes comprise the training set (i.e., $\mathcal{V}_{\text{train}}$) where the nodes for querying are selected. For *Cora*, *Citeseer*, *Pubmed*, and *Reddit*, (1) we have 10, 10, 5, and 20 randomly selected labels, respectively, to initiate the computation of jackknife uncertainty using Algorithm 3.7; (2) the query budgets are set as 100, 100, 50, and 250; and (3) the query step sizes are 20, 20, 10, and 50.
- In semi-supervised node classification, we randomly selected 100, 100, 50, and 200 nodes from *Cora*, *Citeseer*, *Pubmed*, and *Reddit*, respectively, as the training nodes. The sizes of test sets are the same as those in active learning on node classification. During training, we run Algorithm 3.7 to perform uncertainty estimation over the training nodes every 10 epochs and update the scale factor α accordingly. In addition, we also evaluate the model performance when the number of training nodes is significantly small.

In both applications, we adopt a two-layer GCN with 16 hidden layer dimension. For training, we use the Adam optimizer [182] with learning rate 0.01 and train the GCN classifier for 100 epochs. The coverage parameter (α) in Algorithm 3.7 is 0.025, and the hyperparameter τ is set as 2 in semi-supervised node classification. For all other baseline methods, we use the original settings. We report the average results after 20 runs of each method on two applications.

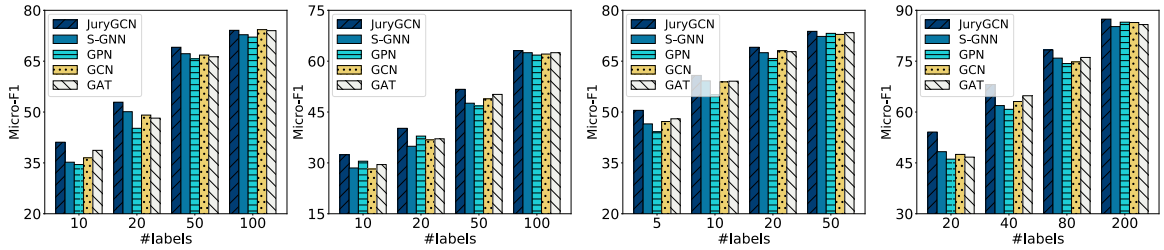
Effectiveness results (RQ1). We evaluate the effectiveness of JuryGCN in the tasks of active learning on node classification and semi-supervised node classification.

Table 3.8: Performance comparison results on active learning on node classification with respect to Micro-F1. Higher is better.

Dataset	Query size	JuryGCN	ANRMAB	AGE	Coreset	Centrality	Degree	Random	SOPT-GCN
Cora	20	**51.1 ± 1.2**	46.8 ± 0.5	49.4 ± 1.0	43.8 ± 0.8	41.9 ± 0.6	38.5 ± 0.7	40.5 ± 1.6	48.8 ± 0.7
	40	**64.7 ± 0.8**	61.2 ± 0.8	58.2 ± 0.7	55.4 ± 0.5	57.3 ± 0.7	48.4 ± 0.3	56.8 ± 1.3	62.6 ± 0.8
	60	**69.9 ± 0.9**	67.8 ± 0.7	65.7 ± 0.8	62.2 ± 0.6	63.1 ± 0.5	58.8 ± 0.6	64.5 ± 1.5	67.9 ± 0.6
	80	**74.2 ± 0.7**	73.3 ± 0.6	72.5 ± 0.4	70.2 ± 0.5	69.1 ± 0.4	67.6 ± 0.4	69.7 ± 1.6	73.6 ± 0.5
	100	**75.5 ± 0.6**	74.9 ± 0.4	74.2 ± 0.3	73.8 ± 0.4	74.1 ± 0.3	73.0 ± 0.2	74.2 ± 1.2	**75.5 ± 0.7**
Citeseer	20	**38.4 ± 1.5**	35.9 ± 1.0	33.1 ± 0.9	30.2 ± 1.2	35.6 ± 1.1	31.5 ± 0.9	30.3 ± 2.3	36.1 ± 0.7
	40	**51.1 ± 0.9**	46.7 ± 1.3	49.5 ± 0.6	42.1 ± 0.8	49.8 ± 1.3	39.8 ± 0.7	41.1 ± 1.8	49.2 ± 0.5
	60	**58.2 ± 0.8**	55.2 ± 0.9	56.1 ± 0.5	52.1 ± 0.9	57.1 ± 0.7	50.1 ± 1.1	49.8 ± 1.3	56.4 ± 0.5
	80	**63.8 ± 1.1**	63.2 ± 0.7	61.5 ± 0.8	59.9 ± 0.6	63.3 ± 1.0	58.8 ± 0.6	58.1 ± 1.1	63.2 ± 0.8
	100	**64.3 ± 1.2**	64.1 ± 0.5	63.2 ± 0.7	62.8 ± 0.4	63.9 ± 0.6	61.8 ± 0.5	62.9 ± 0.8	63.8 ± 0.6
Pubmed	10	**61.8 ± 0.9**	60.5 ± 1.3	58.9 ± 1.1	53.1 ± 0.7	55.8 ± 1.2	56.4 ± 1.5	52.4 ± 1.7	59.5 ± 0.6
	20	**70.2 ± 0.6**	66.8 ± 1.1	68.7 ± 0.7	62.8 ± 0.5	67.2 ± 1.4	64.3 ± 1.0	60.5 ± 1.4	67.9 ± 0.9
	30	**73.9 ± 0.3**	71.6 ± 0.8	72.8 ± 1.0	68.9 ± 0.3	73.5 ± 0.9	70.1 ± 0.7	68.9 ± 1.1	72.3 ± 0.8
	40	74.6 ± 0.4	73.2 ± 0.6	**74.7 ± 0.8**	72.8 ± 0.8	74.1 ± 0.7	72.0 ± 0.8	71.8 ± 1.2	73.8 ± 0.7
	50	**75.4 ± 0.5**	74.7 ± 0.4	75.1 ± 0.5	73.5 ± 0.6	74.2 ± 0.6	72.9 ± 0.5	73.1 ± 1.0	75.2 ± 0.5
Reddit	50	**69.7 ± 1.7**	67.8 ± 0.9	64.2 ± 1.1	62.1 ± 0.6	65.5 ± 1.2	62.5 ± 1.4	63.7 ± 2.4	68.1 ± 1.2
	100	**82.9 ± 1.5**	81.3 ± 1.0	79.5 ± 0.8	81.2 ± 1.0	78.2 ± 0.9	81.1 ± 1.2	80.5 ± 1.6	80.4 ± 1.3
	150	**86.0 ± 1.4**	84.3 ± 0.7	83.2 ± 0.4	84.8 ± 0.9	84.1 ± 1.1	82.5 ± 1.2	81.5 ± 1.4	85.0 ± 1.5
	200	**88.1 ± 0.9**	86.1 ± 0.8	85.8 ± 0.5	85.5 ± 0.8	87.5 ± 0.8	85.4 ± 0.7	83.1 ± 1.8	87.2 ± 0.9
	250	**89.2 ± 0.8**	87.6 ± 0.7	87.1 ± 0.4	86.6 ± 1.1	88.7 ± 0.6	86.1 ± 1.0	87.3 ± 1.5	87.8 ± 1.1

A – Active learning on node classification. Table 3.8 presents the results of active learning at different query steps. We highlight the best performing approach in bold and underline the best competing one. We have the following observations. **(1)** At the final query step (i.e., the 5-th line for each dataset), JuryGCN selects more valuable training nodes than other baseline methods, resulting in higher Micro-F1. Though AGE outperforms JuryGCN on *Pubmed* at 40 queries, the superiority is very marginal (i.e., 0.1%). **(2)** In general, JuryGCN achieves a much better performance when the query size is small. For example, on *Cora*, JuryGCN outperforms SOPT-GCN by 2.3% at 20 queries while the gain becomes 0.6% at 80 queries. One possible explanation is that newly-selected query nodes may contain less fruitful information about new classes/features for a GCN classifier that is already trained with a certain number of labels. **(3)** As the query size increases, the gained performance of each method is diminishing, which is consistent with the second observation. For instance, on *Reddit*, ANRMAB improves 13.5% when query size becomes 100 (from 50), while the gain is only 1.5% at 250 queries.

B – Semi-supervised node classification. We classify nodes by training with various numbers of training nodes. Figure 3.15 summarizes the results on four datasets. We can see that, in general, under the conventional setting of semi-supervised node classification (i.e., the right most bar where the number of training nodes is similar with [7, 8]), JuryGCN improves the performance of GCN to certain extent with respect to Micro-F1 (dark blue vs. yellow). Nonetheless, GCN can achieve a slightly better performance than JuryGCN on *Cora*. In



(a) Cora. (b) Citeseer. (c) Pubmed. (d) Reddit.

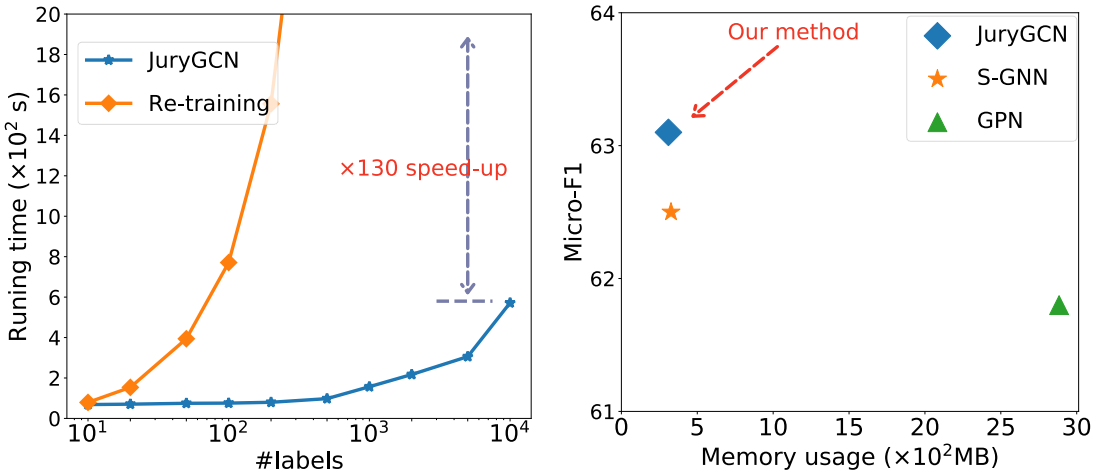
Figure 3.15: Node classification results under various numbers of labels. Higher is better. Best viewed in color.

the mean time, as the number of training nodes becomes smaller, we can observe a consistent superiority of JuryGCN over other methods. For example, on *Citeseer*, when 20 training labels are provided, JuryGCN outperforms the best competing method (i.e., GPN) by 2.3% with respect to Micro-F1, which further demonstrate the effectiveness of JuryGCN in quantifying uncertainty when the training labels are sparse.

To summarize, the uncertainty obtained by JuryGCN is mostly valuable when either the total query budget (for active learning) or the total available labels (for semi-supervised node classification) is small. This is of high importance especially for high-stake applications (e.g., medical image segmentation) where the cost of obtaining high-quality labels is high.

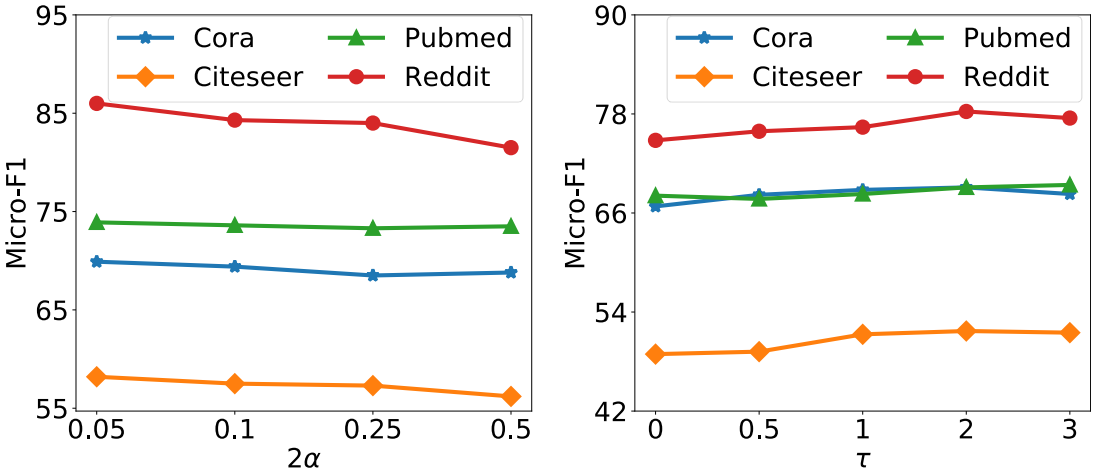
Efficiency results (RQ2). We evaluate the efficiency of JuryGCN on *Reddit* in terms of running time and memory usage (Figure 3.16). Regarding running time, we compare the influence function-based estimation with re-training when leaving out one sample at a time. In Figure 3.16a, as the number of training nodes increases, the total running time of re-training becomes significantly larger than that of JuryGCN. For example, JuryGCN can achieve over 130× speed-up over re-training with $10,000$ training labels. In terms of memory usage in Figure 3.16b, compared to GPN and S-GNN, JuryGCN (i.e., blue diamond at the upper-left corner) reaches the best balance between Micro-F1 and memory usage, with the best effectiveness and lowest memory usage.

Parameter and sensitivity analysis (RQ3). We investigate the sensitivity of JuryGCN with respect to (1) the coverage parameter α in active learning at the third query step and (2) the hyperparameter τ in semi-supervised node classification, where the number of training labels corresponds to the third value on x-axis in Figure 3.15. Figure 3.17 presents the results of sensitivity analysis. For active learning on node classification (i.e., Figure 3.17a), the Micro-F1 results represent the performance at the third query step (i.e., the middle line of each dataset in Table 3.8). And for semi-supervised classification, the number of labels corresponds to the third value on x-axis in Figure 3.15. Regarding the sensitivity of α, results



(a) Running time vs. # labels. (b) Micro-F1 vs. memory usage.

Figure 3.16: Efficiency results with respect to time and memory usage.



(a) Micro-F1 vs. 2α. (b) Micro-F1 vs. τ.

Figure 3.17: Parameter study on the coverage parameter α and the hyperparameter τ.

in Figure 3.17a show that Micro-F1 slightly decreases as α increases. It might be due to the larger target coverage $(1-2\alpha)$[20] caused by the smaller α, resulting in wider confidence interval. Hence, different levels of uncertainty can be accurately captured for selecting valuable query nodes. Regarding the sensitivity of τ, we can observe from Figure 3.17b that JuryGCN is comparatively robust to τ from 0 (i.e., cross-entropy loss) to 3. Meanwhile, by adjusting the importance of training node with the scale factor, Micro-F1 of JuryGCN improves, which is consistent with our findings in the effectiveness results.

[20]The theoretical coverage of jackknife+ is $1-2\alpha$.

CHAPTER 4: DEBIASING GRAPH MINING

Debiasing graph mining is the key component to enable fair graph mining. It addresses the tension between the utility and fairness to further learn graph mining results that are effective and do not favor one group/individual toward another. In this chapter, we present our works in debiasing the mining results based on three fairness definitions. Specifically, we introduce how to ensure (1) individual fairness on graph mining with a principled study of the measure, the debiasing algorithms, and the cost of fairness, (2) degree fairness on graph convolutional network via understanding the mathematical root cause of degree unfairness, and (3) group fairness with respect to multiple sensitive attribute from the information-theoretic perspective.

4.1 INDIVIDUAL FAIRNESS ON GRAPH MINING

In an increasingly connected world, graph mining is playing a more and more important role in many application domains, such as information retrieval [129], community detection [183], recommender systems [184], and security [185]. Decades of research in graph mining have produced a wealth of powerful computational models and algorithms. Despite the remarkable progress, several fundamental questions in relation to algorithmic fairness have remained largely unanswered, e.g., *are the graph mining results fair? If not, how to best mitigate the bias? How would fairness impact the graph mining performance (e.g., ranking precision, classification accuracy)?*

The notion of algorithmic fairness has attracted much attention. To date, researchers have developed a large collection of fair machine learning measures and algorithms, typically for spatial or text data, including statistical measures (e.g., disparate impact [40], statistical parity, equalized odds [186]) and causal reasoning-based measures (e.g., counterfactual fairness [187]). Unlike these settings, a major challenge of fair graph mining lies in the non-IID nature of graph data. Since the data samples (i.e., nodes) in a graph are interconnected, the fundamental IID assumption behind classical fair machine learning methods might be violated. To address this issue, several methods have emerged in recent years, which aim to generalize the traditional fairness notation and bias mitigation algorithms to graph data. For example, fair spectral clustering [46] has been studied to ensure that each cluster contains approximately the same number of elements from each demographic group [188]. Fairness in graph embedding by fulfilling statistical parity has also been explored [47, 48]. Furthermore, there has been research works on fairness in recommendations to enforce statistical parity [189] or other parity-based fairness that measures the differences between

model behaviors for advantaged users and disadvantaged users [190]. It is worth pointing out that the vast majority of the existing work on fair graph mining [46, 47, 48, 188, 189, 190] has almost exclusively been focusing on *group-based fairness*.

However, individual fairness [41] has not been well studied in the context of graph mining. The notation of individual fairness is rooted in the Merriam-Webster's dictionary definition of fairness[21]. In the context of algorithmic fairness, this often translates into a generic design principle that *any two individuals who are similar should receive similar algorithmic outcomes*. Compared with the group-based notation, the individual-based approach offers a fairness measure at a much finer granularity (i.e., at the node level). A thorough study of individual fairness in the context of graph mining will complement and expand the current landscape of fair graph mining, which has mostly focused on group-based fairness. Broadly speaking [191], since bias and discrimination could happen in different forms due to the various settings of tasks with machine automation, there have been debates on which fairness notion should be applied in a given context. The fine granularity of individual fairness might provide a natural remedy for such debates.

This paper presents the first principled study of Individual Fairness on gRaph Mining, which is referred to as the InFoRM problem. We focus on three fundamental questions:

- *Q1. (*InFoRM *Measures)* For the traditional machine learning setting, a major concern of individual fairness lies in its requirement of an appropriate similarity measure, which often requires solving a non-trivial problem that may need expert knowledge to address legal, ethical, or social concerns [41]. Given the rich similarity measures for the graph data [192, 193, 194], we seek to answer the following question: *given a graph mining model and an arbitrary similarity measure, how can we tell if the mining results are fair? If the results are not fair, how can we quantitatively measure the overall bias?*

- *Q2. (*InFoRM *Algorithms)* Generally speaking, a graph mining method consists of three components, including the input graph, the mining model, and the mining results. Each of these three components could introduce and/or amplify the aforementioned bias. *How can we develop generic, effective, and efficient algorithms to mitigate such bias by adjusting the input graph, the mining model, or the mining results, respectively?*

- *Q3. (*InFoRM *Cost)* By mitigating the bias, it is likely to alter the original graph mining results without the fairness consideration, and thus might degrade the mining performance (e.g., ranking, clustering, embedding, etc.). *How can we quantitatively*

[21] In Merriam-Webster, it is defined as *'lack of favoritism toward one side or another'*.

characterize such cost, i.e., to what extent the debiased graph mining results will deviate from the ones without the fairness constraint?

For Q1, we present a generic definition of individual fairness on graph mining based on the Lipschitz property. The fairness measure naturally enables us to quantify the overall bias of graph mining results by the trace of a quadratic form of the mining results. For Q2, building upon the individual fairness measure from Q1, we design three mutually complementary algorithmic frameworks to mitigate the individual bias measure: debiasing the input graph, debiasing the mining model, and debiasing the mining results. For each algorithmic framework, we formulate the framework as an optimization problem, develop effective and efficient solvers, and demonstrate that the framework is applicable to multiple graph mining tasks. In order to debias the input graph, we formulate it as a bi-level optimization problem, where the extra level of optimization can be effectively solved by its KKT conditions [195]. To debias the mining model, we show that the extra time cost incurred due to the fairness consideration is only linear with respect to the number of similarity links. To debias the mining results, we develop a closed-form solution that is applicable to *any* graph mining task, whose results are in the form of a matrix. For Q3, we develop an upper bound on the difference between the debiased mining results and the original mining results. Our analysis reveals that the cost of ensuring individual fairness is closely-related to the input graph structure (e.g., the rank, the spectral norm, etc.).

To our best knowledge, we are the first to study individual fairness on graph mining. In addition to the problem definition, the main contributions of this work can be summarized as follows.

- *Measure.* We provide a novel definition of individual fairness on graph mining, which is capable of (1) identifying if the mining results are fair with respect to any given graph similarity measure and (2) measuring the individual bias as the trace of a quadratic form in the mining results.
- *Algorithms.* We develop generic, effective, and efficient algorithms to mitigate individual bias through the input graph graph, the mining model, and mining results.
- *Analysis.* We provide analysis to (1) understand the quality, complexities, and applicability of the developed debiasing algorithms and (2) reveal the key factors that impact the cost for accommodating individual fairness on graph mining.
- *Evaluations.* We perform extensive empirical evaluations on real-world datasets, which demonstrate that our developed methods are effective and efficient in reducing bias while preserving the performance of vanilla graph mining models.

Table 4.1: Table of symbols in InFoRM.

Symbol	Definition
$\mathbf{A}$	a matrix
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$
$\mathbf{A}^+$	pseudo-inverse of matrix $\mathbf{A}$
$\mathbf{L_A}$	Laplacian matrix of $\mathbf{A}$
$\mathbf{u}$	a vector
$l(\cdot)$	the loss function for a mining task
$\mathbf{S}$	node-node similarity matrix
$\mathbf{Y}$	graph mining results
$\bar{\mathbf{Y}}$	vanilla graph mining results
$\mathbf{Y}^*$	debiased graph mining results
θ	a set of parameters
n, m_1	number of nodes and edges
m_2	number of similarity links
r	dimension of mining results $\mathbf{Y}[i,:]$ for node i
c	damping factor for PageRank
k	number of clusters
d	dimension of node embedding

4.1.1 Problem Definition

In this part, we present the key symbols used throughout the paper (Table 4.1). Then we review the general procedure of several classic graph mining algorithms from the optimization viewpoint. Finally, we formally define three problems of individual fairness for graph mining.

In this work, we use bold upper-case letters for matrices (e.g. $\mathbf{A}$), bold lower-case letters for vectors (e.g. $\mathbf{u}$), and italic lower-case letters for scalars (e.g. c). Regarding matrix indexing conventions, we use rules similar to Numpy in Python. We use $\mathbf{A}[i,j]$ to represent the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column, $\mathbf{A}[i,:]$ to represent the i-th row of matrix $\mathbf{A}$ and $\mathbf{A}[:,j]$ to represent the j-th column of matrix $\mathbf{A}$. We use superscript ^{T} to represent the transpose of a matrix (i.e. $\mathbf{A}^T$ is the transpose of matrix $\mathbf{A}$) and the superscript plus sign to represent the pseudo-inverse of matrix (i.e. $\mathbf{A}^+$ is the pseudo-inverse of matrix $\mathbf{A}$).

Graph mining: An optimization pointview. Given a graph with adjacency matrix $\mathbf{A}$, a graph mining algorithm learns the mining results by optimizing a loss function $l(\mathbf{A}, \mathbf{Y}, \theta)$, where $\mathbf{Y} = \text{argmin}_{\mathbf{Y}}\, l(\mathbf{A}, \mathbf{Y}, \theta)$ is the model output (i.e., the mining results), and θ is the set of all parameters that corresponds to a specific mining task. We use three classic graph mining algorithms, including ranking, clustering, and embedding, summarized in Table 4.2.

The first classic algorithm we apply is PageRank [1]. PageRank is a widely used random

Table 4.2: Examples of graph mining algorithms studied in InFoRM.

Mining Task	Loss Function l	Mining Results Y	Parameters θ
Ranking (PageRank [1])	$\min_{\mathbf{r}} \; c\mathbf{r}^T(\mathbf{I}-\mathbf{A})\mathbf{r} + (1-c)\|\mathbf{r}-\mathbf{e}\|_2^2$	PageRank vector $\mathbf{r}$	damping factor c teleportation vector $\mathbf{e}$
Clustering (spectral clustering [3])	$\min_{\mathbf{U}} \; \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L}\mathbf{U}\right) \quad \text{s.t.} \quad \mathbf{U}^T\mathbf{U}=\mathbf{I}$	matrix $\mathbf{U}$	number of clusters k
Embedding (LINE (1st) [5])	$\max_{\mathbf{X}} \; \sum_{i=1}^n \sum_{j=1}^n \mathbf{A}[i,j]\left(\log g\left(\mathbf{X}[j,:]\mathbf{X}[i,:]^T\right)\right.$ $\left. + b\mathbb{E}_{j'\sim P_n}\left[\log g\left(-\mathbf{X}[j',:]\mathbf{X}[i,:]^T\right)\right]\right)$	embedding matrix $\mathbf{X}$	embedding dimension d number of negative samples b

walk-based ranking algorithm. It outputs the ranking vector $\mathbf{r}$ by minimizing a smoothing term ($\mathbf{r}^T(\mathbf{I}-\mathbf{A})\mathbf{r}$) and a query-specific term ($\|\mathbf{r}-\mathbf{e}\|_2^2$), with c being a regularization parameter to balance the two terms.[22] The second algorithm is spectral clustering [3], which finds the soft cluster membership matrix $\mathbf{U}$ of nodes in a graph by analyzing the spectrum of its graph Laplacian. It has been shown that spectral clustering is equivalent to finding the eigenvectors that are associated with the k smallest eigenvalues, where k is the number of clusters. The final algorithm we use is LINE [5]. Given a graph with n nodes, LINE learns the $n \times d$ embedding matrix $\mathbf{U}$, where each node is mapped into a d-dimensional vector that embeds its structural property.

Individual fairness for graph mining. We aim to answer three questions regarding the individual fairness for graph mining (InFoRM). Based on that, we formally define these three problems and then present our solutions in the subsequent sections.

For Q1 (InFoRM *Measures*), given a graph mining model and an arbitrary similarity measure, we want to (1) determine if the mining results are fair and, if not, (2) quantitatively measure the overall bias. Formally, we define the problem of InFoRM Measures as follows.

Problem 4.1. InFoRM measures.

Input: (1) a non-negative symmetric node-node similarity matrix $\mathbf{S}$, (2) a graph mining algorithm $\mathbf{Y} = \mathrm{argmin}_{\mathbf{Y}}\, l(\mathbf{A}, \mathbf{Y}, \theta)$ from Table 4.2, and (3) a fairness tolerance parameter ϵ.
Output: (1) a binary decision regarding whether or not the mining results are fair and (2) a bias measure $\mathrm{Bias}(\mathbf{Y}, \mathbf{S})$ which measures the overall individual bias of the mining results $\mathbf{Y}$ with respect to the similarity matrix $\mathbf{S}$.

For Q2 (InFoRM *Algorithms*), we aim to develop generic, effective, and efficient algorithms to mitigate the bias of the mining results $\mathrm{Bias}(\mathbf{Y}, \mathbf{S})$, by adjusting the input graph, the mining model, or the mining results. Formally, we define the problem of InFoRM Algorithms as follows.

Problem 4.2. InFoRM algorithms.

[22] $0 < c < 1$ is often called the damping factor in PageRank and its variants.

Input: (1) a non-negative symmetric node-node similarity matrix $\mathbf{S}$, (2) a graph mining algorithm $\mathbf{Y} = \text{argmin}_{\mathbf{Y}}\, l(\mathbf{A}, \mathbf{Y}, \theta)$ from Table 4.2, and (3) a bias measure $\text{Bias}(\mathbf{Y}, \mathbf{S})$ from Problem 4.1.
Output: a revised model output $\mathbf{Y}^*$ which minimizes (1) the loss function $l(\mathbf{A}, \mathbf{Y}, \theta)$ and (2) the individual bias $\text{Bias}(\mathbf{Y}, \mathbf{S})$.

For Q3 (InFoRM *Cost*), we want to quantitatively characterize to what extent the revised graph mining results ($\mathbf{Y}^*$) from Problem 4.2 will deviate from the graph mining results ($\bar{\mathbf{Y}}$) without the fairness constraint. For clarity, we refer to (1) the original results ($\bar{\mathbf{Y}}$) without the fairness constraint as *vanilla mining results*, and (2) the revised results ($\mathbf{Y}^*$) as *debiased mining results*. Formally, we seek to develop an upper bound of such cost, which is defined as follows

Problem 4.3. InFoRM cost.

Input: (1) the vanilla mining results $\bar{\mathbf{Y}}$ without consideration of individual fairness, i.e., $\bar{\mathbf{Y}} = \text{argmin}_{\mathbf{Y}}\, l(\mathbf{A}, \mathbf{Y}, \theta)$ from Table 4.2 and (2) the debiased mining results $\mathbf{Y}^*$ from Problem 4.2.
Output: an upper bound of $\|\mathbf{Y}^* - \bar{\mathbf{Y}}\|_F$.

4.1.2 Problem 4.1: InFoRM Measures

In this part, we address Problem 4.1 and aim to measure the individual fairness and bias for graph mining. That is, given the graph mining results $\mathbf{Y}$ and a node similarity measure $\mathbf{S}$, we want to determine if the mining results are fair, and if not, we want to quantitatively measure the overall bias.

We follow the generic design principle underlying individual fairness that *any two individuals who are similar should receive similar algorithmic outcome* [41]. In our setting, this implies that, if two nodes (i and j) are similar (i.e., $\mathbf{S}[i, j]$ is high), their mining results ($\mathbf{Y}[i, :]$ and $\mathbf{Y}[j, :]$) should be similar as well. We start with the following criterion: the mining results $\mathbf{Y}$ are fair with respect to to the node similarity measure $\mathbf{S}$ if the following condition holds.

$$\|\mathbf{Y}[i, :] - \mathbf{Y}[j, :]\|_F^2 \leq \frac{\epsilon}{\mathbf{S}[i, j]} \;\forall i, j = 1, ..., n \tag{4.1}$$

where $\epsilon > 0$ is a constant for tolerance.

According to Eq (4.1), the difference between the mining results of a pair of nodes i and j is upper bounded by a scalar $\frac{\epsilon}{\mathbf{S}[i,j]}$. The upper bound itself is dependent on the similarity between them $\mathbf{S}[i, j]$. That is, the more similar the node i and the node j, the smaller the

upper-bound, and therefore the smaller the difference between $\mathbf{Y}[i,:]$ and $\mathbf{Y}[j,:]$ is likely to be (i.e., the more similar the mining results between them). An illustrative example is shown in Figure 4.1. Therefore, Eq (4.1) naturally reflects the aforementioned design principle of individual fairness.

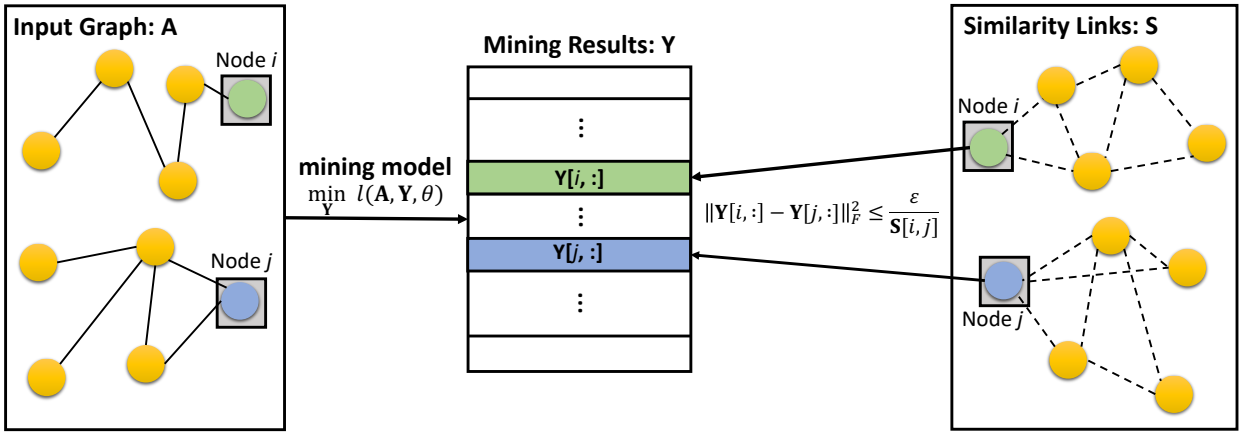


Figure 4.1: An illustrative example of individual fairness for graph mining. $\mathbf{S}$ is a node-node similarity matrix. Individual fairness requires that the difference between the mining results be small for a pair of similar nodes i and j.

The criteria in Eq (4.1) requires the inequality constraint to be held for *every* pair of nodes i and j as long as their similarity $\mathbf{S}[i,j]$ is non-zero.[23] Such a constraint might be too restrictive to be fulfilled. Therefore, we further seek for a relaxed criterion to tell if the mining results are fair. Based on Equation (4.1), we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\|\mathbf{Y}[i,:]-\mathbf{Y}[j,:]\|_F^2\mathbf{S}[i,j]=2\mathrm{Tr}\left(\mathbf{Y}^T\mathbf{L_S}\mathbf{Y}\right)\leq m\epsilon=\delta \tag{4.2}$$

where $\mathbf{L_S}$ is the Laplacian matrix of similarity $\mathbf{S}$, m is the number of non-zero elements in $\mathbf{S}$, and $\mathrm{Tr}\left(\mathbf{Y}^T\mathbf{L_S}\mathbf{Y}\right)$ measures the difference of the mining results between all pairs of nodes. Based on Equation (4.2), we formally define the following fairness definition and bias measure to (1) determine if the mining results are fair and (2) measure the overall bias.

Definition 4.1. (Individual fairness and bias). Given a graph mining results $\mathbf{Y}$ of size $n \times r$, an $n \times n$ non-negative, a symmetric node similarity matrix $\mathbf{S}$, and a constant δ for fairness tolerance, $\mathbf{Y}$ is individually fair with respect to the similarity measure $\mathbf{S}$ if it satisfies

$$\mathrm{Tr}\left(\mathbf{Y}^T\mathbf{L_S}\mathbf{Y}\right)\leq\delta/2 \tag{4.3}$$

[23] If $\mathbf{S}[i,j]=0$, the right hand side of Equation (4.1) will be infinity, which simply means that it becomes a dummy constraint for nodes i and j.

where $\mathbf{L_S}$ is the Laplacian matrix of similarity matrix $\mathbf{S}$. $\text{Bias}(\mathbf{Y}, \mathbf{S}) = \text{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)$ is the overall bias regarding the individual fairness.

For traditional fair machine learning on spatial data or text data, the notation of individual fairness often has a root in the Lipschitz constant [41]. Here, we show that Equation (4.1) can be interpreted from the perspective of the Lipschitz constant.

Definition 4.2. ((D_1, D_2)-Lipschitz property [41]). Given a function f, denote $f(x)$ as the outcome of instance x. We say function f satisfies (D_1, D_2)-Lipschitz property if

$$D_1(f(x), f(y)) \leq L D_2(x, y) \quad \forall \ (x, y), \tag{4.4}$$

where L is the Lipschitz constant, $D_1(\cdot, \cdot)$ and $D_2(\cdot, \cdot)$ are two functions used to measure the dissimilarty of outcomes and instances, respectively.

Let $f(i) = \mathbf{Y}[i, :]$ and $f(j) = \mathbf{Y}[j, :]$, and define $D_1(f(i), f(j)) = \|f(i) - f(j)\|_2$ and $D_2(i, j) = \frac{1}{\mathbf{S}[i,j]}$. It can be shown that the individual fairness definition in Equation (4.1) naturally satisfies (D_1, D_2)-Lipschitz property with ϵ being the Lipschitz constant.

4.1.3 Problem 4.2: InFoRM Algorithms

Generally speaking, a graph mining method consists of three major components, including (1) the input graph, (2) the mining model, and (3) the mining results. Each of these three components can introduce and/or amplify the introduced bias measure. In this part, we present three complementary solutions to mitigate such bias (i.e., $\text{Bias}(\mathbf{Y}, \mathbf{S})$) from the perspective of each component, namely (1) *debiasing the input graph*, (2) *debiasing the mining model*, and (3) *debiasing the mining results*. For each of them, we formulate the bias mitigation problem (i.e., Problem 4.2) as an optimization problem, develop an effective and efficient algorithm to solve the optimization problem, and instantiate it with the three graph mining tasks in Table 4.2. Finally, we compare the three developed solutions.

Debiasing the input graph. Given a graph with adjacency matrix $\mathbf{A}$, if the graph itself is contaminated with bias, it is likely that the bias will be transmitted to, or even amplified in, the mining results $\mathbf{Y}$ if such a graph $\mathbf{A}$ is used to train a graph mining model $l(\mathbf{A}, \mathbf{Y}, \theta)$. The intuition and rationality of debiasing the input graph is as follows. If we have the access to modify the graph and have knowledge about the mining model itself, we aim to learn a new topology of the graph $\tilde{\mathbf{A}}$ so that the bias of mining results based on the modified graph $\tilde{\mathbf{A}}$ is minimized. We also want to make sure that the modified graph $\tilde{\mathbf{A}}$ preserves as

Table 4.3: Algorithm 4.1 instantiations.

Mining Task	Mining Results $\mathbf{Y}$	Partial Derivatives $\mathbf{H}$	Remarks
PageRank	$\mathbf{Y} = \mathbf{r} = (1-c)\,\mathbf{Qe}$	$\mathbf{H} = 2c\mathbf{Q}^T\mathbf{L_S}\mathbf{r}\mathbf{r}^T$	$\mathbf{Q} = \left(\mathbf{I} - c\tilde{\mathbf{A}}\right)^{-1}$
Spectral clustering	$\mathbf{Y} = \mathbf{U}$ = eigenvectors with smallest eigenvalues	$\mathbf{H} = 2\sum_{i=1}^{k}\left(\text{diag}\left(\mathbf{M}_i^T\mathbf{L_S}\mathbf{u}_i\mathbf{u}_i^T\right)\mathbf{1}_{n\times n} - \mathbf{M}_i^T\mathbf{L_S}\mathbf{u}_i\mathbf{u}_i^T\right)$	$\lambda_i = i$-th eigenvalue of $\mathbf{L}_{\tilde{\mathbf{A}}}$ $\mathbf{u}_i$ = eigenvector of $\mathbf{L}_{\tilde{\mathbf{A}}}$ corresponds to λ_i $\mathbf{M}_i = (\lambda_i\mathbf{I} - \mathbf{L}_{\tilde{\mathbf{A}}})^+$
LINE (1st)	$\mathbf{Y}\mathbf{Y}^T = \mathbf{Z}$ (see Equation (4.15))	$\mathbf{H} = 2f\left(\tilde{\mathbf{A}} + \tilde{\mathbf{A}}^T\right) \circ \mathbf{L_S} - 2\text{diag}\left(\mathbf{B}\mathbf{L_S}\right)\mathbf{1}_{n\times n}$	$f()$ calculates Hadamard inverse $\circ$ calculates Hadamard product $\mathbf{B}$ refers to Equation (4.18)

much information of $\mathbf{A}$ as possible. Mathematically, we formulate debiasing the input graph method as the following optimization problem.

$$\min_{\tilde{\mathbf{A}}} \|\tilde{\mathbf{A}} - \mathbf{A}\|_F^2 + \alpha\text{Tr}\left(\mathbf{Y}^T\mathbf{L_S}\mathbf{Y}\right) \quad \text{s.t. } \mathbf{Y} = \underset{\mathbf{Y}}{\text{argmin}}\, l\left(\tilde{\mathbf{A}}, \mathbf{Y}, \theta\right) \tag{4.5}$$

where $\alpha > 0$ is the regularization parameter, and $\mathbf{L_S}$ is the Laplacian matrix of the similarity matrix $\mathbf{S}$.

Equation (4.5) is hard to solve due to its bi-level optimization nature. A generic strategy to solving such a bi-level optimization problem is proposed by Mei and Zhu [149], which reduces the bi-level optimization problem by replacing the lower-level optimization problem with its KKT conditions. By applying this generic strategy to our setting, where the low-level optimization problem is $\mathbf{Y}^* = \text{argmin}_{\mathbf{Y}}\, l\left(\tilde{\mathbf{A}}, \mathbf{Y}, \theta\right)$, we have

$$\min_{\tilde{\mathbf{A}}} \|\tilde{\mathbf{A}} - \mathbf{A}\|_F^2 + \alpha\text{Tr}\left(\mathbf{Y}^T\mathbf{L_S}\mathbf{Y}\right) \quad \text{s.t. } \partial_{\mathbf{Y}} l\left(\tilde{\mathbf{A}}, \mathbf{Y}, \theta\right) = 0 \tag{4.6}$$

We introduce Algorithm 4.1 to solve Equation (4.6). At each iteration of Algorithm 4.1, we first find the mining results $\tilde{\mathbf{Y}}$ based on the current modified graph $\tilde{\mathbf{A}}$ (step 3), and then we use the current mining results $\tilde{\mathbf{Y}}$ to further modify the graph $\tilde{\mathbf{A}}$ (steps 4 – 6). Once we obtain the modified graph $\tilde{\mathbf{A}}$, we use it to generate the debiased mining results $\mathbf{Y}^*$ (step 7). In order to update $\tilde{\mathbf{A}}$, we apply the gradient descent method to the objective function $J = \|\tilde{\mathbf{A}} - \mathbf{A}\|_F^2 + \alpha\text{Tr}\left(\tilde{\mathbf{Y}}^T\mathbf{L_S}\tilde{\mathbf{Y}}\right)$. To this end, we compute the partial derivative of J with respect to $\tilde{\mathbf{A}}$ as

$$\begin{aligned}\frac{\partial J}{\partial \tilde{\mathbf{A}}} &= 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + \alpha \frac{\partial \mathrm{Tr}\left(\tilde{\mathbf{Y}}^T \mathbf{L_S} \tilde{\mathbf{Y}}\right)}{\partial \tilde{\mathbf{A}}} \\ &= 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + \alpha \left[\mathrm{Tr}\left(\frac{\partial \mathrm{Tr}\left(\tilde{\mathbf{Y}}^T \mathbf{L_S} \tilde{\mathbf{Y}}\right)}{\partial \tilde{\mathbf{Y}}^T} \frac{\partial \tilde{\mathbf{Y}}}{\partial \tilde{\mathbf{A}}\,[i,j]}\right)\right] \\ &= 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + \alpha \left[\mathrm{Tr}\left(2\tilde{\mathbf{Y}}^T \mathbf{L_S} \frac{\partial \tilde{\mathbf{Y}}}{\partial \tilde{\mathbf{A}}\,[i,j]}\right)\right]\end{aligned} \tag{4.7}$$

where $\left[\mathrm{Tr}\left(2\tilde{\mathbf{Y}}^T \mathbf{L_S} \frac{\partial \tilde{\mathbf{Y}}}{\partial \tilde{\mathbf{A}}[i,j]}\right)\right]$ is a matrix with its element at i-th row and j-th column as $\mathrm{Tr}\left(2\tilde{\mathbf{Y}}^T \mathbf{L_S} \frac{\partial \tilde{\mathbf{Y}}^T}{\partial \tilde{\mathbf{A}}[i,j]}\right)$ for any $1 \leq i \leq n$, $1 \leq j \leq n$. The corresponding gradient can be computed as $\frac{dJ}{d\tilde{\mathbf{A}}} = \frac{\partial J}{\partial \tilde{\mathbf{A}}} + \left(\frac{\partial J}{\partial \tilde{\mathbf{A}}}\right)^T - \mathrm{diag}\left(\frac{\partial J}{\partial \tilde{\mathbf{A}}}\right)$ if $\tilde{\mathbf{A}}$ is an undirected graph; otherwise, we have its gradient as $\frac{dJ}{d\tilde{\mathbf{A}}} = \frac{\partial J}{\partial \tilde{\mathbf{A}}}$.

A key step in Equation (4.7) is to calculate $\mathbf{H} = \left[\mathrm{Tr}\left(2\tilde{\mathbf{Y}}^T \mathbf{L_S} \frac{\partial \tilde{\mathbf{Y}}}{\partial \tilde{\mathbf{A}}[i,j]}\right)\right]$. Therefore, Algorithm 4.1 can be applied to a variety of graph mining tasks as long as $\frac{\partial \tilde{\mathbf{Y}}}{\partial \tilde{\mathbf{A}}[i,j]}$ exists. We summarize how to calculate $\mathbf{H}$ in Table 4.3 for three graph mining tasks.

Algorithm 4.1: Debiasing the Input Graph

Input : adjacency matrix $\mathbf{A}$, similarity matrix $\mathbf{S}$, a mining algorithm $l(\mathbf{A}, \mathbf{Y}, \theta)$, regularization parameter α, learning rate η.

Output : modified topology $\tilde{\mathbf{A}}$ and debiasd mining results $\mathbf{Y}^*$.

initialize $\tilde{\mathbf{A}} = \mathbf{A}$;
while *not converged* **do**
    find $\tilde{\mathbf{Y}} = \mathrm{argmin}_{\mathbf{Y}}\, l\left(\tilde{\mathbf{A}}, \mathbf{Y}, \theta\right)$;
    calculate partial derivative $\frac{\partial J}{\partial \tilde{\mathbf{A}}}$ by Equation (4.7);
    calculate derivative $\frac{dJ}{d\tilde{\mathbf{A}}}$ based on partial derivative $\frac{\partial J}{\partial \tilde{\mathbf{A}}}$;
    update $\tilde{\mathbf{A}} = \tilde{\mathbf{A}} - \eta \frac{dJ}{d\tilde{\mathbf{A}}}$;
return $\tilde{\mathbf{A}}$ and $\mathbf{Y}^* = \mathrm{argmin}_{\mathbf{Y}}\, l\left(\tilde{\mathbf{A}}, \mathbf{Y}, \theta\right)$;

A – Algorithm 4.1 instantiation with PageRank. Given a symmetric normalized graph $\mathbf{A}$, by Table 4.2, PageRank essentially calculates the fixed-point solution: $\mathbf{r} = (1-c)\left(\mathbf{I} - c\mathbf{A}\right)^{-1}\mathbf{e}$, where c is the damping factor, and $\mathbf{e}$ is the teleportation vector. With that in mind, we can re-write Equation (4.7) as

$$\frac{\partial J}{\partial \tilde{\mathbf{A}}} = 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + 2\alpha \left[\mathbf{r}^T \mathbf{L_S} \frac{\partial \mathbf{r}}{\partial \tilde{\mathbf{A}}\,[i,j]}\right] \tag{4.8}$$

where $\mathbf{r} = (1-c)\left(\mathbf{I} - c\tilde{\mathbf{A}}\right)^{-1}\mathbf{e}$. Based on [147], we have $\frac{\partial \mathbf{r}}{\partial \tilde{\mathbf{A}}[i,j]} = c\mathbf{r}\left[j\right]\left(\mathbf{I} - c\tilde{\mathbf{A}}\right)^{-1}[:,i]$. Then defining $\mathbf{Q} = \left(\mathbf{I} - c\tilde{\mathbf{A}}\right)^{-1}$, we can further simplify Equation (4.8) and get

$$\frac{\partial J}{\partial \tilde{\mathbf{A}}} = 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + 2c\alpha\mathbf{Q}^T\mathbf{L_S}\mathbf{r}\mathbf{r}^T \tag{4.9}$$

Then we can easily learn its debiased topology by applying Algorithm 4.1 with Equation (4.9).

B – Algorithm 4.1 instantiation with spectral clustering. For spectral clustering, as shown in Table 4.2, given an undirected graph with adjacency matrix $\mathbf{A}$, it finds the soft cluster membership matrix $\mathbf{U}$ as the eigenvectors of $\mathbf{L_A}$ associated with the smallest k eigenvalues. With that in mind, we first re-write Equation (4.7) as

$$\frac{\partial J}{\partial \tilde{\mathbf{A}}} = 2\left(\tilde{\mathbf{A}} - \mathbf{A}\right) + 2\alpha\frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \tilde{\mathbf{A}}} \tag{4.10}$$

However, directly calculating $\frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \tilde{\mathbf{A}}}$ is hard, we resort to the chain rule. First, to calculate $\frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}}$, we denote $\mathbf{u}_i$ as the i-th column of $\mathbf{U}$ and write

$$\frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}} = 2\left[\mathrm{Tr}\left(\left(\mathbf{L_S}\mathbf{U}\right)^T\frac{\partial \mathbf{U}}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}\left[i,j\right]}\right)\right] = 2\sum_{i=1}^{k}\left[\mathbf{u}_i^T\mathbf{L_S}\frac{\partial \mathbf{u}_i}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}\left[i,j\right]}\right] \tag{4.11}$$

Denote $\mathbf{M}_i = (\lambda_i\mathbf{I} - \mathbf{L}_{\tilde{\mathbf{A}}})^+$ where λ_i is the i-th eigenvalue. Written in a matrix form, by the derivative of eigenvectors, we have

$$\left[\mathbf{u}_i^T\mathbf{L_S}\frac{\partial \mathbf{u}_i}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}\left[i,j\right]}\right] = \left[\mathbf{u}_i^T\mathbf{L_S}\mathbf{M}\left[:,i\right]\mathbf{u}_i\left[j\right]\right] = \mathbf{M}_i^T\mathbf{L_S}\mathbf{u}_i\mathbf{u}_i^T \tag{4.12}$$

Then, based on [195], we get

$$\begin{aligned}\frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \tilde{\mathbf{A}}} &= \mathrm{diag}\left(\frac{\partial \mathrm{Tr}(\mathbf{U}^T\mathbf{L_S}\mathbf{U})}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}}\right)\mathbf{1}_{n\times n} - \frac{\partial \mathrm{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right)}{\partial \mathbf{L}_{\tilde{\mathbf{A}}}} \\ &= 2\sum_{i=1}^{k}\left(\mathrm{diag}\left(\mathbf{M}_i^T\mathbf{L_S}\mathbf{u}_i\mathbf{u}_i^T\right)\mathbf{1}_{n\times n} - \mathbf{M}_i^T\mathbf{L_S}\mathbf{u}_i\mathbf{u}_i^T\right)\end{aligned} \tag{4.13}$$

where $\mathbf{1}_{n\times n}$ is an $n \times n$ matrix filled with 1. To learn the debiased topology, we can apply Algorithm 4.1 by combining Equations (4.10) and (4.13).

C – Algorithm 4.1 instantiation with LINE. Denote the $n \times d$ embedding matrix learned by

LINE (1st) as $\mathbf{X}$. We apply the chain rule and re-write Equation (4.7) as

$$\begin{aligned}\frac{\partial J}{\partial \tilde{\mathbf{A}}} &= 2\left(\tilde{\mathbf{A}}-\mathbf{A}\right)+2\alpha\frac{\partial \mathrm{Tr}\left(\mathbf{X}^T\mathbf{L_S}\mathbf{X}\right)}{\partial \tilde{\mathbf{A}}} = 2\left(\tilde{\mathbf{A}}-\mathbf{A}\right)+2\alpha\frac{\partial \mathrm{Tr}\left(\mathbf{L_S}\mathbf{X}\mathbf{X}^T\right)}{\partial \tilde{\mathbf{A}}} \\ &= 2\left(\tilde{\mathbf{A}}-\mathbf{A}\right)+2\alpha\left[\mathrm{Tr}\left(\mathbf{L}_\mathbf{S}^T\frac{\partial \mathbf{X}\mathbf{X}^T}{\partial \tilde{\mathbf{A}}\left[i,j\right]}\right)\right]\end{aligned} \tag{4.14}$$

Let $\mathbf{Z}=\mathbf{X}\mathbf{X}^T$. We use the following method to compute $\frac{\partial \mathbf{Z}}{\partial \tilde{\mathbf{A}}[i,j]}$.[24] First, we have

$$\mathbf{Z}\left[s,t\right]=\log\left(\frac{T\left(\tilde{\mathbf{A}}\left[s,t\right]+\tilde{\mathbf{A}}\left[t,s\right]\right)}{d_s d_t^{3/4}+d_s^{3/4}d_t}\right)-\log b \tag{4.15}$$

where d_i is the out degree of node i, and $T=\sum_{i=1}^{n} d_i^{3/4}$. Then we have the partial derivative

$$\begin{aligned}\frac{\partial \mathbf{Z}\left[s,t\right]}{\partial \tilde{\mathbf{A}}\left[i,j\right]} =& \frac{3}{4Td_i^{1/4}}+\frac{1}{\tilde{\mathbf{A}}\left[s,t\right]+\tilde{\mathbf{A}}\left[t,s\right]}\left(\mathbb{1}\left[i=s,j=t\right]+\mathbb{1}\left[i=t,j=s\right]\right) \\ &-\frac{4d_s^{1/4}+3d_t^{1/4}}{4\left(d_s d_t^{1/4}+d_s^{5/4}\right)}\mathbb{1}\left[i=s\right]-\frac{3d_s^{1/4}+4d_t^{1/4}}{4\left(d_s^{1/4}d_t+d_t^{5/4}\right)}\mathbb{1}\left[i=t\right]\end{aligned} \tag{4.16}$$

where $\mathbb{1}$ is the indicator function.

With Equation (4.16), we get the following matrix form of derivatives

$$\left[\mathrm{Tr}\left(\mathbf{L}_\mathbf{S}^T\frac{\partial \mathbf{Z}}{\partial \tilde{\mathbf{A}}\left[i,j\right]}\right)\right]=2f\left(\tilde{\mathbf{A}}+\tilde{\mathbf{A}}^T\right)\circ\mathbf{L_S}-2\mathrm{diag}\left(\mathbf{B}\mathbf{L_S}\right)\mathbf{1}_{n\times n} \tag{4.17}$$

where $f(\tilde{\mathbf{A}})$ calculates the Hadamard inverse matrix $\tilde{\mathbf{A}}$, $\circ$ is the Hadamard product operator, and $\mathbf{B}$ has the following form

$$\mathbf{B}=\frac{3}{4}f\left(\mathbf{d}^{5/4}\left(\mathbf{d}^{-1/4}\right)^T+\mathbf{d}\mathbf{1}_{1\times n}\right)+f\left(\mathbf{d}^{3/4}\left(\mathbf{d}^{1/4}\right)^T+\mathbf{d}\mathbf{1}_{1\times n}\right) \tag{4.18}$$

with $\mathbf{d}^x$ being a column vector of the form $\mathbf{d}^x\left[i\right]=d_i^x$.

Lemma 4.1. (Time and space complexities of Algorithm 4.1 for LINE) To calculate the partial derivatives $\mathbf{H}$, it takes $O\left(min\{m_1,m_2\}+m_2\right)$ time and $O\left(min\{m_1,m_2\}+n\right)$ space, where n is the number of nodes, and m_1 and m_2 are the number of edges in $\mathbf{A}$ and $\mathbf{S}$, respectively.

[24] This method was first developed in [33] in order to establish the relationship between LINE (2nd) and matrix factorization.

Proof. It takes $O\left(min\{m_1, m_2\}\right)$ time to calculate $f\left(\mathbf{A}+\mathbf{A}^T\right) \circ \mathbf{L_S}$ and $O\left(m_2\right)$ time to calculate $\text{diag}\left(\mathbf{BL_S}\right)$. Thus, the overall time complexity is $O\left(min\{m_1, m_2\} + m_2\right)$. For space complexity, it takes $O\left(min\{m_1, m_2\}\right)$ space to save $f\left(\mathbf{A}+\mathbf{A}^T\right) \circ \mathbf{L_S}$ in sparse format and $O\left(n\right)$ space to save $\text{diag}\left(\mathbf{BL_S}\right)$. Therefore, the overall space complexity is $O(min\{m_1, m_2\} + n)$. QED.

Debiasing the mining model. The intuition and rationality of debiasing the mining model is as follows. If we directly incorporate the bias measure ($\text{Bias}\left(\mathbf{Y}, \mathbf{S}\right)$) as a regularization term in the loss function of the given mining model (i.e., those listed in Table 4.2), the generated mining results are likely to have a small bias. Mathematically, we formulate it as

$$\mathbf{Y}^* = \underset{\mathbf{Y}}{\text{argmin}}\ J = l\left(\mathbf{A}, \mathbf{Y}, \theta\right) + \alpha \text{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right) \tag{4.19}$$

where $\alpha > 0$ is the parameter for regularization.

To solve Equation (4.19), we apply (stochastic) gradient descent/ascent-based methods. Since $\mathbf{Y}$ is, in general, not symmetric, its derivative is $\frac{dJ}{d\mathbf{Y}} = \frac{\partial J}{\partial \mathbf{Y}}$. We have

$$\begin{aligned} \frac{dJ}{d\mathbf{Y}} = \frac{\partial J}{\partial \mathbf{Y}} &= \frac{\partial l\left(\mathbf{A}, \mathbf{Y}, \theta\right)}{\partial \mathbf{Y}} + \alpha \frac{\partial \text{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)}{\partial \mathbf{Y}} \\ &= \frac{\partial l\left(\mathbf{A}, \mathbf{Y}, \theta\right)}{\partial \mathbf{Y}} + \alpha \left(\mathbf{L_S} + \mathbf{L_S}^T\right) \mathbf{Y} \\ &= \frac{\partial l\left(\mathbf{A}, \mathbf{Y}, \theta\right)}{\partial \mathbf{Y}} + 2\alpha \mathbf{L_S} \mathbf{Y} \end{aligned} \tag{4.20}$$

The last equality holds because $\mathbf{S}$ is a symmetric matrix and so is its Laplacian matrix $\mathbf{L_S}$. We can see that, compared with the original graph mining model without the fairness consideration, the extra time to calculate $\mathbf{L_S Y}$ is just linear with respect to the number of similarity links in $\mathbf{S}$. Based on that, we develop a generic algorithmic framework (i.e., Algorithm 4.2) to debias the mining model. The key of Algorithm 4.2 is to solve Equation (4.19) (step 2). This can be done either by (stochastic) gradient descent/ascent method based on Equation (4.20), or by a specific algorithm designed for the given graph mining model. For the latter, we give three examples for the mining models in Table 4.2.

Algorithm 4.2: Debiasing the Mining Model

Input : adjacency matrix $\mathbf{A}$, similarity matrix $\mathbf{S}$, a mining model $l(\mathbf{A}, \mathbf{Y}, \theta)$, regularization parameter α, learning rate η.

Output : debiased mining results $\mathbf{Y}^*$.

1 solve Equation (4.19);

2 **return** $\mathbf{Y}^*$;

A – Algorithm 4.2 instantiation with PageRank. Instantiating Equation (4.19) with PageRank, we have that $\mathbf{r}^* = \underset{\mathbf{r}}{\operatorname{argmin}}\ J = c\mathbf{r}^T(\mathbf{I}-\mathbf{A})\mathbf{r} + (1-c)\|\mathbf{r}-\mathbf{e}\|_F^2 + \alpha\mathbf{r}^T\mathbf{L_S}\mathbf{r}$. We can show that J is a quadratic convex function with respect to $\mathbf{r}$ as long as the regularization parameter α is positive. Therefore, its optima has a zero derivative $\frac{\partial J}{\partial \mathbf{r}} = 0$. Then we have

$$\frac{\partial J}{\partial \mathbf{r}} = 2\mathbf{r} - 2c\mathbf{A}\mathbf{r} + 2\alpha\mathbf{L_S}\mathbf{r} - 2(1-c)\mathbf{e} = 0 \Rightarrow \mathbf{r}^* = c\left(\mathbf{A} - \frac{\alpha}{c}\mathbf{L_S}\right)\mathbf{r}^* + (1-c)\mathbf{e} \tag{4.21}$$

which is equivalent to PageRank on a new transition matrix $\mathbf{A} - \frac{\alpha}{c}\mathbf{L_S}$. Furthermore, if the similarity matrix $\mathbf{S}$ is symmetrically normalized (i.e., $\mathbf{L_S} = \mathbf{I} - \mathbf{S}$), we have $\mathbf{r}^* = \left(\frac{c}{1+\alpha}\mathbf{A} + \frac{\alpha}{1+\alpha}\mathbf{S}\right)\mathbf{r}^* + \frac{1-c}{1+\alpha}\mathbf{e}$.

B – Algorithm 4.2 instantiation with spectral clustering. Instantiating Equation (4.19) with spectral clustering, we have that $\mathbf{U}^* = \underset{\mathbf{U}}{\operatorname{argmin}}\ J = \operatorname{Tr}\left(\mathbf{U}^T\mathbf{L_A}\mathbf{U}\right) + \alpha\operatorname{Tr}\left(\mathbf{U}^T\mathbf{L_S}\mathbf{U}\right) = \operatorname{Tr}\left(\mathbf{U}^T\mathbf{L}_{\mathbf{A}+\alpha\mathbf{S}}\mathbf{U}\right)$, which is very similar to the loss function of the original spectral clustering without the fairness consideration in Table 4.2, and both loss functions require $\mathbf{U}$ to be orthonormal. The only difference is that the debiased $\mathbf{U}^*$ is equivalent to the eigenvectors of $\mathbf{L}_{\mathbf{A}+\alpha\mathbf{S}}$, instead of the original $\mathbf{L_A}$, with the k smallest eigenvalues. In other words, debiased spectral clustering $\mathbf{U}^*$ is essentially spectral clustering on a modified graph with an adjacency matrix $\mathbf{A} + \alpha\mathbf{S}$.

C – Algorithm 4.2 instantiation with LINE. Instantiating Equation (4.19) with LINE (1st), we have

$$\begin{aligned}\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmax}} \sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{A}[i,j]\Bigg(&\log g\left(\mathbf{X}[j,:]\,\mathbf{X}[i,:]^T\right) \\ &+ b\mathbb{E}_{j'\sim P_n}\left[\log g\left(-\mathbf{X}[j',:]\,\mathbf{X}[i,:]^T\right)\right]\Bigg) - \alpha\operatorname{Tr}\left(\mathbf{X}^T\mathbf{L_S}\mathbf{X}\right)\end{aligned} \tag{4.22}$$

where $g(x) = 1/(1+e^{-x})$ is the sigmoid function.

Due to the unique edge sampling strategy of LINE, we factorize the bias term (i.e., $\operatorname{Tr}\left(\mathbf{X}^T\mathbf{L_S}\mathbf{X}\right)$) and consider it edge-wise. Specifically, for an edge (i,j), LINE (1st) aims to maximize the following objective function

$$\log g\left(\mathbf{x}_j\mathbf{x}_i^T\right) + b\mathbb{E}_{j'\sim P_n}\left[\log g\left(-\mathbf{x}_{j'}\mathbf{x}_i^T\right)\right] - \alpha\|\mathbf{x}_i - \mathbf{x}_j\|_2^2\mathbf{S}[i,j] \tag{4.23}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the node embeddings for node i and j (i.e. i-th row and j-th row in $\mathbf{X}$), respectively. It is worth pointing out that adding such a bias constraint does not increase the time complexity. To see this, we can show that the optimization for one edge in the original LINE takes $O(db)$ time, where b is the number of negative samples, and d is the

embedding dimension. By adding the bias constraint $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \mathbf{S}[i,j]$, it would introduce an additional $O(d)$ time per edge, which does not change the overall time complexity in the big-O notation.

Debiasing the mining results. If we do not have access to either the input graph or the graph mining model, we could mitigate the individual bias via a post-processing strategy on the mining results. We formulate this mitigation strategy as a regularized optimization problem below.

$$\mathbf{Y}^* = \underset{\mathbf{Y}}{\operatorname{argmin}}\ J = \|\mathbf{Y} - \bar{\mathbf{Y}}\|_F^2 + \alpha \operatorname{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right) \tag{4.24}$$

where $\bar{\mathbf{Y}}$ is the vanilla mining results, i,e., the original model output without the consideration of individual fairness.

We can prove that J is a convex function since $\|\mathbf{Y} - \bar{\mathbf{Y}}\|_F^2$ and $\operatorname{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)$ are both convex, and α is a positive regularization hyper-parameter. Thus, the optimal solution for Equation (4.24) can be obtained by taking the derivative of J with respect to $\mathbf{Y}$ and setting it to zero.

$$\frac{\partial J}{\partial \mathbf{Y}} = \frac{\partial \|\mathbf{Y} - \bar{\mathbf{Y}}\|_F^2}{\partial \mathbf{Y}} + \alpha \frac{\partial \operatorname{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)}{\partial \mathbf{Y}} = 0 \Rightarrow 2\mathbf{Y} - 2\bar{\mathbf{Y}} + 2\alpha \mathbf{L_S} \mathbf{Y} = 0 \Rightarrow (\mathbf{I} + \alpha \mathbf{L_S})\, \mathbf{Y}^* = \bar{\mathbf{Y}} \tag{4.25}$$

Equation (4.25) indicates that debiasing the mining results is essentially solving a linear system with respect to the debiased mining results. Many linear system solvers can be utilized, e.g., Krylov subspace method, conjugate gradient method, etc. The algorithm for debiasing mining results is summarized in Algorithm 4.3.

Algorithm 4.3: Debiasing the Mining Results

Input : vanilla graph mining results $\bar{\mathbf{Y}}$, similarity matrix $\mathbf{S}$, regularization parameter α.

Output : debiased mining results $\mathbf{Y}^*$.

1 calculate $\mathbf{I} + \alpha \mathbf{L_S}$;
2 solve $(\mathbf{I} + \alpha \mathbf{L_S})\, \mathbf{Y}^* = \bar{\mathbf{Y}}$;
3 **return** $\mathbf{Y}^*$;

Analysis and discussions. The three developed algorithmic frameworks are mutually complementary with each other. For example, to debias the input graph (Algorithm 4.1), we modify the input graph and mining results in an iterative way. The potential benefit is that it eliminates or mitigates the bias from the 'origin' (i.e., the input graph), and thus the modified/debiased graph might also help mitigate the bias for other related graph mining models. In order to debias the mining model (Algorithm 4.2), we need the knowledge of

the details of the mining model itself, whereas the input graph remains unchanged. On the contrary, neither the knowledge of the input graph nor the mining model is required for debiasing the mining results (Algorithm 4.3).

Regarding the applicability of the developed frameworks, we can always debias the input graph as long as the gradient $\frac{d\mathbf{Y}}{d\mathbf{A}[i,j]}$ in Equation (4.7) exists. For debiasing the mining model method, it is applicable as long as a (stochastic) gradient descent solution for the vanilla graph mining algorithm (i.e., the one without the consideration of individual fairness) exists. This is because adding the additional bias term in Equation (4.19) would only incur a linear term in computing the gradient. Besides, the convexity of the vanilla graph mining algorithm will not be affected since the bias term itself is convex. For debiasing the mining results method, it can be applied to *any* graph mining model whose mining results $\mathbf{Y}$ are in a matrix form, thanks to its model-agnostic closed-form solution.

The three developed algorithmic frameworks differ in computational efficiency. First, the debiasing the input graph method is the most time- and space-consuming, since $\mathbf{H}$ is usually non-trivial to compute and could be a full matrix with $O(n^2)$ space cost. However, for some special mining models, including all three models in Table 4.2, we can handle this issue by exploring the low-rank structure of $\mathbf{H}$. For example, in PageRank and spectral clustering, $\mathbf{Q}^T\mathbf{L_S r}$ and $\mathbf{M}_i^T\mathbf{L_S u}_i$ are both column vectors of length n, where $\mathbf{Q}^T\mathbf{L_S r}$ can be computed by power iterations, and $\mathbf{M}_i$ can be efficiently calculated by singular value decomposition (SVD). In LINE, $\text{diag}(\mathbf{BL_S})\mathbf{1}_{n\times n}$ is equivalent to vectorizing the diagonal of $\mathbf{BL_S}$ as a column vector and multiplying with $\mathbf{1}_{1\times n}$. As Lemma 4.1 says, Algorithm 4.1 for LINE has a linear complexity. Second, for debiasing the mining model method, the additional time incurred in the gradient computation is linear with respect to the number of non-zero elements in $\mathbf{S}$ (m_2, the number of links in the similarity matrix $\mathbf{S}$), according to Equation (4.20). Finally, for debiasing the mining results method, it always has a linear time complexity with respect to m_2 (the number of links in the similarity matrix $\mathbf{S}$), since we only need to solve a linear system in Equation (4.25).

4.1.4 Problem 4.3: InFoRM Cost

In this part, we address Problem 4.3 (i.e., InFoRM cost), aiming to characterize how the debiased graph mining results $\mathbf{Y}^*$ would deviate from the vanilla ones $\bar{\mathbf{Y}}$ without the fairness consideration. Among the three developed debiasing algorithms, debiasing the mining results method (Algorithm 4.3), being agnostic to both the input graph and the mining model, has the widest applicability. Therefore, we will mainly focus on characterizing the InFoRM cost of this method. The InFoRM costs of the other two developed methods (i.e., debiasing the

input graph and debiasing the mining model) are dependent on the specific graph and/or the specific mining model. After that, we provide a case study that analyzes the InFoRM cost for PageRank with the debiasing mining model method (Algorithm 4.2).

Cost of debiasing the mining results. For debiasing the mining results method, the solution of Equation (4.25) is always optimal, since Equation (4.24) is a convex optimization problem. Based on this solution, we characterize the cost of debiasing the mining results in Lemma 4.2.

Lemma 4.2. Given a graph of n nodes with the adjacency matrix $\mathbf{A}$ and a node-node similarity matrix $\mathbf{S}$, let $\bar{\mathbf{Y}}$ be the vanilla mining results without considering the fairness and $\mathbf{Y}^* = \left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1} \bar{\mathbf{Y}}$ be the solution of Equation (4.24) (i.e., the debiased mining results). If $\|\mathbf{S} - \mathbf{A}\|_F = \delta$, we have that

$$\|\mathbf{Y}^* - \bar{\mathbf{Y}}\|_F \leq 2\alpha\sqrt{n}\left(\delta + \sqrt{r\left(\mathbf{A}\right)}\sigma_{\max}\left(\mathbf{A}\right)\right)\|\bar{\mathbf{Y}}\|_F \tag{4.26}$$

where $r\left(\mathbf{A}\right)$ is the rank of $\mathbf{A}$, and $\sigma_{\max}\left(\mathbf{A}\right)$ is the largest singular value of $\mathbf{A}$.

Proof. Since $\mathbf{Y}^* = \left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1} \bar{\mathbf{Y}}$, by re-arranging terms, we have

$$\|\mathbf{Y}^* - \bar{\mathbf{Y}}\|_F = \alpha\|\mathbf{L_S}\mathbf{Y}^*\|_F \leq \alpha\|\mathbf{L_S}\|_F\|\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\|_F\|\bar{\mathbf{Y}}\|_F \tag{4.27}$$

The last inequality above holds due to the triangle inequality. Since $\|\mathbf{Y}\|_F$ is a constant, our goal is to find upper bounds of $\|\mathbf{L_S}\|_F$ and $\|\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\|_F$, respectively.

First, we derive an upper bound of $\|\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\|_F$. For any matrix $\mathbf{W}$, we have $\|\mathbf{W}\|_F = \sqrt{\sum_{i=1}^{r(\mathbf{W})} \sigma_i^2\left(\mathbf{W}\right)} \leq \sqrt{r\left(\mathbf{W}\right)}\sigma_{\max}\left(\mathbf{W}\right)$, where $\sigma_{\max}\left(\mathbf{W}\right)$ is the largest singular value (i.e., the spectral norm) of matrix $\mathbf{W}$, and $r\left(\mathbf{W}\right)$ is the rank of matrix $\mathbf{W}$ [196]. Applying it to $\|\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\|_F$, we have the following inequality

$$\|\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\|_F \leq \sqrt{n}\sigma_{\max}\left(\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}\right) = \frac{\sqrt{n}}{\sigma_{\min}\left(\mathbf{I} + \alpha \mathbf{L_S}\right)} = \sqrt{n} \tag{4.28}$$

The above inequality holds due to the facts that (1) $\left(\mathbf{I} + \alpha \mathbf{L_S}\right)^{-1}$ is full-rank $n \times n$ matrices; and (2) graph laplacian $\mathbf{L_S}$ is a symmetric singular matrix with smallest singular value being 0, which implies that $\sigma_{\min}\left(\mathbf{I} + \alpha \mathbf{L_S}\right) = 1$.

Next, we derive an upper bound of $\|\mathbf{L_S}\|_F$. Denote $d_i = \mathbf{L_S}[i,i]$. We have

$$\begin{aligned}\|\mathbf{L_S}\|_F^2 &= \sum_i \left(d_i^2 + \sum_{j:j\neq i} \mathbf{S}[i,j]^2 \right) = \sum_i \left[\left(\sum_{j:j\neq i} \mathbf{S}[i,j] \right)^2 + \sum_{j:j\neq i} \mathbf{S}[i,j]^2 \right] \\ &\leq \sum_i \left(2\sum_{j:j\neq i} \mathbf{S}[i,j]^2 + \sum_{k:k\neq i} \mathbf{S}[i,k]^2 + \sum_{l:l\neq i} \mathbf{S}[i,l]^2 \right) \leq 4\|\mathbf{S}\|_F^2\end{aligned} \tag{4.29}$$

Taking square root on both sides and applying triangle inequality, we have the upper bound of $\|\mathbf{L_S}\|_F$ as follows.

$$\|\mathbf{L_S}\|_F \leq 2\|\mathbf{S}\|_F \leq 2\left(\delta + \|\mathbf{A}\|_F\right) \leq 2\left(\delta + \sqrt{r(\mathbf{A})}\sigma_{\max}(\mathbf{A})\right) \tag{4.30}$$

We complete the proof by combining Equations (4.27), (4.28), and (4.30). QED.

From Lemma 4.2, we can see that the cost of debiasing the mining results depends on a number of factors, including the size of input graph (i.e., the number of nodes n), the difference δ between $\mathbf{S}$ and $\mathbf{A}$, the rank of the adjacency matrix $r(\mathbf{A})$, and the largest singular value of the adjacency matrix $\sigma_{\max}(\mathbf{A})$. $r(\mathbf{A})$ of many real graphs could be small, since they often have an (approximate) low-rank structure. Furthermore, if $\mathbf{A}$ is a symmetrically normalized matrix (i.e., $\mathbf{A} \leftarrow \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the degree matrix), its largest singular value is upper bounded by 1. These facts help make the overall upper bound in Lemma 4.2 to be relatively small.

Cost of debiasing the mining model: A case study on PageRank. Given a graph with adjacency matrix $\mathbf{A}$, similarity matrix $\mathbf{S}$, and regularization parameter α, the cost of debiasing the mining model method on PageRank is summarized in Lemma 4.3.

Lemma 4.3. Given a graph with the symmetrically normalized adjacency matrix $\mathbf{A}$ and node-node similarity matrix $\mathbf{S}$, let $\bar{\mathbf{r}}$ be the PageRank vector without considering the fairness and $\mathbf{r}^*$ be the debiased PageRank vector as in Equation (4.21). If teleportation vector $\|\mathbf{e}\|_1 = 1$ and similarity matrix $\|\mathbf{S} - \mathbf{A}\|_F = \delta$, it satisfies

$$\|\mathbf{r}^* - \bar{\mathbf{r}}\|_F \leq \frac{2\alpha n}{1-c}\left(\delta + \sqrt{r(\mathbf{A})}\sigma_{\max}(\mathbf{A})\right) \tag{4.31}$$

where c is the damping factor, and α is the regularization parameter for individual fairness.

Proof. Recall that debiasing the mining model on PageRank is equivalent to solving the linear system $\mathbf{r} = c\left(\mathbf{A} - \frac{\alpha}{c}\mathbf{L_S}\right)\mathbf{r} + (1-c)\,\mathbf{e}$. After re-arranging terms, we can get its closed-form solution as $\mathbf{r}^* = (1-c)\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1}\mathbf{e}$. If we do not consider the individual fairness

constraint, we can easily set $\mathbf{L_S} = \mathbf{0}$ and get $\bar{\mathbf{r}} = (1-c)\left(\mathbf{I} - c\mathbf{A}\right)^{-1}\mathbf{e}$. Then we have the cost of individual fairness in PageRank as

$$\begin{aligned}
\|\mathbf{r}^* - \bar{\mathbf{r}}\|_F &= (1-c)\,\|\left(\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1} - \left(\mathbf{I} - c\mathbf{A}\right)^{-1}\right)\mathbf{e}\|_F \\
&\leq (1-c)\,\|\left(\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1} - \left(\mathbf{I} - c\mathbf{A}\right)^{-1}\right)\|_F\|\mathbf{e}\|_F \\
&\leq (1-c)\,\|\left(\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1} - \left(\mathbf{I} - c\mathbf{A}\right)^{-1}\right)\|_F \\
&= (1-c)\,\|\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1}\cdot\alpha\mathbf{L_S}\cdot\left(\mathbf{I} - c\mathbf{A}\right)^{-1}\|_F \\
&\leq \alpha\,(1-c)\,\|\left(\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S}\right)^{-1}\|_F\cdot\|\mathbf{L_S}\|_F\cdot\|\left(\mathbf{I} - c\mathbf{A}\right)^{-1}\|_F
\end{aligned} \tag{4.32}$$

Since $\mathbf{A}$ is a symmetrically normalized matrix, its Laplacian matrix is $\mathbf{I}-\mathbf{A}$, which reveals that $\mathbf{I} - c\mathbf{A} = (1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}}$ and $\mathbf{I} - c\mathbf{A} + \alpha\mathbf{L_S} = (1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}+\alpha\mathbf{S}}$. Define $\mathbf{C} = (1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}}$ and $\mathbf{D} = (1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}+\alpha\mathbf{S}}$. Based on Equation (4.28), we have the following two inequalities holds.

$$\begin{aligned}
\|(\mathbf{D})^{-1}\|_F &\leq \sqrt{n}\sigma_{\max}\left(\left((1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}+\alpha\mathbf{S}}\right)^{-1}\right) \\
&= \frac{\sqrt{n}}{\sigma_{\min}\left((1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}+\alpha\mathbf{S}}\right)} = \frac{\sqrt{n}}{1-c}
\end{aligned} \tag{4.33}$$

$$\|\left(\mathbf{C}\right)^{-1}\|_F \leq \sqrt{n}\sigma_{\max}\left(\left((1-c)\,\mathbf{I} + \mathbf{L}_{c\mathbf{A}}\right)^{-1}\right) = \frac{\sqrt{n}}{1-c} \tag{4.34}$$

Combining Equations (4.33) and (4.34) with Equation (4.32), we have $\|\mathbf{r}^* - \bar{\mathbf{r}}\|_F \leq \frac{\alpha n}{1-c}\|\mathbf{L_S}\|_F$. As shown in Equation (4.30), we have $\|\mathbf{L_S}\|_F \leq 2\left(\delta + \sqrt{r\left(\mathbf{A}\right)}\sigma_{\max}\left(\mathbf{A}\right)\right)$. Thus, we have $\|\mathbf{r}^* - \bar{\mathbf{r}}\|_F \leq \frac{2\alpha n}{1-c}\left(\delta + \sqrt{r\left(\mathbf{A}\right)}\sigma_{\max}\left(\mathbf{A}\right)\right)$. QED.

Similar to the cost of debiasing the mining results, the cost of debiasing the mining model with PageRank depends on the regularization hyperparameter α, the number of nodes n, rank of adjacency matrix $r\left(\mathbf{A}\right)$, and the largest singular value $\sigma_{\max}\left(\mathbf{A}\right)$.

4.1.5 Experimental Evaluation

In this part, we perform experimental evaluations on InFoRM. The experiments are designed to answer the following questions:

- *RQ1.* How does the individual fairness constraint impact the graph mining performance?
- *RQ2.* How effective are the developed debiasing methods?
- *RQ3.* How efficient are the developed debiasing methods?

Table 4.4: Statistics of datasets to evaluate InFoRM.

Domain	Dataset	# Nodes	# Edges
Collaboration	AstroPh	18,772	198,110
	CondMat	23,133	93,497
Social	Facebook	22,470	171,002
	Twitch	7,126	35,324
Biology	PPI	3,890	76,584

Experimental settings. Here, we provide the detailed experimental settings to evaluate InFoRM.

A – Hardware and software specifications. All experiments are performed on a Windows PC with i7-9800X CPU and 64GB RAM. All datasets are publicly available. All codes are programmed in Python 3.7. The source code can be downloaded at `https://github.com/jiank2/inform`.

B – Dataset descriptions. Table 4.4 summarizes the statistics of the datasets. All datasets are undirected uni-partite graphs. We extract the largest connected components in these datasets for experiments in spectral clustering. The largest one in Table 4.4 is used to test the efficiency of the developed methods. These datasets are collected from various application-domains, including collaboration networks (*Collaboration*), social networks (*Social*), and biology network (*Biology*). We provide the detailed descriptions of these datasets as follows.

- *Collaboration networks.* In this type of networks, nodes usually represent researchers. Two researchers are connected if they have collaborated together. We use two collaboration networks in the field of Physics from arXiv preprint archive[25]: Astro Physics (*AstroPh*) and Condense Matter Physics (*CondMat*) [151].

- *Social networks.* Here, nodes are users, and edges indicate mutual social relationships. Among them, *Facebook* [151] is the page-page network of official Facebook pages, which is collected through Facebook Graph API in November, 2017. *Twitch* [151] is the user-user social network of gamers that streams in English on the popular game streaming website Twitch[26].

- *Biology network.* This domain includes the well-known *PPI* [151] network. It is a subgraph of the protein-protein interaction network for Homo Sapiens.

C – Baseline methods. We compare the performance of the debiased graph mining results with the original graph mining results without consideration of individual fairness.

[25] `https://arxiv.org/`
[26] `https://www.twitch.tv/`

D – Similarity matrix. For each dataset, we construct its node-node similarity $\mathbf{S}$ matrix by two different similarity measures: Jaccard index and cosine similarity. For PageRank and LINE, we filter out similarity links smaller than a pre-defined threshold. The threshold is defined as

$$threshold = \text{mean}(\mathbf{S}) + 0.75\text{std}(\mathbf{S}) \tag{4.35}$$

where mean $(\mathbf{S})$ and std $(\mathbf{S})$ calculate the mean and standard deviation of all non-zero elements in $\mathbf{S}$, respectively.

E – Evaluation metrics. To answer *RQ1*, we use two types of measures. First, we measure the difference between the original/vanilla mining results $\bar{\mathbf{Y}}$ and the debiased mining results $\mathbf{Y}^*$ as Diff $= \|\mathbf{Y}^* - \bar{\mathbf{Y}}\|_F / \|\bar{\mathbf{Y}}\|_F$. For PageRank, we also measure the KL divergence between $\bar{\mathbf{Y}}$ and $\mathbf{Y}^*$ (i.e., KL $\left(\frac{\mathbf{Y}^*}{\|\mathbf{Y}^*\|_1} || \frac{\bar{\mathbf{Y}}}{\|\bar{\mathbf{Y}}\|_1}\right) = \sum_i \frac{\mathbf{Y}^*[i]}{\|\mathbf{Y}^*\|_1} \log \frac{\mathbf{Y}^*[i]/\|\mathbf{Y}^*\|_1}{\bar{\mathbf{Y}}[i]/\|\bar{\mathbf{Y}}\|_1}$). We normalize $\bar{\mathbf{Y}}$ and $\mathbf{Y}^*$, since the norm may not equal to 1. Second, we also use a set of task-specific performance metrics. In detail, for PageRank, we use the precision (Prec) and the normalized discounted cumulative gain (NDCG). We label the top-K entities in the original PageRank as relevant ($rel = 1$) and others as irrelevant ($rel = 0$). Then, we calculate precision at K (Prec@$K = \frac{\#\text{ of relevant items}}{K}$) and NDCG at K (NDCG@$K = \sum_{i=1}^{K} \frac{rel}{\log(1+i)}$). For spectral clustering, we use the normalized mutual information (NMI) to measure the agreement between two cluster assignments before and after debiasing, which is defined as NMI $(\mathcal{C}, \mathcal{C}_0) = \frac{2\text{MI}(\mathcal{C},\mathcal{C}_0)}{H(\mathcal{C})+H(\mathcal{C}_0)}$, where $\mathcal{C}_0$ and $\mathcal{C}$ are the cluster assignments of original and debiased spectral clustering, MI $(\mathcal{C}, \mathcal{C}_0)$ is the mutual information between $\mathcal{C}$ and $\mathcal{C}_0$, and $H(\mathcal{C})$ is the entropy of assignment $\mathcal{C}$. For LINE, we perform link prediction using the debiased mining results and the original mining results and compare their F1 score and ROC-AUC score. To answer *RQ2*, we measure to what extent the individual bias is reduced as Reduction $= 1 - \frac{\text{Tr}\left((\mathbf{Y}^*)^T \mathbf{L_S} \mathbf{Y}^*\right)}{\text{Tr}\left(\bar{\mathbf{Y}}^T \mathbf{L_S} \bar{\mathbf{Y}}\right)}$. Finally, to answer *RQ3*, we measure the running time of each debiasing method (Time) in seconds.

F – Detailed parameter settings. To debias the input graph, for PageRank, we set $\alpha = 1 \times 10^6$ for PPI dataset, $\alpha = 5 \times 10^6$ for other datasets, and $\eta = 5 \times 10^{-4}$ for all datasets; for spectral clustering, we set $\alpha = 3 \times 10^5$, $\eta = 0.02$ for Twitch dataset and $\alpha = 1 \times 10^7$, $\eta = 0.05$ for PPI dataset; for LINE, we set $\alpha = 0.25$, $\eta = 0.5$ for Twitch dataset and $\alpha = 10$, $\eta = 0.025$ for PPI dataset. To debias the mining model and the mining results, we set $\alpha = 0.5$ for all mining tasks. Besides, for PageRank, we set its damping factor $c = 0.85$ and symmetrically normalize the adjacency matrix $\mathbf{A}$ and similarity matrix $\mathbf{S}$ to ensure the convergence of power iterations; for spectral clustering, we set the number of clusters as 10; for LINE, we randomly select 85% of all edges as training set, 5% as validation set, and 10% as test set. During model training, we sample $3200 \times n$ edges for each dataset, where n is the number of nodes, and use the same learning rate as in [5].

Table 4.5: Effectiveness results for PageRank. Lower is better in gray columns. Higher is better in the others.

Debiasing the Input Graph												
Dataset	**Jaccard Index**						**Cosine Similarity**					
	Diff	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**	**Diff**	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**
AstroPh	0.059	4.61×10^{-4}	0.840	0.887	16.3%	3632	0.117	1.99×10^{-3}	0.680	0.738	31.9%	3844
CondMat	0.008	1.06×10^{-5}	0.980	0.986	2.16%	1817	0.031	1.57×10^{-4}	0.940	0.957	9.37%	1922
Facebook	0.031	1.83×10^{-4}	0.920	0.943	7.01%	3442	0.072	9.38×10^{-4}	0.760	0.827	16.6%	3623
Twitch	0.109	5.37×10^{-4}	1.000	1.000	24.7%	564.9	0.299	5.41×10^{-3}	0.860	0.899	62.9%	649.3
PPI	0.185	1.90×10^{-3}	0.920	0.944	43.4%	584.4	0.328	8.07×10^{-3}	0.780	0.838	68.7%	636.8
Debiasing the Mining Model												
Dataset	**Jaccard Index**						**Cosine Similarity**					
	Diff	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**	**Diff**	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**
AstroPh	0.133	3.28×10^{-3}	0.820	0.871	51.0%	23.08	0.143	4.16×10^{-3}	0.880	0.912	50.4%	26.92
CondMat	0.117	2.43×10^{-3}	0.880	0.915	51.6%	12.02	0.149	4.01×10^{-3}	0.860	0.901	54.6%	12.83
Facebook	0.149	3.33×10^{-3}	0.840	0.884	47.7%	32.41	0.179	4.65×10^{-3}	0.840	0.883	53.3%	33.31
Twitch	0.182	4.97×10^{-3}	0.940	0.958	62.0%	16.18	0.315	1.05×10^{-2}	0.940	0.957	73.9%	12.73
PPI	0.211	4.78×10^{-3}	0.920	0.942	50.8%	10.76	0.280	9.56×10^{-3}	0.900	0.928	67.5%	10.50
Debiasing the Mining Results												
Dataset	**Jaccard Index**						**Cosine Similarity**					
	Diff	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**	**Diff**	**KL**	**Prec@50**	**NDCG@50**	**Reduction**	**Time**
AstroPh	0.055	1.40×10^{-3}	0.960	0.971	37.4%	0.038	0.094	4.46×10^{-3}	0.960	0.972	49.2%	0.054
CondMat	0.040	8.26×10^{-4}	0.940	0.959	34.4%	0.021	0.082	3.01×10^{-3}	0.780	0.839	48.9%	0.025
Facebook	0.047	1.12×10^{-3}	0.900	0.930	32.6%	0.048	0.086	3.87×10^{-3}	0.960	0.972	44.6%	0.062
Twitch	0.035	9.75×10^{-4}	0.980	0.986	33.9%	0.033	0.101	5.84×10^{-3}	0.940	0.958	44.6%	0.024
PPI	0.045	1.22×10^{-3}	0.940	0.958	27.0%	0.020	0.112	6.97×10^{-3}	0.940	0.958	45.0%	0.019

Table 4.6: Effectiveness results for LINE. Lower is better in gray columns. Higher is better in the others.

Debiasing the Input Graph														
Dataset	**Jaccard Index**							**Cosine Similarity**						
	Diff	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduce**	**Time**	**Diff**	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduce**	**Time**
Twitch	1.079	0.687	0.691	0.625	0.622	1.92%	1878	1.267	0.687	0.662	0.625	0.606	12.1%	1999
PPI	0.674	0.682	0.678	0.618	0.620	2.06%	1656	0.699	0.682	0.686	0.618	0.621	1.22%	1779
Debiasing the Mining Model														
Dataset	**Jaccard Index**							**Cosine Similarity**						
	Diff	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduction**	**Time**	**Diff**	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduction**	**Time**
AstroPh	0.462	0.973	0.970	0.924	0.914	51.6%	934.7	0.913	0.973	0.966	0.924	0.906	49.5%	923.0
CondMat	0.302	0.963	0.962	0.922	0.920	44.1%	1130	0.439	0.963	0.961	0.922	0.918	41.6%	1133
Facebook	0.323	0.946	0.954	0.888	0.902	49.6%	1099	0.442	0.946	0.957	0.888	0.906	56.0%	1100
Twitch	0.099	0.687	0.690	0.625	0.625	0.64%	333.8	0.152	0.687	0.694	0.625	0.628	0.83%	340.3
PPI	0.238	0.682	0.715	0.618	0.642	5.85%	180.3	0.418	0.682	0.740	0.618	0.669	7.71%	181.6
Debiasing the Mining Results														
Dataset	**Jaccard Index**							**Cosine Similarity**						
	Diff	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduction**	**Time**	**Diff**	**Orig. ROC**	**Fair ROC**	**Orig. F1**	**Fair F1**	**Reduction**	**Time**
AstroPh	0.365	0.973	0.962	0.924	0.898	83.3%	3.284	0.539	0.973	0.963	0.924	0.902	91.1%	6.461
CondMat	0.215	0.963	0.961	0.922	0.918	71.8%	1.464	0.322	0.963	0.960	0.922	0.915	78.4%	2.213
Facebook	0.304	0.946	0.950	0.888	0.890	88.5%	4.122	0.416	0.946	0.953	0.888	0.891	92.4%	7.394
Twitch	0.457	0.687	0.681	0.625	0.629	95.2%	2.320	0.603	0.687	0.658	0.625	0.616	97.6%	4.343
PPI	0.508	0.682	0.713	0.618	0.642	90.1%	1.031	0.722	0.682	0.634	0.618	0.589	97.0%	2.245

Table 4.7: Effectiveness results for spectral clustering. Lower is better in gray columns. Higher is better in the others.

Debiasing the Input Graph								
Dataset	**Jaccard Index**				**Cosine Similarity**			
	Diff	**NMI**	**Reduce**	**Time**	**Diff**	**NMI**	**Reduce**	**Time**
Twitch	0.031	1.000	5.44%	1698	0.107	1.000	24.5%	1714
PPI	1.035	0.914	19.5%	829.3	0.933	0.849	24.1%	985.1
Debiasing the Mining Model								
Dataset	**Jaccard Index**				**Cosine Similarity**			
	Diff	**NMI**	**Reduction**	**RT**	**Diff**	**NMI**	**Reduction**	**RT**
AstroPh	0.885	0.948	10.2%	333.9	1.085	0.868	23.6%	323.4
CondMat	1.108	0.856	26.4%	383.7	1.186	0.742	35.9%	360.7
Facebook	0.972	0.816	31.9%	549.3	0.897	0.810	37.9%	545.0
Twitch	1.147	0.838	88.3%	26.50	1.145	0.875	87.4%	26.62
PPI	0.994	0.658	67.0%	6.047	0.897	0.667	75.2%	6.244
Debiasing the Mining Results								
Dataset	**Jaccard Index**				**Cosine Similarity**			
	Diff	**NMI**	**Reduction**	**Time**	**Diff**	**NMI**	**Reduction**	**Time**
AstroPh	0.071	1.000	24.3%	10.22	0.123	0.984	39.5%	16.46
CondMat	0.071	1.000	34.5%	2.076	0.108	0.985	46.2%	3.196
Facebook	0.056	0.994	24.8%	8.425	0.102	0.994	35.9%	12.81
Twitch	0.150	1.000	90.9%	4.820	0.204	1.000	91.7%	6.513
PPI	0.242	0.811	77.5%	2.896	0.343	0.731	87.4%	4.288

Main results. The evaluation of PageRank is shown in Table 4.5. From the table, we can see that all three debiasing methods can effectively reduce the bias with small changes (i.e., Diff and KL columns in Table 4.5) to the vanilla mining results, while being able to preserve the performance (i.e., Prec@50 and NDCG@50) of the vanilla algorithm without fairness consideration. Comparing among these three methods, the debiasing the input graph method takes the longest running time. However, it is not as effective as the other two methods in terms of reducing the bias.[27] Thus, for spectral clustering and LINE, we mainly evaluate the efficacy on debiasing the mining model and debiasing the mining results, while evaluating the effectiveness only on relatively small-size *Twitch* and *PPI* datasets. Evaluation results for spectral clustering and LINE are shown in Tables 4.7 and 4.6, respectively. From these tables, we can see that our developed methods can effectively reduce bias and preserve the performance of the vanilla graph mining algorithm (i.e., Orig. ROC vs. Fair ROC, Orig. F1 vs. Fair F1 for LINE, and NMI for spectral clustering). Interestingly, as shown in Table 4.6, adding the fairness constraint on LINE sometimes actually improves the link prediction performance (e.g., on Facebook, Twitch, and PPI datasets).

[27]Algorithm 4.1 for PageRank still enjoys a linear complexity in big-O notation by exploring the low-rank structure of $\mathbf{H}$ matrix.

4.2 DEGREE FAIRNESS ON GRAPH CONVOLUTIONAL NETWORK

Graph-structured data naturally appears in many real-world scenarios, ranging from social network analysis [197], drug discovery [154], financial fraud detection [198] to traffic prediction [155], recommendation [199], and many more. The success of deep learning on grid-like data has inspired many graph neural networks in recent years. Among them, Graph Convolutional Network (GCN) [7] is one of the most fundamental and widely used ones, often achieving superior performance in a variety of tasks and applications.

Despite their strong expressive power in node/graph representation learning, recent studies show that GCN tends to under-represent nodes with low degrees [14], which could result in high loss values and low predictive accuracy in many tasks and applications. As shown in Figure 4.2, it is clear that low-degree nodes suffer from higher average loss and lower average accuracy in semi-supervised node classification. Such a performance disparity with respect to degrees is even more alarming, given that node degrees of real-world graphs often follow a long-tailed power-law distribution which means that a large fraction of nodes have low node degrees. In other words, the overall performance of GCN might be primarily beneficial to a few high-degree nodes (e.g., celebrities on a social media platform) but biased against a large number of low-degree nodes (e.g., grassroot users on the same social media platform).

To date, only a few efforts have been made to improve the performance for low-degree nodes. For example, DEMO-Net [64] randomly initializes degree-specific weight parameters, in order to preserve the neighborhood structure for low-degree nodes. SL-DSGCN [14] introduces a degree-specific weight generator based on recurrent neural network (RNN) and a semi-supervised learning module to provide pseudo labels for low-degree nodes. Recently, Tail-GNN [65] proposes a novel neighborhood translation mechanism to infer the missing local context information of low-degree nodes. However, the fundamental cause of degree-related unfairness in GCN largely remains unknown, which in turn prevents us from mitigating such unfairness from its root. Furthermore, existing works introduce either additional degree-specific weight parameters [14, 64] or additional operation on node representations (e.g., forged node generator, neighborhood translation) [65], which significantly increase the computational cost in both time and space and thus hinder the scalability of these models.

In order to tackle these limitations, we introduce the Rawlsian difference principle [69] to mitigate degree-related unfairness in GCN. As one of the earliest definitions of fairness from the theory of distributive justice [69], the Rawlsian difference principle aims to maximize the welfare of the least fortunate group and achieves stability when the worst-off group seeks to preserve its status quo. In the context of GCN, it requires a GCN model to have balanced performance among the groups of nodes with the same degree when the Rawlsian

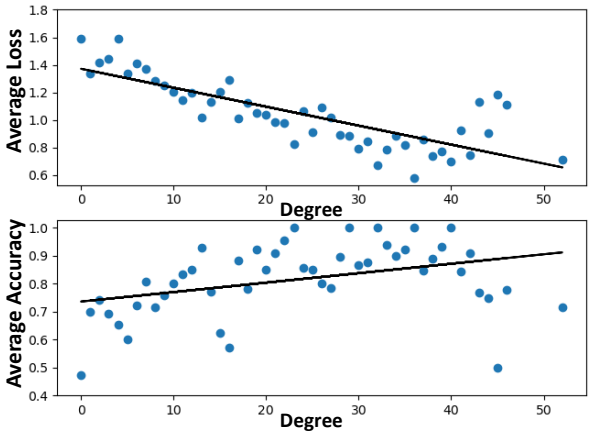


Figure 4.2: An illustrative example of the underrepresentation of low-degree nodes in semi-supervised node classification. A 2-layer GCN is trained on the Amazon-Photo dataset. Blue dots refers to the average loss and average accuracy of a specific degree group (i.e., the set of nodes with the same degree) in the top and bottom figures, respectively. Black lines are the regression lines of the blue dots in each figure. For visualization clarity, we only consider the degree groups that contain more than five nodes.

difference principle is in its stability. Given its essential role in training the GCN through backpropagation, we focus our analysis on the gradients of weight matrices of GCN. In particular, we establish the mathematical equivalence between the gradient of the weight matrix in a graph convolution layer and a weighted summation over the influence matrix of each node, weighted by its corresponding node degree in the input of GCN. This analysis not only reveals the root cause of degree-related unfairness in GCN, but also naturally leads to two new methods to mitigate the performance disparity with respect to node degrees, including (1) a pre-processing method named RawlsGCN-Graph that precomputes a doubly stochastic normalization of the input adjacency matrix to train the GCN and (2) an in-processing method named RawlsGCN-Grad that normalizes the gradient so that each node will have an equal importance in computing the gradients of weight matrices.

The main contributions of this work are summarized as follows.

- *Problem.* To our best knowledge, we are the first to introduce the Rawlsian difference principle to GCN so as to mitigate the degree-related unfairness.
- *Analysis.* We reveal the mathematical root cause of the degree-related unfairness in GCN.
- *Algorithms.* We develop two easy-to-implement methods to mitigate the degree bias, with no need to change the existing GCN architecture or introduce additional parameters.

Table 4.8: Table of symbols in RawlsGCN.

Symbol	Definition
$\mathbf{A}$	a matrix
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{u}$	a vector
$\mathcal{G}$	a graph
$\mathcal{V}$	a set of nodes
$J(\cdot)$	objective function
$\sigma(\cdot)$	activation function
$\mathbf{X}$	node feature matrix
$\mathbf{H}^{(l)}$	node representations at l-th layer
$\mathbf{W}^{(l)}$	weight matrix at l-th layer
L	number of graph convolution layers
d_l	hidden dimension of l-th layer
$\deg(u)$	degree of node u

- *Evaluations.* We perform extensive empirical evaluations on real-world graphs, which demonstrate that our developed methods (1) achieve comparable accuracy with the vanilla GCN, (2) significantly decrease the degree bias, and (3) take almost the same training time as the vanilla GCN. Surprisingly, by mitigating the bias of low-degree nodes, our methods can sometimes improve the overall classification accuracy by a significant margin.

4.2.1 Preliminaries and Problem Definition

In this part, we first introduce the main symbols used throughout this work in Table 4.8. Then we present a brief review on the Graph Convolutional Network (GCN) and the Rawlsian difference principle. Finally, we formally define the problem of enforcing the Rawlsian difference principle on GCN.

In RawlsGCN, unless otherwise specified, we denote matrices with bold upper-case letters (i.e., $\mathbf{A}$), vectors with bold lower-case letters (i.e., $\mathbf{x}$), and scalars with italic lower-case letters (i.e., c). We use rules similar to NumPy in Python for matrix and vector indexing. $\mathbf{A}[i, j]$ represents the entry of matrix $\mathbf{A}$ at the i-th row and the j-th column. $\mathbf{A}[i, :]$ and $\mathbf{A}[:, j]$ represent the i-th row and the j-th column of matrix $\mathbf{A}$, respectively. We use superscript ^{T} to represent the transpose of a matrix, i.e., $\mathbf{A}^T$ is the transpose of matrix $\mathbf{A}$.

Preliminaries. Here, we briefly review the Graph Convolutional Network and introduce the Rawlsian difference principle.

A – Graph Convolutional Network (GCN). We denote a graph as $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathbf{A}, \mathbf{X}\}$ where $\mathcal{V}_\mathcal{G}$

is the set of n nodes in the graph (i.e., $n = |\mathcal{V}_\mathcal{G}|$), $\mathbf{A}$ is the $n \times n$ adjacency matrix, and $\mathbf{X} \in \mathbb{R}^{n \times d_0}$ is the node feature matrix.

GCN is a typical graph neural network model that contains a stack of graph convolution layers. Based on the first-order Chebyshev polynomial, the graph convolution layer learns the latent node representations through the message-passing mechanism in two major steps. First, each node in the graph aggregates its own representation with the representations of its one-hop neighbors. Then, the aggregated representations are transformed through a fully-connected layer. Mathematically, for the l-th graph convolution layer, denoting its output node representations as $\mathbf{H}^{(l)}$ (we assume $\mathbf{H}^{(0)} = \mathbf{X}$ for notational consistency), it computes the latent representation with $\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \forall l \in \{1, \ldots, L\}$ where $\sigma(\cdot)$ is the nonlinear activation function, $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ is the weight matrix, and $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is the renormalized graph Laplacian with $\tilde{\mathbf{D}}$ being the diagonal degree matrix of $\mathbf{A} + \mathbf{I}$.

B – The Rawlsian difference principle. The Rawlsian difference principle is one of the major aspects of the equality principle in the theory of distributive justice by John Rawls [69]. The difference principle achieves equality by maximizing the welfare of the worst-off groups. When the Rawlsian difference principle is in its stability, the performance of all groups are balanced since there is no worst-off group whose welfare should be maximized, and all groups preserve their status quo. For a machine learning model that predicts task-specific labels for each data sample, the welfare is often defined as the predictive accuracy of the model [10, 200]. We denote (1) $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_h\}$ as a dataset that can be divided into h different groups, (2) $J(\mathcal{D}, \mathbf{Y}, \theta)$ as the task-specific loss function that the model with parameters θ aims to minimize where $\mathbf{Y}$ is the model output, and (3) $U(\cdot, \theta)$ as the utility function that measures the predictive accuracy over a set of samples using the model with parameters θ. The Rawlsian difference principle can be mathematically formulated as

$$\min_\theta \ \mathrm{Var}\left(\{U(\mathcal{D}_i, \theta) \,|\, i = 1, \ldots, h\}\right) \quad \text{s.t. } \theta = \operatorname{argmin} \ J(\mathcal{D}, \mathbf{Y}, \theta) \tag{4.36}$$

where $\mathrm{Var}\left(\{U(\mathcal{D}_i, \theta) \,|\, i = 1, \ldots, h\}\right)$ calculates the variance of the utilities of the groups $\{\mathcal{D}_1, \ldots, \mathcal{D}_h\}$.

Problem definition. Despite superior performance of GCN in many tasks, GCN is often biased towards benefiting high-degree nodes. Following the overarching difference principle by John Rawls, we view the inconsistent predictive accuracy of GCN for high-degree and low-degree nodes as a distributive justice problem. However, directly enforcing the Rawlsian difference principle (Equation (4.36)) is non-trivial for two major reasons. First (C1), in many real-world applications, there could be multiple utility measures of interest. For example, in a classification task, an algorithm administrator might be interested in different measures like

the classification accuracy, precision, recall, and F1 score. Even if only one utility measure is considered, it is likely that the measure itself is non-differentiable, which conflicts with the end-to-end training paradigm of GCN. Second (C2), it is hard to decide whether a node is low-degree or high-degree by a clear threshold of degree value. A bad choice of the threshold value could even introduce more bias in calculating the average utilities. For example, if we set the threshold to be too large, the group of low-degree nodes might contain relatively high-degree nodes on which GCN achieves high utility. Then its average utility will increase by including these relatively high-degree nodes. In this case, even when the GCN balances the utilities between the groups of low-degree nodes and high-degree nodes, many nodes with relatively low degrees still suffer from the issue of low predictive accuracy.

To address the first challenge (C1), we replace the utility function U with the loss function J as a proxy measure of predictive accuracy. The intuition lies in the design of the end-to-end training paradigm of GCN, in which we minimize the loss function in order to maximize the predictive accuracy of GCN. Thus, we aim to achieve a balanced loss in the stability of the Rawlsian difference principle. As for the second challenge (C2), instead of setting a hard threshold to split the groups of low-degree nodes and high-degree nodes, we split the node set $\mathcal{V} = \cup_{i=1}^{\deg_{\max}} \mathcal{V}_i$ to a maximum of $\deg_{\max}$ degree groups where $\mathcal{V}_i$ refers to the set of nodes whose degrees are equal to i. With that, we formally define the problem of enforcing the Rawlsian difference principle on GCN as follows.

Problem 4.4. Enforcing the Rawlsian difference principle on GCN.

Input: (1) an undirected graph $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathbf{A}, \mathbf{X}\}$, (2) an L-layer GCN with the set of weights θ, and (3) a task-specific loss function $J(\mathcal{G}, \mathbf{Y}, \theta)$ where $\mathbf{Y}$ is the model output.
Output: a well-trained GCN that (1) minimizes the task specific loss $J(\mathcal{G}, \mathbf{Y}, \theta)$ given the input graph $\mathcal{G}$ and (2) achieves a balanced loss for all degree groups $\mathcal{V}_i$ ($i = 1, \ldots, \deg_{\max}$).

4.2.2 Methodology

In this part, we present a family of algorithms, namely RawlsGCN, to enforce the Rawlsian difference principle on GCN. We first present analysis on the source of degree-related unfairness, which turns out to be rooted in the gradient of weight parameters in the GCN. Then we discuss how to compute doubly stochastic matrix, which is the key to mitigate the degree-related unfairness. Based on that, we present a pre-processing method (RawlsGCN-Graph) and an in-processing method (RawlsGCN-Grad) to solve Problem 4.4.
Source of unfairness. The key to solving Problem 4.4 is to understand why the loss of a GCN varies among nodes with different degrees after training. Since the key component

in training a GCN is the gradient matrix of the weight parameters with respect to the loss function, we seek to understand the root cause of such degree-related unfairness by analyzing it mathematically. In this section, our detailed analysis (Theorem 4.1) reveals the following fact: in a graph convolution layer, the gradient matrix $\frac{\partial J}{\partial \mathbf{W}}$ of the loss function J with respect to the weight parameter $\mathbf{W}$ is equivalent to a weighted summation of the influence matrix of each node,[28] weighted by its degree in input adjacency matrix $\hat{\mathbf{A}}$.

Theorem 4.1. (Source of degree unfairness). Suppose we have an input graph $\mathcal{G} = \{\mathcal{V}_{\mathcal{G}}, \mathbf{A}, \mathbf{X}\}$, the renormalized graph Laplacian $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}} (\mathbf{A} + \mathbf{I}) \tilde{\mathbf{D}}^{-\frac{1}{2}}$, a nonlinear activation function $\sigma(\cdot)$, and an L-layer GCN that minimizes a task-specific loss function J. For any l-th hidden graph convolution layer ($\forall l \in \{1, \ldots, L\}$), the gradient of the loss function J with respect to the weight parameter $\mathbf{W}^{(l)}$ is a linear combination of the influence of each node weighted by its degree in the renormalized graph Laplacian.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{j=1}^{n} \deg_{\hat{\mathbf{A}}}(j) \mathbf{I}_j^{(\text{row})} = \sum_{i=1}^{n} \deg_{\hat{\mathbf{A}}}(i) \mathbf{I}_i^{(\text{col})} \tag{4.37}$$

where $\deg_{\hat{\mathbf{A}}}(i)$ is the degree of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$, $\mathbf{I}_j^{(\text{row})}$ is the row-wise influence matrix of node j of the form $\left(\mathbf{H}^{(l-1)}[j,:]\right)^T \mathbb{E}_{i \sim p_{\hat{N}(j)}} \left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right]$, $\mathbf{I}_i^{(\text{col})}$ is the column-wise influence matrix of node i of the form $\left(\mathbb{E}_{j \sim p_{\hat{N}(i)}} \left[\mathbf{H}^{(l-1)}[j,:]\right]\right)^T \frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}$, $\mathbf{H}^{(l-1)}$ is the input node embeddings of the hidden layer, and $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$ is the node embeddings before the nonlinear activation.

Proof. To compute the derivative of the objective function J with respect to the weight $\mathbf{W}^{(l)}$ in the l-th graph convolution layer, by the graph convolution $\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})$ and the chain rule of matrix derivative, we have

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}[i,j]} = \sum_{a=1}^{n} \sum_{b=1}^{d_l} \frac{\partial J}{\partial \mathbf{H}^{(l)}[a,b]} \frac{\partial \mathbf{H}^{(l)}[a,b]}{\partial \mathbf{W}^{(l)}[i,j]} \tag{4.38}$$

where d_l is the number of columns in $\mathbf{H}^{(l)}$. To compute Equation (4.38), a key term is $\frac{\partial \mathbf{H}^{(l)}[a,b]}{\partial \mathbf{W}^{(l)}[i,j]}$. Denoting σ' as the derivative of the activation function σ, by the graph convolution, we get

$$\begin{aligned} \frac{\partial \mathbf{H}^{(l)}[a,b]}{\partial \mathbf{W}^{(l)}[i,j]} &= \frac{\partial \sigma\left((\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]\right)}{\partial (\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]} \frac{\partial (\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]}{\partial \mathbf{W}^{(l)}[i,j]} \\ &= \sigma'\left((\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})[a,b]\right) (\hat{\mathbf{A}}\mathbf{H}^{(l-1)})[a,i] \mathbb{1}[b=j] \end{aligned} \tag{4.39}$$

[28] We use J to represent $J(\mathcal{G}, \mathbf{Y}, \theta)$ for notational simplicity.

Combining Equations (4.38) and (4.39), we have

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^{(l)}[i,j]} &= \sum_a \left(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\right)^T[i,a]\left(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\right)[a,j] \\ &= \left(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\right)^T[i,:]\left(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\right)[:,j]\end{aligned} \tag{4.40}$$

where $\circ$ represents the element-wise product. Writing Equation (4.40) into matrix form, we have

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = (\mathbf{H}^{(l-1)})^T\hat{\mathbf{A}}^T\left(\frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\right) \tag{4.41}$$

Let $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$ denote the node embeddings before the nonlinear activation, i.e., $\mathbf{H}^{(l)} = \sigma(\mathbf{E}^{(l)})$. By the chain rule of matrix derivative, we have $\frac{\partial J}{\partial \mathbf{E}^{(l)}} = \frac{\partial J}{\partial \mathbf{H}^{(l)}} \circ \sigma'(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})$. Then Equation (4.41) can be written as[29]

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = (\mathbf{H}^{(l-1)})^T\hat{\mathbf{A}}^T\frac{\partial J}{\partial \mathbf{E}^{(l)}} \tag{4.42}$$

To analyze the influence of each node on the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$, we factorize Equation (4.42) as follows.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\mathbf{A}}^T[i,j]\left(\mathbf{H}^{(l-1)}[i,:]\right)^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[j,:]} \tag{4.43}$$

Denoting the distribution $p_{\hat{\mathcal{N}}(i)}$ over the neighborhood of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$ such that $p_{\hat{\mathcal{N}}(i)}(j) \propto \hat{\mathbf{A}}[i,j] = \hat{\mathbf{A}}[j,i], \forall j \sim p_{\hat{\mathcal{N}}(i)}$, we can re-write Equation (4.43) as

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}^{(l)}} &= \sum_{j=1}^{n}\deg_{\hat{\mathbf{A}}}(j)\left(\mathbf{H}^{(l-1)}[j,:]\right)^T\mathbb{E}_{i\sim p_{\hat{\mathcal{N}}(j)}}\left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right] \\ &= \sum_{i=1}^{n}\deg_{\hat{\mathbf{A}}}(i)\left(\mathbb{E}_{j\sim p_{\hat{\mathcal{N}}(i)}}\left[\mathbf{H}^{(l-1)}[j,:]\right]\right)^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\end{aligned} \tag{4.44}$$

where $\deg_{\hat{\mathbf{A}}}(i) = \sum_{i=1}^{n}\hat{\mathbf{A}}[i,j] = \sum_{i=1}^{n}\hat{\mathbf{A}}[j,i]$ is the degree of node i in the renormalized graph Laplacian $\hat{\mathbf{A}}$. We define the row-wise influence matrix $\mathbf{I}_j^{(\text{row})}$ of a node j and the column-wise influence matrix $\mathbf{I}_i^{(\text{col})}$ of a node i as follows.

$$\mathbf{I}_j^{(\text{row})} = (\mathbf{H}[j,:])^T\mathbb{E}_{i\sim p_{\hat{\mathcal{N}}(j)}}\left[\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]}\right] \quad \mathbf{I}_i^{(\text{col})} = \left(\mathbb{E}_{j\sim p_{\hat{\mathcal{N}}(i)}}[\mathbf{H}[j,:]]\right)^T\frac{\partial J}{\partial \mathbf{E}^{(l)}[i,:]} \tag{4.45}$$

Then Equation (4.44) can be written as the weighted summation over the (row-/column-wise)

[29] A simplified result on linear GCN without nonlinear activation is shown in [201], while our result (Equation (4.42)) generalizes to GCN with *arbitrary* differentiable nonlinear activation function.

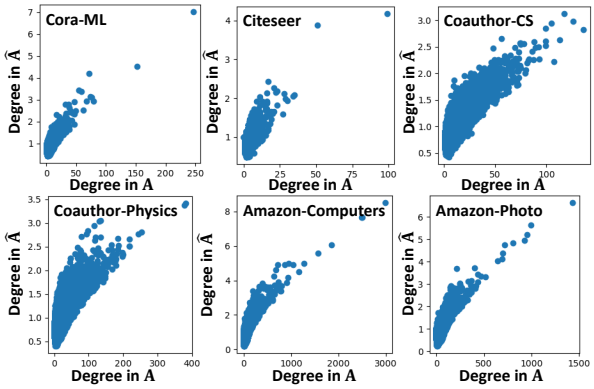


Figure 4.3: Node degrees in the original adjacency matrix $\mathbf{A}$ (x-axis) vs. the corresponding node degrees in the renormalized graph Laplacian $\hat{\mathbf{A}}$ (y-axis).

influence matrix of each node, weighted by its corresponding degree in the renormalized graph Laplacian $\hat{\mathbf{A}}$.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \sum_{j=1}^{n} \deg_{\hat{\mathbf{A}}}(j)\, \mathbf{I}_j^{(\text{row})} = \sum_{i=1}^{n} \deg_{\hat{\mathbf{A}}}(i)\, \mathbf{I}_i^{(\text{col})} \tag{4.46}$$

QED.

Remark. By Theorem 4.1, the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ is essentially equivalent to a linear combination of the influence matrix of each node in the graph, with the corresponding node degree $\deg_{\hat{\mathbf{A}}}$ in the renormalized graph Laplacian $\hat{\mathbf{A}}$ serving as the importance of the influence matrix. As a consequence, as long as the node degrees are not equal to a constant (e.g., 1), the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ will favor the nodes with higher degrees in $\hat{\mathbf{A}}$. It is noteworthy that, even if $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ is symmetrically normalized, such normalization only guarantees the largest eigenvalue of $\hat{\mathbf{A}}$ to be 1, whereas the degrees in $\hat{\mathbf{A}}$ are *not* constant as shown in Figure 4.3. From the figure, we have two key observations: (1) the node degrees in $\hat{\mathbf{A}}$ are not equal among all nodes (y-axis); and (2) there is a positive correlation between a node degree in $\mathbf{A}$ (x-axis) and a node degree in $\hat{\mathbf{A}}$ (y-axis). Putting everything together, it means that the higher the node degree in $\mathbf{A}$, the more importance it has on the gradient of the weight matrix. This shows exactly why the vanilla GCN favors high-degree nodes while being biased against low-degree nodes.

Doubly stochastic matrix computation. In order to mitigate the node degree-related unfairness, the key idea is to normalize the importance of influence matrices (i.e., the node degrees in $\hat{\mathbf{A}}$) to 1 in Equation (4.46), such that each node will have equal importance in updating the weight parameters. To achieve that, Equation (4.46) naturally requires that

the rows and columns of $\hat{\mathbf{A}}$ sum up to 1, which is a doubly stochastic matrix.

Computing the doubly stochastic matrix is non-trivial. To achieve that, we adopt the Sinkhorn-Knopp algorithm, which is an iterative algorithm to balance a matrix into doubly stochastic form [202]. Mathematically speaking, given a non-negative $n \times n$ square matrix $\mathbf{A}$, it aims to find two $n \times n$ diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ such that $\mathbf{D}_1\mathbf{A}\mathbf{D}_2$ is doubly stochastic. The intuition of the Sinkhorn-Knopp algorithm is to learn a sequence of matrices whose rows and columns are alternatively normalized. Mathematically, defining $\mathbf{r}_0 = \mathbf{c}_0 = \mathbf{1}$ to be the column vectors of all 1s and $\mathbf{x}^{-1}$ to be the operator for element-wise reciprocal, i.e., $\mathbf{x}^{-1}[i] = 1/\mathbf{x}[i]$, the Sinkhorn-Knopp algorithm alternatively calculates the following equations.

$$\mathbf{c}_{k+1} = \left(\mathbf{A}^T\mathbf{r}_k\right)^{-1} \qquad \mathbf{r}_{k+1} = \left(\mathbf{A}\mathbf{c}_{k+1}\right)^{-1} \tag{4.47}$$

If the algorithm converges after K iterations, the doubly stochastic form of $\mathbf{A}$ is $\mathbf{P} = \text{diag}(\mathbf{r}_K)\mathbf{A}\text{diag}(\mathbf{c}_K)$ where $\text{diag}(\mathbf{r}_K)$ and $\text{diag}(\mathbf{c}_K)$ diagonalize the column vectors $\mathbf{r}_K$ and $\mathbf{c}_K$ into diagonal matrices. The time and space complexities of computing the doubly stochastic matrix $\mathbf{P}$ is linear with respect to the size of input matrix $\mathbf{A}$.

Lemma 4.4. (Time and space complexities of the Sinkhorn-Knopp algorithm). Let $\mathbf{A}$ be an $n \times n$ non-negative matrix with m non-zero elements, the time complexity of the Sinkhorn-Knopp algorithm is $O(K(m+n))$ where K is the number of iterations to convergence. It takes an additional $O(m+n)$ space.

Proof. In the k-th iteration (where $1 \leq k \leq K$) of the Sinkhorn-Knopp algorithm, it takes $O(m)$ time to compute $\mathbf{A}^T\mathbf{r}_k$ and $\mathbf{A}\mathbf{c}_{k+1}$. Then it takes $O(n)$ time for element-wise reciprocal. Thus, the time complexity of an iteration is $O(m+n)$. Since the algorithm takes K iterations to converge, the overall time complexity is $O(K(m+n))$. Regarding the space complexity, it takes an additional $O(n)$ space to store vectors $\mathbf{c}_k$ and $\mathbf{r}_k$ in the k-th iteration and an additional $O(m)$ time to store the resulting doubly stochastic form of matrix $\mathbf{A}$. Thus, the overall space complexity is $O(m+n)$. QED.

Next, we prove the convergence of the Sinkhorn-Knopp algorithm in our setting. To this end, we first present the definition of the diagonal of a matrix corresponding to a column permutation.

Definition 4.3. (Diagonal of a matrix corresponding to a column permutation [202]). Let $\mathbf{A}$ be an $n \times n$ square matrix and δ be a permutation over the set $\{1, \ldots, n\}$.

(1) The sequence of elements $\mathbf{A}[1, \delta(1)], \mathbf{A}[2, \delta(2)], \ldots, \mathbf{A}[n, \delta(n)]$ is called the diagonal of $\mathbf{A}$ corresponding to δ.

(2) If δ is the identity, the diagonal is the main diagonal of $\mathbf{A}$.

(3) If $\mathbf{A}\left[i, \delta\left(i\right)\right] > 0, \forall i$, the diagonal is a positive diagonal.

We then provide the formal definition of the support of a matrix in Definition 4.4.

Definition 4.4. (Support of a non-negative square matrix [202]). Let $\mathbf{A}$ be an $n \times n$ non-negative matrix. $\mathbf{A}$ is said to have support if $\mathbf{A}$ contains a positive diagonal. $\mathbf{A}$ is said to have total support if $\mathbf{A} \neq \mathbf{0}$ and if every positive element of $\mathbf{A}$ lies on a positive diagonal, where $\mathbf{0}$ is the zero matrix of the same size as $\mathbf{A}$.

By Definitions 4.3 and 4.4, Sinkhorn and Knopp [202] proved the following result.

Theorem 4.2. (Sinkhorn-Knopp theorem [202]). If $\mathbf{A}$ is an $n \times n$ non-negative matrix, the Sinkhorn-Knopp algorithm converges and finds the unique doubly stochastic matrix of the form $\mathbf{D}_1\mathbf{A}\mathbf{D}_2$ if and only if $\mathbf{A}$ has total support, where $\mathbf{D}_1$ and $\mathbf{D}_2$ are diagonal matrices.

Proof. Omitted. QED.

Based on Theorem 4.2, we give Lemma 4.5 which says that the Sinkhorn-Knopp algorithm always converges in our setting and finds the doubly stochastic matrix with respect to the renormalized graph Laplacian $\hat{\mathbf{A}}$.

Lemma 4.5. (Convergence of the Sinkhorn-Knopp algorithm on renormalized graph Laplacian). Given an adjacency matrix $\mathbf{A}$, if $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\left(\mathbf{A} + \mathbf{I}\right)\tilde{\mathbf{D}}^{-\frac{1}{2}}$ with $\tilde{\mathbf{D}}$ as the degree matrix of $\mathbf{A}+\mathbf{I}$, the Sinkhorn-Knopp algorithm always converges to find the unique doubly stochastic form of $\hat{\mathbf{A}}$.

Proof. The key idea is to prove that $\hat{\mathbf{A}}$ has total support. Let $\deg_{\hat{\mathbf{A}}}\left(i\right)$ be the degree of node i in $\hat{\mathbf{A}}$. It is trivial that $\hat{\mathbf{A}}$ has support because its main diagonal is positive. In order to prove that $\hat{\mathbf{A}}$ has total support, for any undirected edge $\mathbf{A}\left[i, j\right]$, we define a column permutation δ_{ij} as a permutation that satisfies (1) $\delta_{ij}\left(i\right) = j$, (2) $\delta_{ij}(j) = i$, and (3) $\delta_{ij}(k) = k, \forall k \neq i$ and $k \neq j$. Then, for any edge (i, j), by applying the permutation δ_{ij}, the diagonal of $\hat{\mathbf{A}}$ corresponding to δ_{ij} is a positive diagonal due to the non-negativity of $\hat{\mathbf{A}}$. Thus, $\hat{\mathbf{A}}$ has total support because all positive elements of $\hat{\mathbf{A}}$ lie on positive diagonals, which completes the proof. QED.

Lemma 4.5 guarantees that we can always calculate the doubly stochastic form of $\hat{\mathbf{A}}$ using the Sinkhorn-Knopp algorithm, in order to ensure the equal importance of node influence in calculating the gradient of the weight parameter $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ in the l-th layer.

RawlsGCN algorithms. If the gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ is computed using the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ with respect to the renormalized graph Laplacian $\hat{\mathbf{A}}$, it is fair with respect to node degrees because all nodes will have equal importance in determining the gradient, i.e., their degrees in $\hat{\mathbf{A}}_{DS}$ are all equal to 1. Thus, a fair gradient with respect to node degrees can be calculated using Equation (4.42) as follows.

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}} = \left(\mathbf{H}^{(l-1)}\right)^T \hat{\mathbf{A}}_{DS}^T \frac{\partial J}{\partial \mathbf{E}^{(l)}} \tag{4.48}$$

where $\mathbf{E}^{(l)} = \hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}$.

Observing the computation of fair gradient in Equation (4.48), it naturally leads to two methods to mitigate the degree-related unfairness, including (1) a pre-processing method named RawlsGCN-Graph which utilizes the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ as the input adjacency matrix and (2) an in-processing method named RawlsGCN-Grad that normalizes the gradient in GCN with Equation (4.48).

A – Method #1: Pre-processing with RawlsGCN-*Graph.* If we are allowed to modify the input of the GCN whereas the model itself are fixed, we can precompute the input renormalized graph Laplacian $\hat{\mathbf{A}}$ into its doubly stochastic form $\hat{\mathbf{A}}_{DS}$ and feed $\hat{\mathbf{A}}_{DS}$ as the input of the GCN. With that, the gradient computed using Equation (4.42) is equivalent to Equation (4.48). As a consequence, the Rawlsian difference principle is naturally ensured since all nodes in $\hat{\mathbf{A}}_{DS}$ have the same degree. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ and an L-layer GCN, RawlsGCN-Graph adopts the following 3-step strategy.

1. Precompute $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\mathbf{A} + \mathbf{I}$.
2. Precompute $\hat{\mathbf{A}}_{DS}$ by applying the Sinkhorn-Knopp algorithm on $\hat{\mathbf{A}}$.
3. Input $\hat{\mathbf{A}}_{DS}$ and $\mathbf{X}$ to the GCN for model training.

B – Method #2: In-processing with RawlsGCN-*Grad.* If we have access to the model or the model parameters while the input data is fixed, we can precompute the doubly stochastic matrix $\hat{\mathbf{A}}_{DS}$ and use $\hat{\mathbf{A}}_{DS}$ to compute the fair gradient by Equation (4.48). Then training GCN with the fair gradient ensures the Rawlsian difference principle because nodes of different degrees share the same importance in determining the gradient for gradient descent-based optimization. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathbf{A}, \mathbf{X}\}$ and an L-layer GCN, the general workflow of RawlsGCN-Grad is as follows.

1. Precompute $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$ where $\tilde{\mathbf{D}}$ is the diagonal degree matrix of $\mathbf{A} + \mathbf{I}$.
2. Precompute $\hat{\mathbf{A}}_{DS}$ by applying the Sinkhorn-Knopp algorithm on $\hat{\mathbf{A}}$.

3. Input $\hat{\mathbf{A}}$ and $\mathbf{X}$ to the GCN for model training.

4. For each graph convolution layer $l \in \{1, \ldots, L\}$, compute the fair gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}}$ for each weight matrix $\mathbf{W}^{(l)}$ using Equation (4.48).

5. Update the model parameters $\mathbf{W}^{(l)}$ using the fair gradient $\frac{\partial J}{\partial \mathbf{W}^{(l)}}_{\text{fair}}$.

6. Repeat steps 4 – 5 until the model converges.

An advantage of both RawlsGCN-Graph and RawlsGCN-Grad is that no additional time complexity will be incurred during the learning process of GCN, since we can precompute and store $\hat{\mathbf{A}}_{DS}$. For RawlsGCN-Graph, $\hat{\mathbf{A}}_{DS}$ has the same number of non-zero elements as $\hat{\mathbf{A}}$ because computing $\hat{\mathbf{A}}_{DS}$ is essentially rescaling each non-zero element in $\hat{\mathbf{A}}$. Thus, there is no additional time cost during the graph convolution operation with $\hat{\mathbf{A}}_{DS}$. For RawlsGCN-Grad, computing the fair gradient with Equation (4.48) enjoys the same time complexity as the vanilla gradient computation (Equation (4.42)) because $\hat{\mathbf{A}}_{DS}$ and $\hat{\mathbf{A}}$ has exactly the same number of non-zero entries. Thus, there is no additional time cost in the big-O notation in optimizing the GCN parameters. In terms of the additional costs in computing $\hat{\mathbf{A}}_{DS}$, it bears a linear time and space complexities with respect to the number of nodes and the number of edges in the graph as stated in Lemma 4.4.

4.2.3 Experimental Evaluation

In this section, we evaluate our developed RawlsGCN methods in the task of semi-supervised node classification to answer the following questions:

- *RQ1.* How accurate are the RawlsGCN methods in node classification?
- *RQ2.* How effective are the RawlsGCN methods in debiasing?
- *RQ3.* How efficient are the RawlsGCN methods in time and space?

Experimental settings. We present the detailed experimental settings for RawlsGCN.
A – Hardware and software specifications. All datasets are publicly available. All codes are programmed in Python 3.8.5 and PyTorch 1.9.0. All experiments are performed on a Linux server with 2 Intel Xeon Gold 6240R CPUs at 2.40 GHz and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory. The source code of RawlsGCN can be found at `https://github.com/jiank2/rawlsgcn`.
B – Dataset descriptions. We utilize six publicly available real-world networks for evaluation. Their statistics, including the number of nodes, the number of edges, number of node features,

Table 4.9: Statistics of datasets to evaluate RawlsGCN.

Dataset	# Nodes	# Edges	# Features	# Classes	Median Deg.
Cora-ML	2,995	16,316	2,879	7	3
Citeseer	3,327	9,104	3,703	6	2
Coauthor-CS	18,333	163,788	6,805	15	6
Coauthor-Physics	34,493	495,924	8,415	5	10
Amazon-Computers	13,752	491,722	767	10	22
Amazon-Photo	7,650	238,162	745	8	22

number of classes, and the median of node degrees (Median Deg.), are summarized in Table 4.9. For semi-supervised node classification, we use a fixed random seed to generate the training/validation/test sets for each network. The training set contains 20 nodes per class. The validation set and the test set contain 500 nodes and 1000 nodes, respectively.

C – Baseline methods. We compare RawlsGCN with several baseline methods, including GCN [7], DEMO-Net [64], DSGCN [14], Tail-GNN [65], Adversarial Fair GCN (AdvFair) [47], and REDRESS [59].[30]

- *GCN* [7] refers to the original Graph Convolutional Network (GCN) without fairness considerations. In our experiments, we adopt the same architecture as in [7] but increasing the hidden dimension to 64 for a fair comparison.

- *DEMO-Net* [64] uses multi-task graph convolution where each task learns degree-specific node representations in order to preserve the degree-specific graph structure. We use DEMO-Net with degree-specific weight function instead of hashing function due to its higher classification accuracy and better stability. For a fair comparison, we remove the components for order-free and seed-oriented representation learning as they are irrelevant to fairness with respect to node degree.

- *DSGCN* [14] mitigates degree-related bias by degree-specific graph convolution, which infers the degree-specific weights using a recurrent neural network (RNN). We use 2 degree-specific graph convolution layers in DSGCN with the same hidden dimension settings as the vanilla GCN. We set the number of RNN cell to 10 (i.e., nodes with degree larger than 10 will share the same degree-specific weight), which is consistent with [14]. We set the activation function of the RNN cell to tanh function. For a fair comparison, we only use the degree-specific graph convolution module (i.e., DSGCN in [14]) for our experiments.

[30]We use the official PyTorch implementation of GCN, Tail-GNN, and REDRESS for experimental evaluation. For DEMO-Net, we implement our own PyTorch version and consult with the original authors for a sanity check. For DSGCN, we implement our own PyTorch version due to the lack of publicly available implementation.

- *Tail-GNN* [65] learns robust embedding for low-degree nodes (i.e., tail nodes) in an adversarial learning fashion with the novel neighborhood translation mechanism. It first generates forged tail nodes from nodes with degree higher than a certain threshold k. Then the neighborhood translation operation predicts the missing information of tail nodes and forged tail nodes by a translation model learned from head nodes. After that, a discriminator is applied to predict whether a node is head or tail based on the node representations. In our experiments, we set $k = 5$ for forged tail nodes generation. If a training node u has degree less than k, we do not generate the forged tail node using this training node.

- *Adversarial Fair GCN* (AdvFair) is a variant of [47] which ensures group fairness for graph embeddings in the compositional setting (i.e., for different combinations of sensitive attributes). We set the node degree as the sensitive attribute, i.e., nodes of the same degree form a demographic group. For a fair comparison, we compute the node embeddings using 2 graph convolution layers with ReLU activation, each of which has 64 hidden dimension. The 'filtered' embeddings are computed by the filter, which is a 2-layer multi-layer perceptron (MLP) with 128 and 64 hidden dimensions, respectively. The discriminator is a 2-layer MLP where the first layer contains 64 hidden dimensions and the second layer predicts the sensitive attribute of each node. Both the filter and the discriminator use leaky ReLU as the activation function. A multi-class logistic regression is applied on the 'filtered' embeddings for node classification.

- *REDRESS* [59] ensures individual fairness of graph neural network (GNN) by optimizing the similarity between the ranking lists of model input and output. In our experiments, we set the backbone GNN model as the vanilla GCN model described above.

D – Evaluation metrics. We use cross entropy as the loss function in semi-supervised node classification. To answer *RQ1*, we evaluate the accuracy of node classification (i.e., Acc. in Table 4.10). For metrics in *RQ2*, we define the bias with respect to the Rawlsian difference principle as the variance of degree-specific average cross entropy (i.e., AvgCE) to be consistent with the definition of Problem 4.4. Mathematically, it is defined as

$$\begin{aligned} \text{AvgCE}(k) &= \mathbb{E}\left[\{\text{CE}(u), \forall \text{ node } u \text{ such that } \deg(u) = k\}\right] \\ \textit{Bias} &= \text{Var}\left(\{\text{AvgCE}(k), \forall \text{ node degree } k\}\right) \end{aligned} \tag{4.49}$$

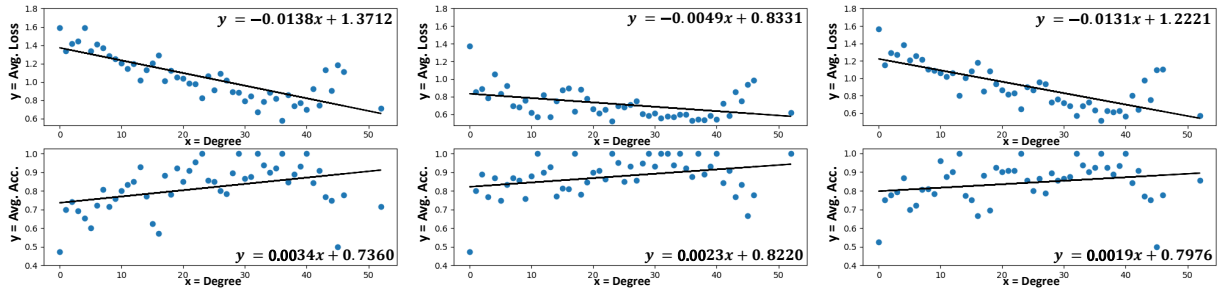
where $\text{CE}(u)$ and $deg(u)$ are the cross entropy and the degree of node u, respectively. To measure the efficiency (*RQ3*), we count the number of learnable parameters (# Param. in Table 4.11), GPU memory usage in MB (Memory in Table 4.11), and training time in seconds.

Table 4.10: Effectiveness for node classification. Lower is better for bias (in gray). Higher is better for accuracy (Acc., in percentage).

Method	Cora-ML		Citeseer		Coauthor-CS	
	Acc.	**Bias**	**Acc.**	**Bias**	**Acc.**	**Bias**
GCN	80.10 ± 0.812	0.392 ± 0.046	68.60 ± 0.341	0.353 ± 0.040	93.28 ± 0.194	0.075 ± 0.004
DEMO-Net	61.60 ± 0.687	0.181 ± 0.015	60.26 ± 0.408	0.315 ± 0.022	65.90 ± 0.583	0.164 ± 0.006
DSGCN	30.26 ± 5.690	8.003 ± 2.766	31.42 ± 3.257	6.887 ± 1.947	44.20 ± 7.155	1.460 ± 0.397
Tail-GNN	78.54 ± 0.582	0.503 ± 0.284	66.34 ± 0.009	0.655 ± 0.382	92.66 ± 0.196	0.052 ± 0.031
AdvFair	67.56 ± 2.594	10.01 ± 2.480	50.26 ± 6.277	3.146 ± 2.425	84.82 ± 2.254	12.26 ± 6.797
REDRESS	75.70 ± 0.620	0.955 ± 0.213	65.80 ± 0.518	0.944 ± 0.077	92.44 ± 0.233	0.028 ± 0.003
RawlsGCN-Graph (Ours)	76.96 ± 1.098	0.105 ± 0.012	69.34 ± 0.745	0.196 ± 0.013	92.52 ± 0.264	0.043 ± 0.002
RawlsGCN-Grad (Ours)	79.34 ± 1.247	0.232 ± 0.065	68.81 ± 0.462	0.283 ± 0.047	92.68 ± 0.240	0.058 ± 0.007
Method	**Coauthor-Physics**		**Amazon-Computers**		**Amazon-Photo**	
	Acc.	**Bias**	**Acc.**	**Bias**	**Acc.**	**Bias**
GCN	93.96 ± 0.367	0.023 ± 0.001	64.84 ± 0.641	0.353 ± 0.026	79.58 ± 1.507	0.646 ± 0.038
DEMO-Net	77.50 ± 0.566	0.084 ± 0.010	26.48 ± 3.455	0.456 ± 0.021	39.92 ± 1.242	0.243 ± 0.013
DSGCN	79.08 ± 1.533	0.262 ± 0.075	27.68 ± 1.663	1.407 ± 0.685	26.76 ± 3.387	0.921 ± 0.805
Tail-GNN	OOM	OOM	76.24 ± 1.491	1.547 ± 0.670	86.00 ± 2.715	0.471 ± 0.264
AdvFair	87.44 ± 1.132	0.892 ± 0.502	53.50 ± 5.362	4.395 ± 1.102	75.80 ± 3.563	51.24 ± 39.94
REDRESS	94.48 ± 0.172	0.019 ± 0.001	80.36 ± 0.206	0.455 ± 0.032	89.00 ± 0.369	0.186 ± 0.030
RawlsGCN-Graph (Ours)	94.06 ± 0.196	0.016 ± 0.000	80.16 ± 0.859	0.121 ± 0.010	88.58 ± 1.116	0.071 ± 0.006
RawlsGCN-Grad (Ours)	94.18 ± 0.306	0.021 ± 0.002	74.18 ± 2.530	0.195 ± 0.029	83.70 ± 0.672	0.186 ± 0.068

E – Detailed parameter settings. For all methods, we use a 2-layer GCN as the backbone model with the hidden dimension as 64. We evaluate all methods on 5 different runs and report their average performance. For the purpose of the reproducibility, the random seeds for these 5 runs are varied from 0 to 4. We use the Adam [182]optimizer to train all methods. Unless otherwise specified, the default weight decay of the optimizer is set to 0.0005. Regarding the learning rate, for RawlsGCN-Graph, RawlsGCN-Grad, and Adversarial Fair GCN, we search the learning rate that achieves the highest average classification accuracy in the set of $\{0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025\}$. For DSGCN, due to its long running time, we search the learning rate in the set of $\{0.05, 0.025, 0.01, 0.005\}$. For REDRESS, we search the best choice of α (see details in [59]) in the range of $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Regarding the number of training epochs, we train RawlsGCN models for 100 epochs without early stopping. For Adversarial Fair GCN, we train the model for 1000 epochs with a patience of 200 for early stopping. For DSGCN, due to long training time, we train the model for 200 epochs without early stopping, which is consistent with the settings in GCN. For all other parameter settings for GCN, DEMO-Net, Tail-GNN, and REDRESS, we use the suggested hyperparameters (including learning rate, weight decay, number of epochs, and early stopping conditions) in the released source code.

Effectiveness results. The evaluation results are shown in Table 4.10. We do not report the results of Tail-GNN for the Coauthor-Physics dataset due to the out-of-memory (OOM) error. From the table, we can see that RawlsGCN-Graph and RawlsGCN-Grad are



(a) GCN. (b) RawlsGCN-Graph. (c) RawlsGCN-Grad.

Figure 4.4: Visualization on how our developed RawlsGCN algorithms improve the performance of low-degree nodes on the Amazon-Photo dataset. (a) shows the results for vanilla GCN. (b) shows the results for RawlsGCN-Graph. (c) shows the results for RawlsGCN-Grad. Similar to Figure 4.2, blue dots refer to the average loss (Avg. Loss) and average accuracy (Avg. Acc.) of a specific degree group in the top and bottom figures, respectively. Black lines are the regression lines of the blue dots in each figure. For a more clear visualization, we only consider the degree groups which contain more than five nodes.

the only two methods that can consistently reduce the bias across all datasets. Though REDRESS reduces bias more than our methods on the Coauthor-CS dataset, it bears a much higher bias than our methods in other datasets. Surprisingly, on the Amazon-Computers and Amazon-Photo datasets, RawlsGCN-Graph and RawlsGCN-Grad significantly improve the overall classification accuracy by mitigating the bias of low-degree nodes. This is because, compared with the vanilla GCN, the classification accuracy of low-degree nodes are increased while the classification accuracy of high-degree nodes are largely retained, resulting in the significantly improved overall accuracy.

In Figure 4.4, we visualize how RawlsGCN-Graph and RawlsGCN-Grad benefit the low-degree nodes and balance the performance between low-degree nodes and high-degree nodes. From the figure, we observe that both RawlsGCN-Graph (Figure 4.4b) and RawlsGCN-Grad (Figure 4.4c) show lower average loss values and higher average classification accuracy compared with GCN (Figure 4.4a). Moreover, to visualize how our methods balance the performance between low-degree nodes and high-degree nodes, we perform linear regression on the average loss and average accuracy with respect to node degree. From the figure, we can see that the slopes of the regression lines for RawlsGCN-Graph and RawlsGCN-Grad are flatter than the slopes of the regression line in GCN.

Efficiency results. We measure the memory and time consumption of all methods in Table 4.11. In the table, GCN (100 epochs) and GCN (200 epochs) denote the GCN models trained with 100 epochs and 200 epochs, respectively. From the table, we have two key observations. (1) Compared with other baseline methods, RawlsGCN-Graph and

Table 4.11: Efficiency of training a 2-layer GCN on the Amazon-Photo dataset. Lower is better for all columns. GPU memory usage (Memory) is measured in MB. Training time is measured in seconds.

Method	**# Param.**	**Memory**	**Training Time**
GCN (100 epochs)	48,264	1,461	13.335
GCN (200 epochs)	48,264	1,461	28.727
DEMO-Net	11,999,880	1,661	9158.5
DSGCN	181,096	2,431	2714.8
Tail-GNN	2,845,567	2,081	94.058
AdvFair	89,280	1,519	148.11
REDRESS	48,264	1,481	291.69
RawlsGCN-Graph (Ours)	48,264	1,461	11.783
RawlsGCN-Grad (Ours)	48,264	1,461	12.924

Table 4.12: Ablation study of different matrix normalization techniques. Lower is better for bias (i.e., the gray column). Higher is better for accuracy (Acc.).

Method	**Normalization**	**Cora-ML**		**Citeseer**		**Coauthor-CS**	
		Acc.	**Bias**	**Acc.**	**Bias**	**Acc.**	**Bias**
RawlsGCN-Graph	Row	79.74 ± 0.320	0.098 ± 0.004	69.18 ± 0.595	0.240 ± 0.013	92.78 ± 0.331	0.052 ± 0.002
	Column	76.78 ± 1.360	0.260 ± 0.330	69.12 ± 0.781	0.243 ± 0.093	92.44 ± 0.609	0.049 ± 0.012
	Symmetric	77.04 ± 1.606	0.109 ± 0.015	69.20 ± 0.735	0.196 ± 0.014	92.56 ± 0.120	0.042 ± 0.001
	Doubly Stochastic	76.98 ± 1.098	0.105 ± 0.012	69.34 ± 0.745	0.196 ± 0.013	92.52 ± 0.264	0.043 ± 0.002
RawlsGCN-Grad	Row	79.78 ± 0.349	0.230 ± 0.017	68.64 ± 0.215	0.274 ± 0.036	92.92 ± 0.440	0.069 ± 0.006
	Column	79.94 ± 0.599	0.253 ± 0.077	68.48 ± 0.204	0.302 ± 0.049	92.78 ± 0.407	0.058 ± 0.006
	Symmetric	79.68 ± 0.458	0.199 ± 0.008	68.68 ± 0.248	0.286 ± 0.042	93.00 ± 0.341	0.063 ± 0.006
	Doubly Stochastic	79.34 ± 1.247	0.232 ± 0.065	68.81 ± 0.462	0.283 ± 0.047	92.68 ± 0.240	0.058 ± 0.007
Method	**Normalization**	**Coauthor-Physics**		**Amazon-Computers**		**Amazon-Photo**	
		Acc.	**Bias**	**Acc.**	**Bias**	**Acc.**	**Bias**
RawlsGCN-Graph	Row	94.36 ± 0.488	0.013 ± 0.000	78.54 ± 1.125	0.092 ± 0.013	87.98 ± 0.791	0.076 ± 0.006
	Column	93.98 ± 0.508	0.016 ± 0.003	78.18 ± 4.354	0.196 ± 0.106	88.32 ± 2.315	0.138 ± 0.112
	Symmetric	93.98 ± 0.248	0.016 ± 0.000	80.22 ± 0.803	0.126 ± 0.012	89.12 ± 0.945	0.071 ± 0.005
	Doubly Stochastic	94.06 ± 0.196	0.016 ± 0.000	80.16 ± 0.859	0.121 ± 0.010	88.58 ± 1.116	0.071 ± 0.006
RawlsGCN-Grad	Row	94.08 ± 0.204	0.027 ± 0.001	63.46 ± 1.376	0.453 ± 0.039	82.86 ± 1.139	0.852 ± 0.557
	Column	94.26 ± 0.294	0.020 ± 0.002	75.48 ± 1.273	0.218 ± 0.033	84.96 ± 1.235	0.221 ± 0.064
	Symmetric	94.30 ± 0.346	0.021 ± 0.001	66.42 ± 0.584	0.353 ± 0.021	82.92 ± 1.121	0.744 ± 0.153
	Doubly Stochastic	94.18 ± 0.306	0.021 ± 0.002	74.18 ± 2.530	0.195 ± 0.029	83.70 ± 0.672	0.186 ± 0.068

RawlsGCN-Grad have fewer number of parameters to learn and are much more efficient in memory consumption. (2) RawlsGCN-Graph and RawlsGCN-Grad bear almost the same training time as the vanilla GCN with 100 epochs of training (i.e., GCN (100 epochs)) while all other baseline methods significantly increase the training time.

Ablation study. To evaluate the effectiveness of the doubly stochastic normalization on the renormalized graph Laplacian, we compare it with three other normalization methods, including row normalization, column normalization, and symmetric normalization. From Table 4.12, we observe that, although row normalization and symmetric normalization outperform the doubly stochastic normalization in some cases, it can also increase the bias in other cases, e.g., Amazon-Computers and Amazon-Photo. Meanwhile, for Amazon-Photo

dataset, while all these normalization methods lead to similar accuracy (within 2% difference), doubly stochastic normalization leads to a much smaller bias than others. All in all, the doubly stochastic normalization is the best one that (1) consistently mitigates bias for both RawlsGCN-Graph and RawlGCN-Grad and (2) achieves good balance between accuracy and fairness.

Table 4.13: Pros and cons of RawlsGCN-Graph and RawlsGCN-Grad.

	RawlsGCN-Graph	**RawlsGCN-Grad**
Pros	• No need to modify the GNN model. • Higher accuracy on graph with more diverse degree empirically. • Smaller bias than RawlsGCN-Grad.	• Higher accuracy on graph with less diversity in node degree empirically. • Able to work in use cases like distributed training of large graphs.
Cons	• Lower accuracy on smaller graph/-graph with less diversity in node degree empirically. • May be unable to work in use cases like distributed training on extremely large graphs.	• Slightly higher bias than RawlsGCN-Graph. • Need to change the optimizer of model.

4.2.4 Discussion: RawlsGCN-Graph vs. RawlsGCN-Grad

Finally, we discuss the advantages and disadvantages of RawlsGCN. We list the pros and cons of RawlsGCN in Table 4.13. Moreover, though RawlsGCN-Graph and RawlsGCN-Grad have the same pre-processing procedure in the setting of fixed input graph, we believe that the general idea of normalizing the gradient in RawlsGCN-Grad is useful for distributed training of extremely large graphs, in which a local subgraph of each node is often sampled using a (non-)deterministic sampler for feature aggregation and gradient computation. In this setting, the input graph is not deterministic during training and often asymmetric. Consequently, it is often impossible to precompute the doubly stochastic matrix for RawlsGCN-Graph. However, we can still use the sampling distribution of local subgraphs to calculate the normalized gradient using Equations (4.45) and (4.46).

4.3 GROUP FAIRNESS WITH MULTIPLE SENSITIVE ATTRIBUTES

The increasing amount of data and computational power have empowered machine learning algorithms to play crucial roles in automated decision-making for a variety of real-world applications, including credit scoring [203], criminal justice [204], and healthcare analysis [205]. As the application landscape of machine learning continues to broaden and deepen, so does the concern regarding the potential, often unintentional, bias it could introduce or amplify. For example, recent media coverage has revealed that a well-trained image generator could turn a low-resolution picture of a black man into a high-resolution image of a white man due to the skewed data distribution that causes the model to disfavor the minority group;[31] and another article highlighted an automated credit card application system assigning a dramatically higher credit limit to a man than to his female partner, even though his partner has a better credit history.[32]

As such, algorithmic fairness, which aims to mitigate unintentional bias caused by automated learning algorithms, has become increasingly important. To date, researchers have proposed a variety of fairness notions [40, 41]. Among them, one of the most fundamental notions is *group fairness*.[33] Generally speaking, to ensure group fairness, the first step is to partition the entire population into a few demographic groups based on a pre-defined sensitive attribute (e.g., gender). Then the fair learning algorithm will enforce parity of a certain statistical measure among those demographic groups. Group fairness can be instantiated with many statistical notions of fairness. Statistical parity [206] enforces the learned classifier to accept equal proportion of population from the pre-defined majority group and minority group. Likewise, disparate impact [40] ensures the acceptance rate for the minority group should be no less than four-fifth of that for the group with the highest acceptance rate, which is analogous to the famous 'four-fifth' rule in the legal support area [207]. In addition, equalized odds and equal opportunity [186] are used to enforce the classification accuracies to be equal across all demographic groups conditioned on ground-truth outcomes or positively labeled populations, respectively. The vast majority of the existing works in group fairness primarily focus on debiasing with respect to a single sensitive attribute. However, it is quite common for multiple sensitive attributes (e.g., gender, race, marital status, etc.) to co-exist in a real-world application. We ask: *would a debiasing algorithm designed to ensure the group fairness for a particular sensitive attribute (e.g., marital status) unintentionally amplify the*

[31] `https://shorturl.at/fHIQ5`

[32] `https://www.nytimes.com/2019/11/10/business/Apple-credit-card-investigation.html`

[33] An orthogonal work in algorithmic fairness is individual fairness. Although it promises fairness by 'treating similar individuals similarly' in principle, it is often hard to be operationalized in practice due to its strong assumption on the distance metrics and data distributions.

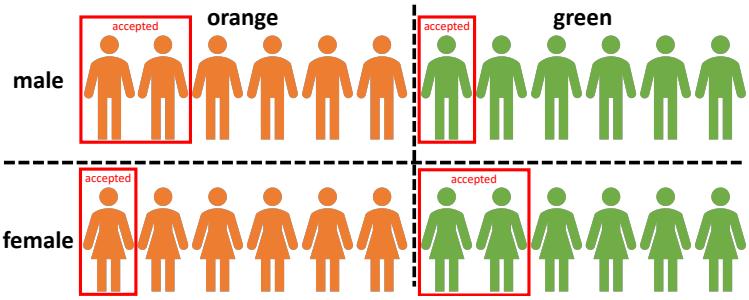


Figure 4.5: An illustrative example of bias in job application classification when considering multiple sensitive attributes. Rows indicate gender (e.g., male vs. female) and columns indicate race (e.g., orange vs. green).[34]Boxed individuals receive job offers. If we consider gender or race alone, statistical parity is enforced due to the equal acceptance rate. However, when considering gender and race (i.e., forming finer-grained gender-race groups), the classification result is biased in the fine-grained gender-race groups. This is because the acceptance rates in two fine-grained groups (i.e., male-green group and female-orange) are lower than that of the two other fine-grained groups (i.e., male-orange and female-green).

group bias with respect to another sensitive attribute (e.g., gender)? If so, how can we ensure a fair learning outcome with respect to all sensitive attributes of concern simultaneously?

Existing works for answering these questions [40, 47, 206, 208] have two major limitations. The first limitation is that some existing works could only debias multiple *distinct* sensitive attributes [47], which fails to mitigate bias on the fine-grained groups formed by multiple sensitive attributes. Figure 4.5 provides an illustrative example of the difference between fairness with respect to multiple distinct sensitive attributes and fairness among fine-grained groups of multiple sensitive attributes. The second limitation is that the optimization problems behind some existing works are often subject to surrogate constraints of statistical parity [40, 206, 208] instead of directly optimizing statistical parity itself, resulting in unstable performance on bias mitigation unless the learned models could perfectly model the relationship between the training data and the ground-truth outcomes.

In this paper, we tackle these two limitations by studying the problem of *information-theoretic intersectional fairness* (InfoFair), which aims to directly enforce statistical parity on multiple sensitive attributes simultaneously. Though our focused fairness notion is statistical parity, the developed method can be generalized to other statistical fairness notions (e.g., equalized odds and equal opportunity) with minor modifications. The key idea in solving the InfoFair problem is to consider all sensitive attributes of interest as a vectorized sensitive attribute in order to partition the demographic groups and then minimize the dependence between the learning outcomes and this vectorized attribute. More specifically, we measure

[34]We use imaginary race groups to avoid potential offenses.

Table 4.14: Table of symbols in InfoFair.

Symbol	Definition
$\mathcal{D}$	a set
$\mathbf{W}$	a matrix
$\mathbf{h}$	a vector
$\mathbf{h}[i]$	the i-th element in $\mathbf{h}$
$\Pr(\cdot)$	the probability of an event happening
$p_{\cdot,\cdot}$	joint distribution of two random variables
$p_{\cdot}$	marginal distribution of a random variable
$H(\cdot)$	entropy
$H(\cdot\|\cdot)$	conditional entropy
$I(\cdot;\cdot)$	mutual information

the dependence using mutual information originated in information theory [209]. Building upon it, we formulate the InfoFair problem as an optimization problem regularized on mutual information minimization.

The main contributions of this work are as follows.

- *Problem definition.* We formally define the problem of information-theoretic intersectional fairness and formulate it as an optimization problem, where the key idea is to minimize both the task-specific loss function (e.g., cross-entropy loss in classification) and mutual information between the learning outcomes and the vectorized sensitive attribute.
- *End-to-end algorithmic framework.* We introduce a novel end-to-end bias mitigation framework named InfoFair by optimizing a variational representation of mutual information. The developed framework is extensible and capable of solving any learning task with a gradient-based optimizer.
- *Empirical evaluations.* We perform empirical evaluations in the fair classification task on three real-world datasets. The evaluation results demonstrate that our developed framework can effectively mitigate bias with little sacrifice in the classification accuracy.

4.3.1 Preliminaries and Problem Definition

In this part, we present a table of the main symbols used in this work. Then we briefly review the concepts of statistical parity and mutual information, as well as their relationships. Finally, we formally define the problem of information-theoretic intersectional fairness.

In this work, matrices are denoted by bold uppercase letters (e.g., $\mathbf{X}$), vectors are denoted by bold lowercase letters (e.g., $\mathbf{y}$), scalars are denoted by italic lowercase letters (e.g., c), and sets are denoted by calligraphic letters (e.g., $\mathcal{D}$). We use superscript ^{T} to denote transpose (e.g., $\mathbf{h}^T$ is the transpose of $\mathbf{h}$) and superscript $\mathcal{C}$ to denote the complement of a set (e.g., set $\mathcal{D}^{\mathcal{C}}$ is the complement of set $\mathcal{D}$). We use a convention similar to Numpy in Python for vector indexing (e.g., $\mathbf{h}[i]$ is the i-th element in vector $\mathbf{h}$).

Preliminaries. We briefly introduce statistical parity and mutual information, both of which are essential to study information-theoretic intersectional fairness.

A – Statistical parity is one of the most intuitive and widely-used group fairness notions. Given a set of data points $\mathcal{X}$, their corresponding labels $\mathbf{y}$ and a sensitive attribute s, classification with statistical parity aims to learn a classifier to predict outcomes that (1) are as accurate as possible with respect to $\mathbf{y}$ and (2) do not favor one group over another with respect to s. Mathematically, statistical parity is defined as follows.

Definition 4.5. (Statistical parity [206]). Suppose we have (1) a population $\mathcal{X}$, (2) a hypothesis $h : \mathcal{X} \to \{0, 1\}$ which assigns a binary label to individual x drawn from $\mathcal{X}$, and (3) a sensitive attribute which splits the population $\mathcal{X}$ into majority group $\mathcal{M}$ and minority group $\mathcal{M}^{\mathcal{C}}$ (i.e., $\mathcal{X} = \mathcal{M} \cup \mathcal{M}^{\mathcal{C}}$). An individual x is accepted if $h(x) = 1$ and rejected if $h(x) = 0$. The hypothesis $h : \mathcal{X} \to \{0, 1\}$ is said to have statistical parity on the population $\mathcal{X}$ as long as

$$\Pr[h(x) = 1 | x \in \mathcal{M}] = \Pr\left[h(x) = 1 | x \in \mathcal{M}^{\mathcal{C}}\right] \tag{4.50}$$

where $\Pr[\cdot]$ denotes the probability of an event happening.

Many methods have been proposed to achieve statistical parity. For example, Zemel et al. [210] learn fair representation by regularizing the difference in expected positive rate for majority and minority groups. Zhang, Lemoine and Mitchell [211] propose an adversarial learning-based framework for fair classification, in which the output of the predictor is used to predict the sensitive attribute by the adversary. Kearns et al. [208] propose a learner-auditor framework to enforce subgroup fairness through fictitious play strategy.

B – Mutual information was first introduced in the 1940s [209]. Given two random variables, mutual information measures the dependence between them by quantifying the amount of information in bits obtained on one random variable through observing the other one. Let (x, y) be a pair of random variables x and y. Suppose their joint distribution is $p_{x,y}$ and the marginal distributions are p_x and p_y. The mutual information between x and y is defined as

$$I(x; y) = H(x) - H(x|y) = \int_x \int_y p_{x,y} \log \frac{p_{x,y}}{p_x p_y} dx dy \tag{4.51}$$

where $H(x) = -\int_x p_x \log p_x dx$ is the entropy of x and $H(x|y) = -\int_x \int_y p_{x,y} \log p_{x|y} dxdy$ is the conditional entropy of x given y. Unlike correlation coefficients (e.g., Pearson's correlation coefficient) which could only capture the linear dependence between two random variables, mutual information is more general in capturing both the linear and nonlinear dependences between two random variables. We have $I(x;y) = 0$ if and only if two random variables x and y are independent to each other.

According to Lemma 4.6, there is an equivalence between statistical parity and zero mutual information.

Lemma 4.6. (Equivalence between statistical parity and zero mutual information [210, 212]). Statistical parity requires a sensitive attribute to be statistically independent to the learning results, which is equivalent to zero mutual information. Mathematically, given a learning outcome $\tilde{\mathbf{y}}$ and the sensitive attribute s, we have

$$\underbrace{p_{\tilde{\mathbf{y}}|s} = p_{\tilde{\mathbf{y}}}}_{\text{statistical parity}} \Leftrightarrow p_{\tilde{\mathbf{y}},s} = p_{\tilde{\mathbf{y}}} p_s \Leftrightarrow \underbrace{I(\tilde{\mathbf{y}}; s) = 0}_{\text{zero mutual information}} \tag{4.52}$$

Proof. Omitted. QED.

Information-theoretic intersectional fairness. In order to generalize Lemma 4.6 from a single sensitive attribute to a set of sensitive attributes $\mathcal{S} = \left\{s^{(1)}, \ldots, s^{(k)}\right\}$, we first introduce the concept of vectorized sensitive attribute $\mathbf{s}$ given $\mathcal{S}$. We define the vectorized sensitive attribute $\mathbf{s} = [s^{(1)}, \ldots, s^{(k)}]$ as a multi-dimensional random variable, where each element of $\mathbf{s}$ represents the corresponding sensitive attribute in $\mathcal{S}$ (e.g., $\mathbf{s}[i] = s^{(i)}$ is the i-th sensitive attribute). Based on that, we have the following equivalence. For notational simplicity, we denote $I\left(\tilde{\mathbf{y}}; s^{(1)}, \ldots, s^{(k)}\right)$, $p_{\tilde{\mathbf{y}}, s^{(1)}, \ldots, s^{(k)}}$, and $p_{s^{(1)}, \ldots, s^{(k)}}$ with $I(\tilde{\mathbf{y}}; \mathbf{s})$, $p_{\tilde{\mathbf{y}},\mathbf{s}}$, and $p_{\mathbf{s}}$, respectively.

$$p_{\tilde{\mathbf{y}}|\mathbf{s}} = p_{\tilde{\mathbf{y}}} \Leftrightarrow p_{\tilde{\mathbf{y}},\mathbf{s}} = p_{\tilde{\mathbf{y}}} p_{\mathbf{s}} \Leftrightarrow I(\tilde{\mathbf{y}}; \mathbf{s}) = 0 \tag{4.53}$$

Based on Equation (4.53), we formally define the problem of information-theoretic intersectional fairness as a mutual information minimization problem.

Problem 4.5. InfoFair: Information-theoretic intersectional fairness

Input: (1) a set of k sensitive attributes $\mathcal{S} = \left\{s^{(1)}, \ldots, s^{(k)}\right\}$, (2) a set of n data points $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{s}_i, y_i) | i = 1, \ldots, n\}$ where $\mathbf{x}_i$ is the feature vector of the i-th data point, y_i is its label, and $\mathbf{s}_i = \left[s_i^{(1)}, \ldots, s_i^{(k)}\right]$ describes the vectorized sensitive attributes on $\mathcal{S}$ of the i-th data point (with $s_i^{(j)}$ being the corresponding attribute value of the j-th sensitive attribute $s^{(j)}$), and (3) a learning algorithm represented by $l(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta)$, where l is the loss function,

and $\tilde{\mathbf{y}}^* = \text{argmin}_{\tilde{\mathbf{y}}} l(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta)$ is the optimal learning outcome on the input data with θ being model parameters.

Output: a set of revised learning outcomes $\{\tilde{\mathbf{y}}^*\}$ which minimizes (1) the empirical risk $\mathbb{E}_{(\mathbf{x},\mathbf{s},y)\sim\mathcal{D}}[l(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta)]$ and (2) the expectation of mutual information between the learning outcomes and the sensitive attributes $\mathbb{E}_{(\mathbf{x},\mathbf{s},y)\sim\mathcal{D}}[I(\tilde{\mathbf{y}}; \mathbf{s})]$.

Remark. A byproduct of InfoFair is that the statistical parity can also be achieved on any subset of sensitive attributes included in $\mathcal{S}$, which is summarized in Lemma 4.7. This could be particularly useful in that the algorithm administrator does not need to re-train the model in order to obtain fair learning results if s/he is only interested in a subset of available sensitive attributes.

Lemma 4.7. (Statistical parity on the subset of sensitive attributes). Consider statistical parity as the fairness notion, and suppose we are given a learning outcome $\tilde{\mathbf{y}}$, a set of k sensitive attributes $\mathcal{S} = \left\{s^{(1)}, \ldots, s^{(k)}\right\}$, and the vectorized sensitive attribute $\mathbf{s} = \left[s^{(1)}, \ldots, s^{(k)}\right]$. If $\tilde{\mathbf{y}}$ is fair with respect to $\mathbf{s}$, $\tilde{\mathbf{y}}$ is fair with respect to any vectorized sensitive attribute $\mathbf{s}_{\text{sub}}$ induced from the subset of sensitive attributes $\mathcal{S}_{\text{sub}} \subseteq \mathcal{S} = \left\{s^{(1)}, \ldots, s^{(k)}\right\}$.

Proof. Omitted. QED.

4.3.2 Methodology

In this part, we present a generic end-to-end algorithmic framework named InfoFair for information-theoretic intersectional fairness. We first formulate the problem as a mutual information minimization problem and then present a variational representation of mutual information. Based on that, we present the InfoFair framework to solve the optimization problem, followed by discussions on generalizations and variants of the InfoFair framework.

Objective function. Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{s}_i, y_i) | i = 1, \ldots, n\}$, the InfoFair problem (Problem 4.5) can be naturally formulated as minimizing the following objective function,

$$J = \mathbb{E}_{(\mathbf{x},\mathbf{s},y)\sim\mathcal{D}}[l(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta) + \alpha I(\tilde{\mathbf{y}}; \mathbf{s})] \tag{4.54}$$

where l is a task-specific loss function for a learning task, θ is the model parameter, $\tilde{\mathbf{y}}$ is the learning outcome, and $\alpha > 0$ is the regularization hyperparameter. An example of loss function l is the negative log likelihood

$$l(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta) = -\log \tilde{\mathbf{y}}[y] \tag{4.55}$$

where y is the class label, and $\tilde{\mathbf{y}}$ denotes the probabilities of being classified into the corresponding class.

To optimize the above objective function, a key challenge lies in optimizing the mutual information $I(\tilde{\mathbf{y}}; \mathbf{s})$ between the learning outcome $\tilde{\mathbf{y}}$ and the vectorized sensitive feature $\mathbf{s}$. Inspired by the seminal work of Belghazi et al. [213], a natural choice would be to apply off-the-shelf mutual information estimation methods for high-dimensional data. Examples include MINE [213], Deep Infomax [214], and CCMI [215], which estimate mutual information by parameterizing neural networks to maximize tight lower bounds of mutual information. However, in a mutual information minimization problem like Equation (4.54), it is often counter-intuitive to maximize a lower bound of mutual information. Though one could still maximize the objective function of these estimators to estimate the mutual information and use such estimation to guide the optimization of Equation (4.54) as a minimax game, it is hindered by two hurdles. First, it requires learning a well-trained estimator to estimate the mutual information during each epoch of optimizing Equation (4.54). Second, if the estimator is not initialized with proper parameter settings, mutual information may be poorly estimated, which could further result in failing to find a good saddle point in such a minimax game.

Variational representation of mutual information. We take a different strategy from MINE and other similar methods by deriving a variational representation of mutual information $I(\tilde{\mathbf{y}}; \mathbf{s})$. Our variational representation leverages a variational distribution of the vectorized sensitive feature $\mathbf{s}$ given the learning outcome $\tilde{\mathbf{y}}$ (Lemma 4.8).

Lemma 4.8. (Variational representation of mutual information). Suppose the joint distribution of the learning outcome $\tilde{\mathbf{y}}$ and the vectorized sensitive feature $\mathbf{s}$ is $p_{\tilde{\mathbf{y}},\mathbf{s}}$, and the marginal distributions of $\tilde{\mathbf{y}}$ and $\mathbf{s}$ are $p_{\tilde{\mathbf{y}}}$ and $p_{\mathbf{s}}$, respectively. Mutual information $I(\tilde{\mathbf{y}}; \mathbf{s})$ between $\tilde{\mathbf{y}}$ and $\mathbf{s}$ is as follows.

$$I(\tilde{\mathbf{y}}; \mathbf{s}) = H(\mathbf{s}) + \mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}}}\left[\log q_{\mathbf{s}|\tilde{\mathbf{y}}}\right] + \mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}}}\left[\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}}\right] \tag{4.56}$$

where $q_{\mathbf{s}|\tilde{\mathbf{y}}}$ is the variational distribution of $\mathbf{s}$ given $\tilde{\mathbf{y}}$.

Proof. Omitted. QED.

Next, we minimize the variational representation shown in Lemma 4.8, which contains three terms: (1) the entropy $H(\mathbf{s})$, (2) the expectation of log likelihood $\mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}}}\left[\log q_{\mathbf{s}|\tilde{\mathbf{y}}}\right]$, and (3) the expectation of log density ratio $\mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}}}\left[\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}}\right]$. For the first term $H(\mathbf{s})$, we assume it to be a constant term, which can be ignored in the optimization stage. The rationale

behind our assumption is that, in most (if not all) use cases, the vectorized sensitive feature $\mathbf{s}$ relates to the demographic information of an individual (e.g., gender, race, marital status, etc.), which should remain unchanged during the learning process. Then the remaining key challenges lie in (C1) calculating $\log q_{\mathbf{s}|\tilde{\mathbf{y}}}$ and (C2) estimating $\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}}$. The intuition of C1 and C2 is that we strive to find a learning outcome $\tilde{\mathbf{y}}$ such that (1) $\tilde{\mathbf{y}}$ fails to predict the vectorized sensitive feature $\mathbf{s}$ (referring to C1), while (2) making it hard to distinguish if the vectorized sensitive feature $\mathbf{s}$ is generated from the variational distribution or sampled from the original distribution (referring to C2).

C1 – Calculating $\log q_{\mathbf{s}|\tilde{\mathbf{y}}}$. It can be naturally formulated as a prediction problem, where the input is the learning outcome $\tilde{\mathbf{y}}$, and the output is the probability of $\mathbf{s}$ being predicted. To solve it, we parameterize a decoder $f(\tilde{\mathbf{y}}; \mathbf{s}; \mathbf{W})$ (e.g., a neural network) as a sensitive feature predictor to 'reconstruct' $\mathbf{s}$, where $\mathbf{W}$ is the learnable parameters in the decoder.

$$\log q_{\mathbf{s}|\tilde{\mathbf{y}}} = \log f(\tilde{\mathbf{y}}; \mathbf{s}; \mathbf{W}) \tag{4.57}$$

For categorical sensitive attribute, $\log q_{\mathbf{s}|\tilde{\mathbf{y}}}$ refers to the log likelihood of classifying $\tilde{\mathbf{y}}$ into label $\mathbf{s}$, which can be interpreted as the negative of cross-entropy loss of the decoder $f(\tilde{\mathbf{y}}; \mathbf{s}; \mathbf{W})$. Moreover, if $\mathbf{s}$ contains multiple categorical sensitive attributes, solving Equation (4.57) requires solving a multi-label classification problem, which itself is not trivial to solve. In this case, we further reduce it to a single-label problem by applying a mapping function $\text{map}(\cdot)$ to map the multi-hot encoding $\mathbf{s}$ into a one-hot encoding $\hat{\mathbf{s}}$ (i.e., $\hat{\mathbf{s}} = \text{map}(\mathbf{s})$).

C2 – Estimating $\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}}$. In practice, calculating $p_{\tilde{\mathbf{y}},\mathbf{s}}$ and $p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}$ individually is hard since the underlying distributions $p_{\tilde{\mathbf{y}},\mathbf{s}}$ and $p_{\tilde{\mathbf{y}}}$ are often unknown. Recall that our goal is to estimate the log of the ratio between these two joint distributions. Therefore, we estimate it through *density ratio estimation*, where the numerator $p_{\tilde{\mathbf{y}},\mathbf{s}}$ denotes the original joint distribution of the learning outcome $\tilde{\mathbf{y}}$ and ground-truth vectorized sensitive feature $\mathbf{s}$, and the denominator $p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}$ denotes the joint distribution of the learning outcome $\tilde{\mathbf{y}}$ and the vectorized sensitive feature $\tilde{\mathbf{s}}$ generated from the learning outcome using the aforementioned decoder.

We further reduce this density ratio estimation problem to a class probability estimation problem, which was originally developed in [216] for solving a different problem (i.e., the classification problem with the input distribution and the test distribution differing arbitrarily). The core idea is that, given a pair of learning outcome and vectorized sensitive feature, we want to predict whether it is drawn from the original joint distribution or from the joint distribution inferred by the decoder. We label each pair of learning outcome and ground-truth vectorized sensitive feature $(\tilde{\mathbf{y}}, \mathbf{s})$ with a positive label ($c = 1$) and each pair of learning outcome and generated vectorized sensitive feature $(\tilde{\mathbf{y}}, \tilde{\mathbf{s}})$ with a negative label ($c = -1$). After that, we

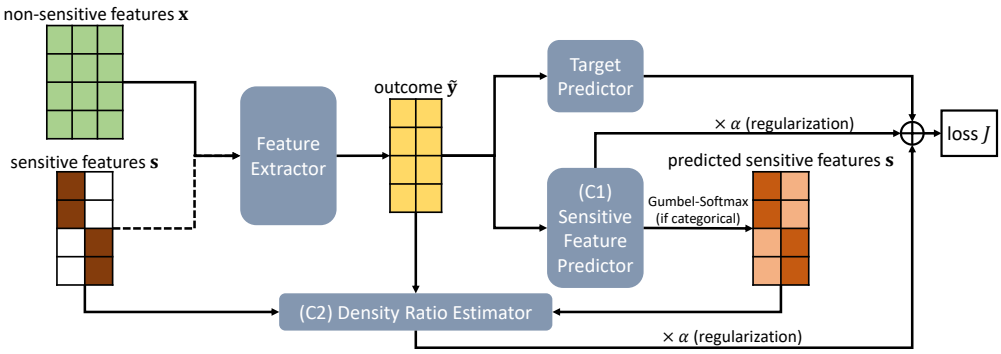


Figure 4.6: A General overview of the INFOFAIR framework. The dashed line between sensitive feature $\mathbf{s}$ and feature extractor means that sensitive features can be optionally passed into feature extractor as the input.

re-write the probability densities as $p_{\tilde{\mathbf{y}},\mathbf{s}} = \Pr[c=1|\tilde{\mathbf{y}},\mathbf{s}]$ and $p_{\tilde{\mathbf{y}}}q_{\mathbf{s}|\tilde{\mathbf{y}}} = \Pr[c=-1|\tilde{\mathbf{y}},\mathbf{s}]$. Then the density ratio can be further re-written as

$$\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}}q_{\mathbf{s}|\tilde{\mathbf{y}}}} = \log \frac{\Pr[c=1|\tilde{\mathbf{y}},\mathbf{s}]}{\Pr[c=-1|\tilde{\mathbf{y}},\mathbf{s}]} = \text{logit}\left(\Pr[c=1|\tilde{\mathbf{y}},\mathbf{s}]\right) \tag{4.58}$$

Furthermore, if we model $\Pr[c=1|\tilde{\mathbf{y}},\mathbf{s}]$ using logistic regression (i.e., $\Pr[c=1|\tilde{\mathbf{y}},\mathbf{s}] = \text{logistic}(\tilde{\mathbf{y}};\mathbf{s})$), Equation (4.58) is reduced to a simple linear function as

$$\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}}}{p_{\tilde{\mathbf{y}}}q_{\mathbf{s}|\tilde{\mathbf{y}}}} = \text{logit}\left(\text{logistic}(\tilde{\mathbf{y}};\mathbf{s})\right) = \mathbf{w}_1^T\tilde{\mathbf{y}} + \mathbf{w}_2^T\mathbf{s} \tag{4.59}$$

where both $\mathbf{w}_1$ and $\mathbf{w}_2$ are learnable parameters. Putting everything together, we re-write Equation (4.54) as

$$J = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[l(\mathbf{x};\mathbf{s};y;\tilde{\mathbf{y}};\theta) + \alpha \log q_{\mathbf{s}|\tilde{\mathbf{y}}}\right] + \alpha\mathbb{E}_{\{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}}\}\cup\{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}}}q_{\mathbf{s}|\tilde{\mathbf{y}}}\}}\left[\mathbf{w}_1^T\tilde{\mathbf{y}} + \mathbf{w}_2^T\mathbf{s}\right] \tag{4.60}$$

where $p_{\tilde{\mathbf{y}},\mathbf{s}}$ is the joint distribution of the learning outcome $\tilde{\mathbf{y}}$ and ground-truth vectorized sensitive feature $\mathbf{s}$, and $p_{\tilde{\mathbf{y}}}q_{\mathbf{s}|\tilde{\mathbf{y}}}$ is the joint distribution of the learning outcome $\tilde{\mathbf{y}}$ and predicted vectorized sensitive feature $\mathbf{s}$.

INFOFAIR: Overall framework. Based on the objective function (Equation (4.60)), we develop a generic end-to-end framework named INFOFAIR to solve the information-theoretic intersectional fairness problem. A general overview of the model architecture is shown in Figure 4.6. INFOFAIR contains four main modules, including (1) feature extractor, (2) target predictor, (3) sensitive feature predictor, and (4) a density ratio estimator. In principle, as long as each module is differentiable, INFOFAIR can be optimized by any gradient-based optimizer.

The general workflow of InfoFair is as follows.

1. The non-sensitive features and sensitive features (optional) are passed into a feature extractor to extract the learning outcomes.

2. The learning outcomes will be fed into a target predictor to predict the targets for a certain downstream task (i.e., $l\left(\mathbf{x};\mathbf{s};y;\tilde{\mathbf{y}};\theta\right)$ in Equation (4.60)).

3. The learning outcomes will be passed into the sensitive feature predictor to 'reconstruct' the vectorized sensitive features (i.e., $\log q_{\mathbf{s}|\tilde{\mathbf{y}}}$ in Equation (4.60)).

4. Together with the learning outcomes and the ground-truth vectorized sensitive features, the predicted vectorized sensitive features will be used to estimate the density ratio between the original distribution and the variational distribution (i.e., $\mathbf{w}_1^T\tilde{\mathbf{y}}+\mathbf{w}_2^T\mathbf{s}$ in Equation (4.60)).

Given a data point with categorical sensitive attribute(s), the predicted vectorized sensitive feature $\mathbf{s}$ is usually denoted as a one-hot vector. However, learning a one-hot vector is a difficult problem due to the discrete nature of vector elements, which makes the computation non-differentiable. To address this issue, we approximate such one-hot encoding by Gumbel-Softmax [217], which can be calculated as $\mathbf{s}\left[i\right]=\frac{\exp([\log(\mathbf{o}_{\mathbf{s}}[i])+g_i]/\tau)}{\sum_{j=1}^{n_{\mathbf{s}}}\exp([\log(\mathbf{o}_{\mathbf{s}}[j])+g_j]/\tau)}$, where $\mathbf{o}_{\mathbf{s}}$ is the output of the sensitive feature predictor, $n_{\mathbf{s}}$ is the dimension of $\mathbf{s}$, $g_1,\ldots,g_{n_{\mathbf{s}}}$ are i.i.d. points drawn from Gumbel$(0,1)$ distribution, and τ is the softmax temperature. As $\tau\to\infty$, the Gumbel-Softmax samples are uniformly distributed; while as $\tau\to 0$, the Gumbel-Softmax distribution converges to a one-hot categorical distribution. In InfoFair, we start with a high temperature and then anneal it during epochs of training.

InfoFair: Generalizations and variants. InfoFair is able to be generalized in multiple aspects. Here, we give some brief descriptions, each of which could be a future direction in applying our framework.

A – InfoFair *with equal opportunity.* The InfoFair framework is generalizable to enforce equal opportunity [186], another widely-used group fairness notion. We leave for future work to explore the potential of InfoFair in enforcing equal opportunity.

Equal opportunity ensures equality across demographic groups for a preferred label (i.e., the label that benefits an individual). Mathematically, it is defined as follows.

Definition 4.6. (Equal opportunity [186]). Following the settings of Definition 4.5, if equal opportunity is enforced, the hypothesis $h:\mathcal{X}\to\{0,1\}$ satisfies

$$\Pr\left[h\left(x\right)=1|x\in\mathcal{M},y=1\right]=\Pr\left[h\left(x\right)=1|x\in\mathcal{M}^{\mathcal{C}},y=1\right] \tag{4.61}$$

where $\Pr[\cdot]$ denotes the probability of an event happening.

Analogous to the relationship between mutual information and statistical parity, ensuring equal opportunity is essentially a conditional mutual information minimization problem.

$$\underbrace{p_{\tilde{\mathbf{y}}|\mathbf{s},y=1} = p_{\tilde{\mathbf{y}}|y=1}}_{\text{equal opportunity}} \Leftrightarrow \quad \underbrace{I(\tilde{\mathbf{y}};\mathbf{s}|y=1) = 0}_{\text{zero conditional mutual information}} \tag{4.62}$$

By the definition of conditional mutual information, we have $I(\tilde{\mathbf{y}};\mathbf{s}|y=1) = H(\mathbf{s}|y=1) - H(\mathbf{s}|\tilde{\mathbf{y}},y=1)$. For $H(\mathbf{s}|y=1)$, we assume it as a constant term by the similar rationale of statistical parity. Similarly, we can re-write $H(\mathbf{s}|\tilde{\mathbf{y}},y=1)$ as

$$H(\mathbf{s}|\tilde{\mathbf{y}},y=1) = \mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}|y=1}}\left[-\log q_{\mathbf{s}|\tilde{\mathbf{y}},y=1}\right] - \mathbb{E}_{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}|y=1}}\left[\log \frac{p_{\tilde{\mathbf{y}},\mathbf{s}|y=1}}{p_{\tilde{\mathbf{y}}|y=1} q_{\mathbf{s}|\tilde{\mathbf{y}},y=1}}\right] \tag{4.63}$$

Then, to compute $\log q_{\mathbf{s}|\tilde{\mathbf{y}},y=1}$, we could adopt similar strategy as computing $\log q_{\mathbf{s}|\tilde{\mathbf{y}}}$ described before by constructing a decoder $f(\tilde{\mathbf{y}};\mathbf{s};\mathbf{W})$ to 'reconstruct' $\mathbf{s}$ for *positive training samples*. Similarly, estimating the density ratio can be achieved by applying Equation (4.59) on *positive training samples*. Thus, InfoFair is able to enforce equal opportunity by minimizing

$$J = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[l(\mathbf{x};\mathbf{s};y;\tilde{\mathbf{y}};\theta) + \alpha \log q_{\mathbf{s}|\tilde{\mathbf{y}}}\right] + \alpha \mathbb{E}_{\left\{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}},\mathbf{s}|y=1}\right\}\cup\left\{(\tilde{\mathbf{y}},\mathbf{s})\sim p_{\tilde{\mathbf{y}}|y=1} q_{\mathbf{s}|\tilde{\mathbf{y}},y=1}\right\}}\left[\mathbf{w}_1^T\tilde{\mathbf{y}} + \mathbf{w}_2^T\mathbf{s}\right] \tag{4.64}$$

B – Relationship to adversarial debiasing. Adversarial debiasing framework [211] consists of (1) a predictor that predicts the class membership probabilities using given data and (2) an adversary that takes the output of the predictor to predict the sensitive attribute of given data. The framework is optimized to minimize the loss function of the predictor while maximizing the loss function of the adversary. If we merge feature extractor and target predictor to one single module and remove the density ratio estimator, InfoFair will degenerate to the adversarial debiasing method.

C – Relationship to information bottleneck. If we set the loss function l in Equation (4.54) as the negative mutual information $-I(\tilde{\mathbf{y}};y)$, Equation (4.54) becomes the information bottleneck method [218]. Then the goal becomes to learning $\tilde{\mathbf{y}}$ that depends on the vectorized sensitive attribute $\mathbf{s}$ minimally and ground truth y maximally.

D – Fairness for continuous-valued sensitive features. Most existing works in fair machine learning only consider categorical sensitive attribute (e.g., gender, race). The InfoFair framework could be generalized to continuous-valued features, as mutual information supports continuous-valued random variables. This advantage could empower our framework to work in even more application scenarios. For example, in image classification, we can classify images

without the impact of certain image patches (e.g., patches that relate to individual's skin color). However, a major difficulty lies in modeling the variational distribution of sensitive attribute given the learning outcomes extracted from feature extractor. A potential solution could be utilizing a generative model (e.g., VAEs [219]) as the sensitive feature predictor.
E – Fairness for non-IID graph data. For fair graph mining, given a graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ where $\mathbf{A}$ is the adjacency matrix and $\mathbf{X}$ is the node feature matrix, we can use graph convolutional layer(s) as a feature extractor with the weight of the last layer to be identity matrix $\mathbf{I}$ and no nonlinear activation in the last graph convolution layer, in order to extract node representations. The reason for such a specific architecture in the last graph convolution layer is as follows. In general, a graph convolutional layer consists of two operations: feature aggregation $\mathbf{Z} = f_{\text{aggregate}}(\mathbf{A}; \mathbf{X}) = \mathbf{AX}$ and feature transformation $\mathbf{H} = f_{\text{transform}}(\mathbf{Z}; \mathbf{W}) = \sigma(\mathbf{ZW})$, where $\mathbf{W}$ is learnable parameters, and σ is usually a nonlinear activation. The last layer in GCN [7] is simply $\text{softmax}(\mathbf{AXW})$, which can be viewed as a general multi-class logistic regression on the aggregated feature $\mathbf{Z} = \mathbf{AX}$ (i.e., $\text{softmax}(\mathbf{ZW})$).
F – Fairness beyond classification. Note that InfoFair does not have specific restrictions on the architecture of the feature extractor, target predictor, or sensitive target predictor, which empowers it to handle many different types of tasks by selecting the proper architecture for each module. For example, if an analyst aims to learn fair representations with respect to gender for recommendation, s/he can set the feature extractor to be a multi-layer perceptron (MLP) for learning outcome extraction, the target predictor layer to be an MLP that predicts a rating and minimizes the mean squared error (MSE) between the predicted rating and ground-truth rating, and the sensitive target predictor to be another MLP with softmax to predict the gender based on extracted embedding.

4.3.3 Experimental Evaluation

In this part, we conduct experimental evaluations. All experiments are designed to answer the following questions:

- *RQ1.* How does the fairness impact the learning performance?
- *RQ2.* How effective is InfoFair in mitigating bias?

Experimental settings. The detailed experimental settings to evaluate InfoFair, including dataset descriptions, baseline methods, evaluation metrics, parameter settings, and all other necessary information for reproducibility, are presented as follows.
A – Hardware and software specifications. All three datasets are publicly available online. All models (i.e., InfoFair and baseline methods) are implemented with PyTorch 1.9.0 and are

Table 4.15: Statistics of datasets to evaluate InfoFair.

Dataset	# Samples	# Attributes	# Classes
COMPAS	6,172	52	2
Adult Income	45,222	14	2
Dutch Census	60,420	11	2

trained on a Linux server with 2 Intel Xeon Gold 6240R CPUs at 2.40 GHz and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory. The source code of InfoFair can be downloaded at `https://github.com/jiank2/infofair`.

B – Dataset descriptions. We test InfoFair on three commonly-used datasets in fair machine learning research. The statistics of these datasets are summarized in Table 4.15.

For all datasets, we randomly split them into 80% training set, 10% validation set, and 10% test set. A description of each dataset, as well as the pre-processing procedures, is shown below.

- *COMPAS* dataset contains in total of 6,172 criminal defendants in Broward County, Florida. Each defendant is described by 52 attributes used by the COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) algorithm for scoring his/her likelihood of re-offending crimes in the following 2 years. The goal is to determine whether a criminal defendant will re-offend in the next 2 years. To pre-process the dataset, we remove duplicate features and features related to date, case number, and descriptions of criminal charge. We then manually calculate the length of stay in jail for each criminal and quantize them into three bins: length less than 1 week, length larger than 1 week but less than 3 months, and length larger than 3 months. Similarly, for features related to the count of criminal charges, we quantize them into three bins: count is 0, count within 1 to 3, and count more than 3. The rest of the pre-processing procedures are as follows: (1) continuous-valued features are kept as is; and (2) discreate features are transformed into a single boolean value or one-hot encoding for binary and other features.

- *Adult Income* dataset contains in total of 45,222 individuals. Each individual is described by 14 attributes that relate to his/her personal demographic information, including gender, race, education, marital status, etc. The goal is to predict whether a person can earn a salary over $50,000 a year. To pre-process it, all features in the original datasets are used. We first remove data samples with missing values. Then we follow similar procedures as processing *COMPAS* dataset.

- *Dutch Census* dataset contains in total of 60,420 individuals.[35] Each individual is described by 11 attributes that relate to her/his demographic and economic information to predict whether s/he has a prestigious occupation. The pre-processing procedures are the same as the *Adult Income* dataset.

C – Baseline methods. We compare InfoFair with the following baseline methods.

- *Learning Fair Representations (LFR)* [210] learns a set of fair prototype representations. Each data sample is first mapped to a prototype, which is used to predict a fair outcome. We use the implementation by IBM AIF360[36] and the same grid search strategy for hyperparameters in [210].

- *MinDiff* [220] ensures equal false positive rate by minimizing the maximum mean discrepancy (MMD) between the two demographic groups with negative samples only. We implement our own version of MinDiff with the Gaussian kernel. The hyperparameters for the Gaussian kernel is set to be consistent with [220]. For fair comparison, we set the regularization hyperparameter to 0.1, which is the same with the corresponding setting for InfoFair.

- *Disparate Impact (DI)* [40] ensures disparate impact by interpolating the original data distribution with an unbiased distribution. For fair comparison, we set the linear interpolation coefficient, which is referred to as λ in [40], such that the interpolation ratios of [40] and ours are the same, i.e., $\frac{1-\lambda}{\lambda} = \frac{1}{\alpha}$.

- *Adversarial Debiasing (Adversarial)* [211] uses an adversary to predict the sensitive attribute using the prediction from a predictor. Both the predictor and the adversary can be flexibly chosen. Since its official source code is not available, we implement the model using the same machine configurations as InfoFair. For fair comparison, we switch (1) the predictor to feature extractor and target predictor in our framework and (2) the adversary to sensitive feature predictor in our framework. We also set the same learning rate as our framework.

- *Fair Classification with Fairness Constraints (FCFC)* [206] measures the statistical imparity as the covariance between the sensitive attribute of a data sample and the distance of the corresponding data sample to the decision boundary of a linear classifier. We use the official implementation of FCFC provided by Zafar et al. and adopt their released parameter settings in our experiments.

[35] `https://sites.google.com/site/faisalkamiran/`
[36] `https://github.com/Trusted-AI/AIF360`

- *GerryFair* [208] ensures subgroup fairness for cost-sensitive classification through fictitious play from the game-theoretic perspective. Since the relationship between α in InfoFair and parameters of *GerryFair* is unclear, we use the default parameters provided in the officially released source code.

- *Generalized Demographic Parity (GDP)* [221] computes the weighted total variation distance on the local average prediction and the global average prediction. For fair comparison, we use the official implementation, set the same backbone model for feature extraction and prediction, and use the same regularization hyperparameter (0.1) as InfoFair.

D – Evaluation metrics. To answer *RQ1*, we measure the performance of classification using micro F1 and macro F1 scores (Micro/Macro F1). To answer *RQ2*, we measure to what extent the bias is reduced by the average statistical imparity (Imparity) and the relative bias reduction (Reduction) on average statistical imparity. The average statistical imparity (Imparity) is defined as Imparity = $\text{avg}\left(\left|\Pr\left[\hat{y}=c|\mathbf{x}\in g_1\right]-\Pr\left[\hat{y}=c|\mathbf{x}\in g_2\right]\right|\right)$ for any class label c and any pair of two different demographic groups g_1 and g_2. The relative bias reduction measures the relative decrease of the imparity of the debiased outcomes $\text{Imparity}_{\text{debiased}}$ to the imparity of vanilla outcomes (i.e., outcomes without fairness consideration) $\text{Imparity}_{\text{vanilla}}$. It is computed mathematically as Reduction $= 1-\frac{\text{Imparity}_{\text{debiased}}}{\text{Imparity}_{\text{vanilla}}}$. Note that the relative bias reduction defined above can be negative if the debiased learning outcome is more biased than the vanilla learning outcome.

E – Experimental protocol and model architecture. The learning task we consider is fair classification with respect to categorical sensitive attribute(s). For all datasets, we take both non-sensitive features and sensitive features as input to the feature extractor. Regarding the model architecture, for *Adult Income* and *Dutch Census*, the feature extractor is a 1-layer MLP with hidden dimension 32, the target predictor contains one hidden layer that calculates the log likelihood of predicting class label using the embeddings output by the feature extractor, and the sensitive feature predictor is similar to the target predictor that leverages one hidden layer to calculate the log likelihood of predicting the vectorized sensitive feature using the extracted embeddings. For *COMPAS*, we set the feature extractor to be a 2-layer MLP with hidden dimension 32 in each layer, while keeping all other modules to be the same as they are for *Adult Income* and *Dutch Census*.

F – Detailed parameter setting. For all datasets, we set the regularization parameter $\alpha = 0.1$. The number of epochs for training is set to 100 with a patience of 5 for early stopping. Weight decay is set to 0.01. We tune the learning rate as 0.001 for *DI* and 0.0001 for *MinDiff*, *Adversarial*, and our method. All learnable model parameters are optimized with the Adam

Table 4.16: Debiasing results on all datasets. Lower is better for the gray column (Imparity). Higher is better for all others.

Debiasing results on the *COMPAS* dataset									
Method	**gender**			**race**			**gender & race**		
	Micro/Macro F1	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**
Vanilla	0.972/0.972	0.050	0.000%	0.972/0.972	0.181	0.000%	0.972/0.972	0.234	0.000%
LFR	0.554/0.357	0.000	100.0%	N/A	N/A	N/A	N/A	N/A	N/A
MinDiff	0.972/0.972	0.050	0.000%	N/A	N/A	N/A	N/A	N/A	N/A
DI	0.972/0.972	0.050	0.000%	0.972/0.972	0.181	0.000%	0.972/0.972	0.234	0.000%
Adversarial	0.554/0.357	0.000	100.0%	0.554/0.357	0.000	100.0%	0.554/0.357	0.000	100.0%
FCFC	0.446/0.308	0.000	100.0%	0.446/0.308	0.000	100.0%	0.446/0.308	0.000	100.0%
GerryFair	0.972/0.972	0.050	0.000%	0.972/0.972	0.181	0.000%	0.972/0.972	0.234	0.000%
GDP	0.972/0.972	0.050	0.000%	0.972/0.972	0.181	0.000%	0.972/0.972	0.234	0.000%
InfoFair	0.924/0.923	0.038	23.15%	0.815/0.803	0.179	1.010%	0.877/0.872	0.231	1.350%
Debiasing results on the *Adult Income* dataset									
Method	**gender**			**race**			**gender & race**		
	Micro/Macro F1	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**
Vanilla	0.830/0.762	0.066	0.000%	0.830/0.762	0.062	0.000%	0.830/0.762	0.083	0.000%
LFR	0.743/0.426	0.000	100.0%	N/A	N/A	N/A	N/A	N/A	N/A
MinDiff	0.828/0.746	0.058	12.06%	N/A	N/A	N/A	N/A	N/A	N/A
DI	0.823/0.730	0.053	19.85%	0.825/0.743	0.056	10.62%	0.823/0.736	0.081	2.276%
Adversarial	0.743/0.426	0.000	100.0%	0.743/0.426	0.000	100.0%	0.743/0.426	0.000	100.0%
FCFC	0.257/0.204	0.000	100.0%	0.257/0.204	0.000	100.0%	0.257/0.204	0.000	100.0%
GerryFair	0.833/0.752	0.056	15.70%	0.833/0.752	0.067	−7.664%	0.797/0.710	0.215	−158.3%
GDP	0.825/0.744	0.055	16.73%	0.827/0.749	0.059	6.351%	0.824/0.740	0.075	9.246%
InfoFair	0.816/0.721	0.047	29.24%	0.810/0.686	0.042	32.11%	0.818/0.714	0.082	1.532%
Debiasing results on the *Dutch Census* dataset									
Method	**gender**			**marital status**			**gender & marital status**		
	Micro/Macro F1	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**	**Micro/Macro F1**	**Imparity**	**Reduction**
Vanilla	0.832/0.831	0.119	0.000%	0.832/0.831	0.079	0.000%	0.832/0.831	0.172	0.000%
LFR	0.521/0.342	0.000	100.0%	N/A	N/A	N/A	N/A	N/A	N/A
MinDiff	0.831/0.830	0.107	10.16%	N/A	N/A	N/A	N/A	N/A	N/A
DI	0.825/0.824	0.104	12.43%	0.830/0.830	0.080	−1.156%	0.814/0.811	0.127	26.65%
Adversarial	0.521/0.342	0.000	100.0%	0.521/0.342	0.000	100.0%	0.521/0.342	0.000	100.0%
FCFC	0.479/0.324	0.000	100.0%	0.479/0.324	0.000	100.0%	0.479/0.324	0.000	100.0%
GerryFair	0.826/0.823	0.078	34.29%	0.826/0.823	0.070	11.70%	0.826/0.823	0.125	27.53%
GDP	0.828/0.826	0.097	18.31%	0.827/0.826	0.086	−9.056%	0.827/0.825	0.131	23.80%
InfoFair	0.817/0.813	0.068	43.08%	0.815/0.811	0.077	2.017%	0.819/0.817	0.128	25.65%

optimizer [182]. The starting temperature for Gumbel-Softmax is set to 1 and is divided by 2 every 50 epochs for annealing. To reduce randomness and enhance reproducibility, we run 5 different initializations with random seed from 0 to 4.

Main results. We test InfoFair, as well as baseline methods, in three different settings: debiasing binary sensitive attribute (i.e., gender for all three datasets), debiasing non-binary sensitive attribute (i.e., race for *COMPAS* and *Adult Income*, marital status for *Dutch Census*), and debiasing multiple sensitive attributes (i.e., gender & race for *COMPAS* and *Adult Income*, gender & marital status for *Dutch Census*). For each dataset and each setting, we train each model on the training set, then select the trained model with best bias mitigation performance on the validation set, and report its performance on the test set. For the vanilla model (without any fairness consideration), we report the model with the highest micro and macro F1 scores. This is because the algorithm administrators are often more concerned with maximizing the utility of classification algorithms. The results of *LFR* and *MinDiff* in debiasing non-binary sensitive attribute and multiple sensitive attributes are absent since they only handle binary sensitive attribute by design.

A – Effectiveness results. The effectiveness results of InfoFair and all baseline methods on *COMPAS*, *Adult Income*, and *Dutch Census* datasets are shown in Table 4.16. From the table, we have the following observations. First, our method is the only method that can mitigate bias (i.e., Imparity and Reduction) effectively and consistently with a small degree of sacrifice to the vanilla classification performance (i.e., Micro/Macro F1) for all datasets and all settings. Second, though *LFR*, *Adversarial Debiasing*, and *FCFC* achieve the perfect bias reduction, their classification performance is severely reduced by predicting all data samples with the same label (i.e., negative label for *LFR* and *Adversarial Debiasing*, positive label for *FCFC*). Though, in a few settings, *DI*, *GerryFair*, and *GDP* mitigate more bias than InfoFair, they either *amplify* the bias or fail to outperform InfoFair in the other settings. All in all, InfoFair achieves the best balance in reducing the bias and maintaining the classification accuracy in most cases.

B – Trade-off between micro F1 score and average statistical imparity. Figure 4.7 shows the results of the trade-off between the micro F1 score (Micro F1) and the average statistical imparity (Imparity). From the figure, we can observe that InfoFair achieves the best trade-off between accuracy and fairness (i.e., being closer to the bottom right corner in Figure 4.7) in most cases.

Ablation study. Let $T = \mathbb{E}\left[l\left(\mathbf{x}; \mathbf{s}; y; \tilde{\mathbf{y}}; \theta\right)\right]$ be the empirical loss of target predictor, $S = \alpha\mathbb{E}\left[\log q_{\mathbf{s}|\tilde{\mathbf{y}}}\right]$ be the empirical loss of sensitive feature predictor, and $D = \alpha\mathbb{E}\left[\mathbf{w}_1^T\tilde{\mathbf{y}} + \mathbf{w}_2^T\mathbf{s}\right]$ be the empirical loss of density ratio estimator, the objective function of InfoFair (Equation (4.60)) can be written as $J = T + S + D$. To evaluate the effectiveness on optimizing the variational representation of mutual information, we compare with two variants of the objective function, i.e., $T + S$ and $T + D$, on the same datasets and the same set of sensitive attributes described before. Experimental protocols and parameter settings are kept the same among all compared objective functions (i.e., $T + S + D$, $T + S$, and $T + D$). The results of the ablation study are shown in Figure 4.8. From the figure, we observe that our objective function (i.e., $T + S + D$) can mitigate more bias than the other two variants (i.e., $T + S$ and $T + D$) in most cases. This implies that our variational representation can better model the dependence between the learning outcomes and the vectorized sensitive features.

When debiasing race on *COMPAS* and debiasing marital status on *Dutch Census*, we observe that the cardinalities of the demographic groups are more imbalanced. Recall that the goal of sensitive feature predictor is to reduce the accuracy of predicting sensitive feature using the extracted embedding. When the demographic groups are more imbalanced, it tends to learn an embedding that contains information about a wrong demographic group to reduce its accuracy. Thus, though $T + S$ may achieve lower imparity, the statistical dependence between the extracted embeddings and the sensitive feature may not be reduced, meaning

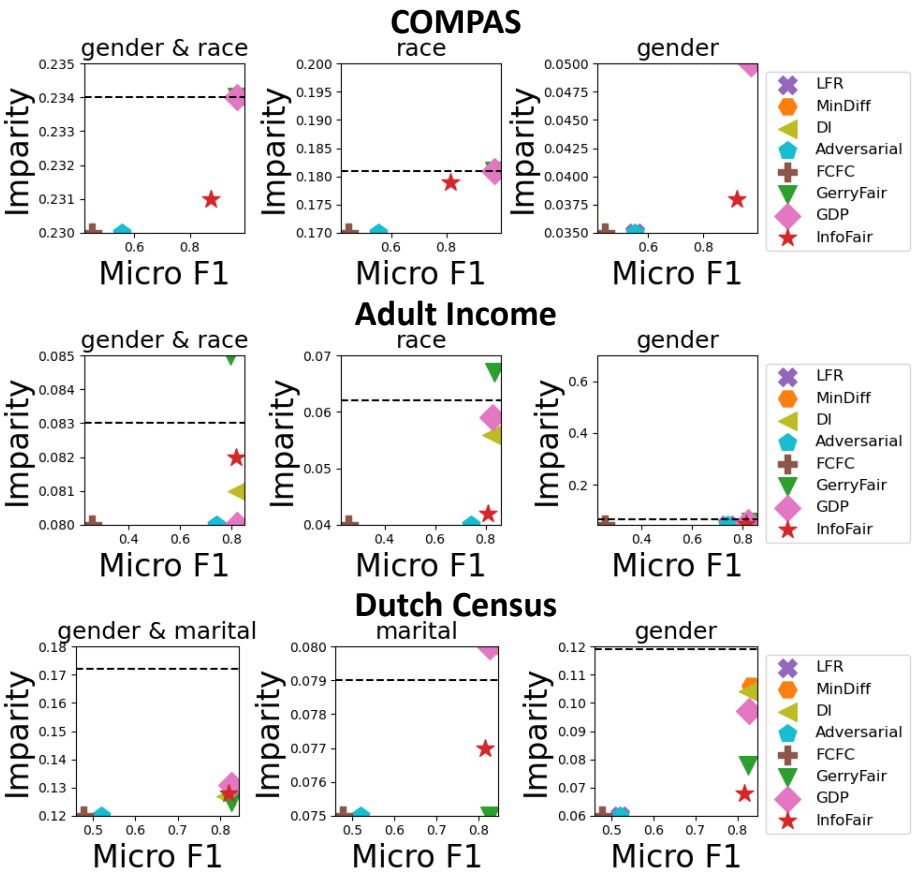


Figure 4.7: Trade-off between the micro F1 score and the average statistical imparity. Best viewed in color. Red star represents InfoFair. The closer to bottom right, the better trade-off between the micro F1 score and the average statistical imparity. Bias is amplified by a method if its corresponding point is above the dashed line (which denotes the imparity of Vanilla).

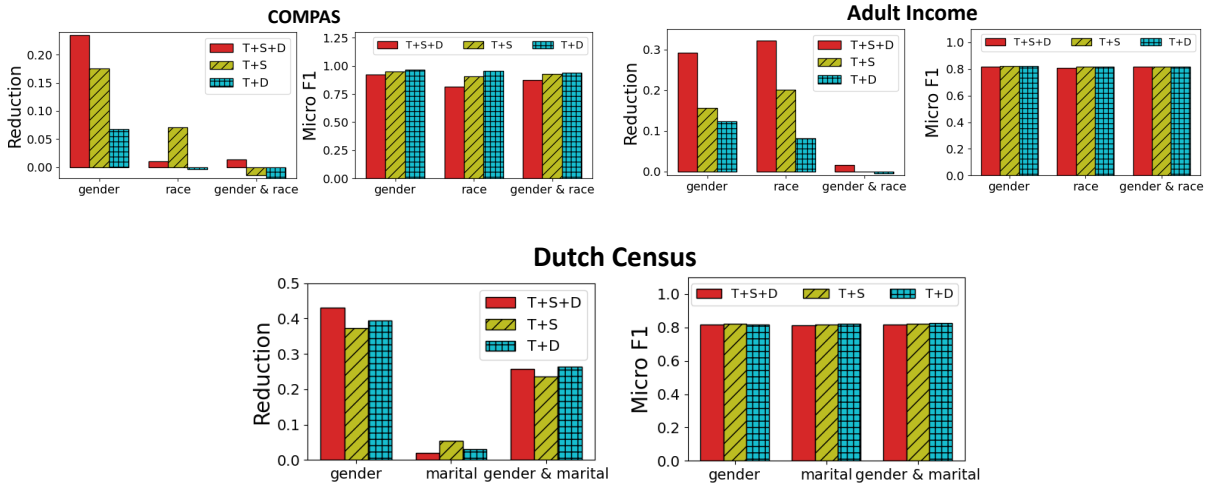


Figure 4.8: Ablation study on the variants of objective function. Best viewed in color. Higher is better.

that the extracted embeddings are merely shifted to correlate with wrong demographic groups. However, with the addition of density ratio estimator, we ensure that (1) not only the sensitive feature predictor makes wrong prediction (2) but also the distribution of sensitive attribute and the extracted embeddings modeled by S ($p_{\tilde{\mathbf{y}}} q_{\mathbf{s}|\tilde{\mathbf{y}}}$) are similar to its corresponding original distribution (i.e., $p_{\tilde{\mathbf{y}},\mathbf{s}}$).

CHAPTER 5: SAFEGUARDING FAIR GRAPH MINING

Address the safeguarding task is a postlude in building an algorithmic foundation of fair graph mining. It brings together all key properties to make the developed debiasing techniques deployable in the real-world setting. Notably, the real world often consists of adversarial activities, and it is unclear how these adversarial activities would affect the model fairness. To this end, in this chapter, we address the safeguarding challenge by studying how to attack the fairness of a graph mining model, such that the model after adversarial attacks would be more biased but still expresses strong performance in the downstream tasks. We further empirically reveal the properties of the adversarial edges in the graph that contribute the most to effective and efficient fairness attacks.

5.1 INTRODUCTION

Algorithmic fairness in graph learning has received much research attention [17, 18, 47]. Despite its substantial progress, existing studies mostly assume the benevolence of input graphs and aim to ensure that the bias would not be perpetuated or amplified in the learning process. However, malicious activities in the real world are commonplace. For example, consider a financial fraud detection system which utilizes a transaction network to classify whether a bank account is fraudulent or not [153, 198]. An adversary may manipulate the transaction network (e.g., malicious banker with access to the transaction data, theft of bank accounts to make malicious transactions), so that the graph-based fraud detection model would exhibit unfair classification results with respect to people of different demographic groups. Consequently, a biased fraud detection model may infringe civil liberty to certain financial activities and impact the well-being of an individual negatively [222]. It would also make the graph learning model fail to provide the same quality of service to people of certain demographic groups, causing the financial institutions to lose business in the communities of the corresponding demographic groups. Thus, it is critical to understand how resilient a graph learning model is with respect to adversarial attacks on fairness, which we term as *fairness attacks*.

To date, fairness attacks have not been well studied. Sporadic literature often follows two strategies: (1) adversarial data point injection, which is often designed for tabular data rather than graphs [223, 224, 225, 226], or (2) adversarial edge injection, which only attacks the group fairness of a graph neural network [85]. It is thus crucial to study how to attack different fairness definitions for a variety of graph learning models with strategies other than

data/edge injection.

To achieve this goal, we study the Fairness attacks on graph learning (FATE) problem. We formulate it as a bi-level optimization, where the lower-level problem optimizes a task-specific loss function to make the fairness attacks deceptive, and the upper-level problem leverages the supervision signal to modify the input graph and maximize the bias function corresponding to a user-defined fairness definition. To solve the bi-level optimization problem, we design a meta learning-based solver (FATE), whose key idea is to compute the meta-gradient of the upper-level bias function with respect to the input graph to guide the fairness attacks. Compared with existing works, FATE has two major advantages. First, it is capable of attacking *any* fairness definition on *any* graph learning model, as long as the corresponding bias function and the task-specific loss function are differentiable. Second, it is equipped with the ability for either continuous or discretized poisoning attacks on the graph topology. We also briefly discuss its ability for poisoning attacks on node features later.

The major contributions of this work are summarized as follows.

- *Problem definition.* We formally define the problem of fairness attacks on graph learning (the FATE problem). Based on the definition, we formulate it as a bi-level optimization problem, whose key idea is to maximize a bias function in the upper level while minimizing a task-specific loss function for a graph learning task.

- *Attacking framework.* We design an end-to-end attacking framework named FATE. It learns a perturbed graph topology via meta learning, such that the bias with respect to the learning results trained with the perturbed graph will be amplified.

- *Empirical evaluation.* We conduct experiments on three benchmark datasets to demonstrate the efficacy of the FATE framework in amplifying the bias while being the most deceptive method (i.e., achieving the highest micro F1 score) on semi-supervised node classification. We also conduct empirical analysis to understand the types of edges contributing the most to fairness attacks.

5.2 PRELIMINARIES AND PROBLEM DEFINITION

Notations. Throughout this work, we use bold upper-case letter for matrix (e.g., $\mathbf{A}$), bold lower-case letter for vector (e.g., $\mathbf{x}$), and calligraphic letter for set (e.g., $\mathcal{G}$). We use superscript ^{T} to denote the transpose of a matrix/vector (e.g., $\mathbf{x}^T$ is the transpose of $\mathbf{x}$). Regarding matrix/vector indexing, we use conventions similar to Numpy in Python. For

example, $\mathbf{A}[i,j]$ is the entry of $\mathbf{A}$ at the i-th row and j-th column; $\mathbf{x}[i]$ is the i-th entry of $\mathbf{x}$; and $\mathbf{A}[i,:]$ and $\mathbf{A}[j,:]$ are the i-th row and j-th column of $\mathbf{A}$, respectively.

Algorithmic fairness. The general principle of algorithmic fairness is to ensure the learning results would not favor one side over another.[37] Among several fairness definitions that follow this principle, group fairness [40, 186] and individual fairness [41] are the most widely studied ones. Group fairness splits the entire population into multiple demographic groups by a sensitive attribute (e.g., gender) and ensures the parity of a statistical property among the learning results of those groups. For example, statistical parity, a classic group fairness definition, guarantees the statistical independence between the learning results (e.g., predicted labels of a classification algorithm) and the sensitive attribute [40]. Individual fairness suggests that similar individuals should be treated similarly. It is often formulated as a Lipschitz inequality such that distance between the learning results of two data points should be no larger than the difference between these two data points [41].

Problem definition. Existing work [85] for fairness attacks on graphs randomly injects adversarial edges that satisfy certain property, so that the disparity between the learning results of two different demographic groups would be amplified. However, it suffers from three major limitations. (1) First, it only attacks statistical parity while overlooking other fairness definitions (e.g., individual fairness [41]). (2) Second, it only considers adversarial edge injection, excluding other manipulations like edge deletion or reweighting. Hence, it is essential to investigate the possibility to attack other fairness definitions on real-world graphs with an arbitrary choice of manipulation operations. (3) Third, it does not consider the utility of graph learning models while achieving the fairness attacks, resulting in performance degradation in the downstream tasks. However, an institution that applies the graph learning models are often utility-maximizing [227, 228]. Thus, a performance degradation in the utility might make the fairness attacks not deceptive from the perspective of a utility-maximizing institution.

In this work, we seek to overcome the aforementioned limitations. To be specific, given an input graph, an optimization-based graph learning model, and a user-defined fairness definition, we aim to learn a modified graph such that a bias function of the corresponding fairness definition would be maximized for *effective* fairness attacks, while minimizing the task-specific loss function with respect to the graph learning model for *deceptive* fairness attacks. Formally, we define the problem of fairness attacks on graph learning, which is referred to as the FATE problem.

Problem 5.1. FATE: Fairness attacks on graph learning.

[37] `https://www.merriam-webster.com/dictionary/fairness`

Given: (1) an undirected graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, (2) a task-specific loss function $l(\mathcal{G}, \mathcal{Y}, \Theta, \theta)$ where $\mathcal{Y}$ is the graph learning results, Θ is the set of learnable variables, and θ is the set of hyperparameters, (3) a bias function $b(\mathbf{Y}, \Theta^*, \mathbf{F}, \theta)$ where $\Theta^* = \arg\min_\Theta l(\mathcal{G}, \mathcal{Y}, \Theta, \theta)$, and $\mathbf{F}$ is the matrix that contains auxiliary fairness-related information (e.g., sensitive attribute values of all nodes in $\mathcal{G}$ for group fairness, pairwise node similarity matrix for individual fairness), and(4) an integer budget B.
Find: a poisoned graph $\widetilde{\mathcal{G}} = \left\{\widetilde{\mathbf{A}}, \widetilde{\mathbf{X}}\right\}$ which satisfies the following properties: (1) $d\left(\mathcal{G}, \widetilde{\mathcal{G}}\right) \leq B$ where $d\left(\mathcal{G}, \widetilde{\mathcal{G}}\right)$ is the distance between the input graph $\mathcal{G}$ and the poisoned graph $\widetilde{\mathcal{G}}$ (e.g., $\|\mathbf{A}, \widetilde{\mathbf{A}}\|_{1,1}$), (2) the bias function $b(\mathbf{Y}, \Theta^*, \mathbf{F})$ is maximized for effectiveness, and (3) the task-specific loss function $l\left(\widetilde{\mathcal{G}}, \mathcal{Y}, \Theta, \theta\right)$ is minimized for deceptiveness.

5.3 METHODOLOGY

In this part, we formulate Problem 5.1 as a bi-level optimization problem, followed by a generic meta learning-based solver named FATE.

5.3.1 Problem Formulation

Given an input graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ with adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, an attacker aims to learn a poisoned graph $\widetilde{\mathcal{G}} = \{\widetilde{\mathbf{A}}, \widetilde{\mathbf{X}}\}$ such that the graph learning model will be maximally biased when trained on $\widetilde{\mathcal{G}}$. In this work, we consider the following settings for the attacker.

- *The goal of the attacker.* The attacker aims to amplify the bias of the graph learning results output by a victim graph learning model. And the bias to be amplified is a choice made by the attacker based on which fairness definition the attacker aims to attack.
- *The knowledge of the attacker.* Following similar settings in [85], we assume the attacker has access to the adjacency matrix, the feature matrix of the input graph, and the sensitive attribute of all nodes in the graph. For a (semi-)supervised learning problem, we assume that the ground-truth labels of the training nodes are also available to the attacker. For example, for a graph-based financial fraud detection problem, the malicious banker may have access to the demographic information (i.e., sensitive attribute) of the account holders and also know whether some bank accounts are fraudulent or not, which are the ground-truth labels for training nodes. Similar to

[80, 81, 85], the attacker has no knowledge about the parameters of the victim model. Instead, the attacker will perform a gray-box attack by attacking a surrogate graph learning model.

- *The capability of the attacker.* The attacker is able to perturb up to B edges/features in the graph (i.e., $\|\mathbf{A} - \widetilde{\mathbf{A}}\|_{1,1} \leq B$ or $\|\mathbf{X} - \widetilde{\mathbf{X}}\|_{1,1} \leq B$).

Based on that, we formulate Problem 5.1 as a bi-level optimization problem as follows.

$$\widetilde{\mathcal{G}} = \underset{\mathcal{G}}{\operatorname{argmax}}\ b\left(\mathbf{Y}, \Theta^*, \mathbf{F}\right) \quad \text{s.t. } \Theta^* = \arg\min_{\Theta} l\left(\mathcal{G}, \mathbf{Y}, \Theta, \theta\right),\ d\left(\mathcal{G}, \widetilde{\mathcal{G}}\right) \leq B \tag{5.1}$$

where the lower-level problem learns an optimal surrogate graph learning model Θ^* by minimizing $l\left(\mathcal{G}, \mathbf{Y}, \Theta, \theta\right)$, the upper-level problem finds a poisoned graph $\widetilde{\mathcal{G}}$ that could maximize a bias function $b\left(\mathbf{Y}, \Theta^*, \mathbf{F}\right)$ for the victim graph learning model, and the distance $d\left(\mathcal{G}, \widetilde{\mathcal{G}}\right)$ between the input graph $\mathcal{G}$ and the poisoned graph $\widetilde{\mathcal{G}}$ is constrained to satisfy the setting about the budgeted attack. Note that Equation (5.1) is applicable to attack *any* fairness definition on *any* graph learning model, as long as the bias function $b\left(\mathbf{Y}, \Theta^*, \mathbf{F}\right)$ and the loss function $l\left(\mathcal{G}, \mathbf{Y}, \Theta, \theta\right)$ are differentiable.

Lower-level optimization problem. Many graph learning models are essentially solving optimization problems. Take the graph convolutional network (GCN) [7] as an example. It learns the node representation by aggregating information from its neighborhood, i.e., message passing. Mathematically, for an L-layer GCN, the hidden representation at k-th layer can be represented as $\mathbf{E}^{(k)} = \sigma\left(\widehat{\mathbf{A}}\mathbf{E}^{(k-1)}\mathbf{W}^{(k)}\right)$ where σ is a nonlinear activation function (e.g., ReLU), $\widehat{\mathbf{A}} = \mathbf{D}^{-1/2}\left(\mathbf{A} + \mathbf{I}\right)\mathbf{D}^{-1/2}$ with $\mathbf{D}$ being the degree matrix of $(\mathbf{A} + \mathbf{I})$, and $\mathbf{W}^{(k)}$ is the learnable weight matrix of the k-th layer. Then the lower-level optimization problem aims to learn the set of parameters $\Theta^* = \left\{\mathbf{W}^{(k)} | k = 1, \ldots, L\right\}$ that could minimize a task-specific loss function (e.g., cross-entropy loss for semi-supervised node classification).

Upper-level optimization problem. To attack the fairness aspect of a graph learning model, we aim to maximize a differentiable bias function $b\left(\mathbf{Y}, \Theta^*, \mathbf{F}\right)$ with respect to a user-defined fairness definition in the upper-level optimization problem. For example, for statistical parity [40], the fairness-related auxiliary information matrix $\mathbf{F}$ can be defined as the one-hot demographic membership matrix, where $\mathbf{F}\left[i, j\right] = 1$ if and only if node i belongs to the j-th demographic group. Then the statistical parity is equivalent to the statistical independence between the learning results $\mathbf{Y}$ and $\mathbf{F}$. Based on that, existing studies propose several differentiable measurements of the statistical dependence between $\mathbf{Y}$ and $\mathbf{F}$ as the bias function. For example, Bose and Hamilton [47] use mutual information $I\left(\mathbf{Y}; \mathbf{F}\right)$ as the

bias function; Prost et al. [220] define the bias function as the Maximum Mean Discrepancy $\mathrm{MMD}\left(\mathbf{Y}_0, \mathbf{Y}_1\right)$ between the learning results of two different demographic groups $\mathbf{Y}_0$ and $\mathbf{Y}_1$.

5.3.2 The FATE Framework

To solve Equation (5.1), we present a generic attacking framework named FATE to learn the poisoned graph. The key idea is to view Equation (5.1) as a meta learning problem, which treats the graph $\mathcal{G}$ as a hyperparameter and aims to find suitable hyperparameter settings for a learning task [229]. With that, we learn the poisoned graph $\widetilde{\mathcal{G}}$ using the meta-gradient of the bias function $b\left(\mathbf{Y}, \Theta^*, \mathbf{F}\right)$ with respect to $\mathcal{G}$. In the following, we introduce two key parts of FATE in details, including meta-gradient computation and graph poisoning with meta-gradient.

Meta-gradient computation. The key term in learning the poisoned graph is the meta-gradient of the bias function with respect to the graph $\mathcal{G}$. Before computing the meta-gradient, we assume that the lower-level optimization problem converges in T epochs. Thus, we first pre-train the lower-level optimization problem by T epochs to obtain the optimal model $\Theta^* = \Theta^{(T)}$ before computing the meta-gradient. The training of the lower-level optimization problem can also be viewed as a dynamic system with the following updating rule

$$\Theta^{(t+1)} = \mathrm{opt}^{(t+1)}\left(\mathcal{G}, \Theta^{(t)}, \theta, \mathbf{Y}\right), \ \forall t \in \{1, \ldots, T\} \tag{5.2}$$

where $\Theta^{(1)}$ refers to Θ at initialization, $\mathrm{opt}^{(t+1)}(\cdot)$ is an optimizer that minimizes the lower-level loss function $l\left(\mathcal{G}, \mathbf{Y}, \Theta^{(t)}, \theta\right)$ at the $(t+1)$-th epoch. From the perspective of the dynamic system, by applying the chain rule and unrolling the training of lower-level problem with Equation (5.2), the meta-gradient $\nabla_{\mathcal{G}} b$ can be written as

$$\nabla_{\mathcal{G}} b = \nabla_{\mathcal{G}} b\left(\mathbf{Y}, \Theta^{(T)}, \mathbf{F}\right) + \sum_{t=0}^{T-2} A_t B_{t+1} \ldots B_{T-1} \nabla_{\theta^{(T)}} b\left(\mathbf{Y}, \Theta^{(T)}, \mathbf{F}\right) \tag{5.3}$$

where $A_t = \nabla_{\mathcal{G}} \Theta^{(t+1)}$ and $B_t = \nabla_{\Theta^{(t)}} \Theta^{(t+1)}$. However, Equation (5.3) is computationally expensive in both time and space. To further speed up the computation, we adopt a first-order approximation of the meta-gradient [230] and simplify the meta-gradient as

$$\nabla_{\mathcal{G}} b \approx \nabla_{\Theta^{(T)}} b\left(\mathbf{Y}, \Theta^{(T)}, \mathbf{F}\right) \cdot \nabla_{\mathcal{G}} \Theta^{(T)} \tag{5.4}$$

Since the input graph is undirected, the derivative of the symmetric adjacency matrix $\mathbf{A}$ can

be computed as follows by applying the chain rule of a symmetric matrix [195].

$$\nabla_{\mathbf{A}} b \leftarrow \nabla_{\mathbf{A}} b + (\nabla_{\mathbf{A}} b)^T - \operatorname{diag}(\nabla_{\mathbf{A}} b) \tag{5.5}$$

For the node feature matrix $\mathbf{X}$, its derivative is equal to the partial derivative $\nabla_{\mathbf{X}} b$ since it is often an asymmetric matrix.

Graph poisoning with meta-gradient. After computing the meta-gradient of the bias function $\nabla_{\mathcal{G}} b$, we aim to poison the input graph guided by $\nabla_{\mathcal{G}} b$. We introduce two poisoning strategies: (1) continuous poisoning and (2) discretized poisoning.

A – Continuous poisoning attack. The continuous poisoning attack is straightforward by reweighting edges in the graph. We first compute the meta-gradient of the bias function $\nabla_{\mathbf{A}} b$, then use it to poison the input graph in a gradient descent-based updating rule as follows.

$$\mathbf{A} \leftarrow \mathbf{A} - \eta \nabla_{\mathbf{A}} b \tag{5.6}$$

where η is a learning rate to control the magnitude of the poisoning attack. The learning rate should satisfy $\eta \leq \frac{B}{\|\nabla_{\mathbf{A}}\|_{1,1}}$ to ensure the constraint on the budgeted attack.

B – Discretized poisoning attack. The discretized poisoning attack aims to select a set of edges to be added/deleted. It is guided by a poisoning preference matrix defined as follows.

$$\nabla_{\mathbf{A}} = (\mathbf{1} - 2\mathbf{A}) \circ \nabla_{\mathbf{A}} b \tag{5.7}$$

where $\mathbf{1}$ is an all-one matrix with the same dimension as $\mathbf{A}$, and $\circ$ denotes the Hadamard product. A large positive $\nabla_{\mathbf{A}}[i, j]$ indicates strong preference in adding an edge, if nodes i and j are not connected (i.e., positive $\nabla_{\mathbf{A}} b[i, j]$, positive $(\mathbf{1} - 2\mathbf{A})[i, j]$), or deleting an edge, if nodes i and j are connected (i.e., negative $\nabla_{\mathbf{A}} b[i, j]$, negative $(\mathbf{1} - 2\mathbf{A})[i, j]$). Then a greedy selection strategy is applied to find the set of edges $\mathcal{E}_{\text{attack}}$ to be added/deleted.

$$\mathcal{E}_{\text{attack}} = \operatorname{topk}(\nabla_{\mathbf{A}}, \delta) \tag{5.8}$$

where $\operatorname{topk}(\nabla_{\mathbf{A}}, \delta)$ selects δ entries with highest preference score in $\nabla_{\mathbf{A}}$. Note that, if we only want to add edges without any deletion, all negative entries in $\nabla_{\mathbf{A}} b$ should be zeroed out before computing Equation (5.7). Likewise, if edges are only expected to be deleted, all positive entries should be zeroed out.

Remarks. Poisoning node feature matrix $\mathbf{X}$ follows the same steps as poisoning adjacency matrix $\mathbf{A}$ without applying Equation (5.5).

Overall framework. Algorithm 5.1 summarizes the detailed steps on fairness attack with

Algorithm 5.1: FATE

Given : an undirected graph $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$, the set of training nodes $\mathcal{V}_{\text{train}}$, fairness-related auxiliary information matrix $\mathbf{F}$, total budget B, budget in step i δ_i, the bias function b, number of pre-training epochs T.

Find : the poisoned graph $\widetilde{\mathcal{G}}$.

1 poisoned graph $\widetilde{\mathcal{G}} \leftarrow \mathcal{G}$;
2 cumulative budget $\Delta \leftarrow 0$;
3 step counter $i \leftarrow 0$;
4 **while** $\Delta < B$ **do**
5 $\nabla_{\widetilde{\mathcal{G}}} b \leftarrow 0$;
6 **for** $t = 1$ *to* T **do**
7 update $\Theta^{(t)}$ to $\Theta^{(t+1)}$ with a gradient-based optimizer (e.g., Adam);
8 **end**
9 get $\mathbf{Y}^{(T)}$ and $\Theta^{(T)}$;
10 compute the meta-gradient $\nabla_{\mathcal{G}} b \leftarrow \nabla_{\Theta^{(T)}} b\left(\mathbf{Y}, \Theta^{(T)}, \mathbf{F}\right) \cdot \nabla_{\mathcal{G}} \Theta^{(T)}$;
11 **if** *attack the adjacency matrix* **then**
12 compute the derivative $\nabla_{\widetilde{\mathbf{A}}} b \leftarrow \nabla_{\widetilde{\mathbf{A}}} b + \left(\nabla_{\widetilde{\mathbf{A}}} b\right)^T - \operatorname{diag}\left(\nabla_{\widetilde{\mathbf{A}}} b\right)$;
13 **end**
14 **if** *discretized poisoning attack* **then**
15 compute the poisoning preference matrix $\nabla_{\widetilde{\mathbf{A}}}$ by Equation (5.7);
16 select the edges to poison in $\nabla_{\widetilde{\mathbf{A}}}$ with budget δ_i by Equation (5.8);
17 update the corresponding entries in $\widetilde{\mathcal{G}}$;
18 **else**
19 update $\widetilde{\mathcal{G}}$ by Equation (5.6) with budget δ_i;
20 **end**
21 $\Delta \leftarrow \Delta + \delta_i$;
22 $i \leftarrow i + 1$;
23 **end**
24 **return** $\widetilde{\mathcal{G}}$;

FATE. To be specific, after initialization (line 1), we pre-train the surrogate graph learning model (steps 6 – 8) and get the pre-trained surrogate model $\Theta^{(T)}$ as well as learning results $\mathbf{Y}^{(T)}$ (step 9). After that, we compute the meta gradient of the bias function (steps 10 – 13) and perform either discretized attack or continuous attack based on the interest of the attacker (i.e., discretized poisoning attack in steps 14 – 17 or continuous poisoning attack in steps 18 – 20).

Limitations. Since FATE leverages the meta-gradient to poison the input graph, it requires the bias function $b\left(\mathbf{Y}, \Theta^{(T)}, \mathbf{F}\right)$ to be differentiable in order to calculate the meta-gradient $\nabla_{\mathcal{G}} b$. In Sections 5.4 and 5.5, we present carefully chosen bias functions for FATE. And we leave it for future work on exploring the ability of FATE in attacking other fairness definitions.

Moreover, though the meta-gradient can be efficiently computed via auto-differentiation in many deep learning packages (e.g., PyTorch[38], TensorFlow[39]), it requires $O(n^2)$ space complexity to store the meta-gradient when attacking fairness via edge flipping. It is still a challenging open problem on how to efficiently compute the meta-gradient in terms of space. One possible remedy for discretized attack might be a low-rank approximation on the perturbation matrix formed by $\mathcal{E}_{\text{attack}}$. Since the difference between the benign graph and poisoned graph are often small and budgeted ($d\left(\mathcal{G}, \widetilde{\mathcal{G}}\right) \leq B$), it is likely that the edge manipulations may be around a few set of nodes, which makes the perturbation matrix to be an (approximately) low-rank matrix.

5.4 INSTANTIATION #1: STATISTICAL PARITY ON GRAPH NEURAL NETWORKS

Here, we instantiate the FATE framework by attacking statistical parity on graph neural networks in a binary node classification problem with a binary sensitive attribute. We briefly discuss how to choose (1) the surrogate graph learning model used by the attacker, (2) the task-specific loss function in the lower-level optimization problem, and (3) the bias function in the upper-level optimization problem.

Surrogate graph learning model. We assume that the surrogate model to be used by the attacker is a 2-layer linear GCN [231] with different hidden dimensions and model parameters at initialization.

Lower-level loss function. We consider a semi-supervised node classification task for the graph neural network to be attacked. Thus, the lower-level loss function is chosen as the cross entropy between the ground-truth label and the predicted label: $l\left(\mathcal{G}, \mathbf{Y}, \Theta, \theta\right) = \frac{1}{|\mathcal{V}_{\text{train}}|} \sum_{i \in \mathcal{V}_{\text{train}}} \sum_{j=1}^{c} y_{i,j} \ln \widehat{y}_{i,j}$, where $\mathcal{V}_{\text{train}}$ is the set of training nodes with $|\mathcal{V}_{\text{train}}|$ being its cardinality, c is the number of classes, $y_{i,j}$ is a binary indicator of whether node i belongs to class j, and $\widehat{y}_{i,j}$ is the prediction probability of node i belonging to class j.

Upper-level bias function. We aim to attack statistical parity in the upper-level problem, which asks for $\mathrm{P}\left[\hat{y}=1\right] = \mathrm{P}\left[\hat{y}=1|s=1\right]$. Suppose $p\left(\widehat{y}\right)$ is the probability density function (PDF) of $\widehat{y}_{i,1}$ for any node i, and $p\left(\widehat{y}|s=1\right)$ is the PDF of $\widehat{y}_{i,1}$ for any node i belonging to the demographic group with sensitive attribute value $s=1$. We observe that $\mathrm{P}\left[\hat{y}=1\right]$ and $\mathrm{P}\left[\hat{y}=1|s=1\right]$ are equivalent to the cumulative distribution functions (CDF) of $p\left(\widehat{y} < \frac{1}{2}\right)$ and $p\left(\widehat{y} < \frac{1}{2}|s=1\right)$, respectively. To estimate both $\mathrm{P}\left[\hat{y}=1\right]$ and $\mathrm{P}\left[\hat{y}=1|s=1\right]$ with a differentiable function, we first estimate their probability density functions ($p\left(\widehat{y} < \frac{1}{2}\right)$ and $p\left(\widehat{y} < \frac{1}{2}|s=1\right)$) with kernel density estimation (KDE, Definition 5.1).

[38] `https://pytorch.org/`

[39] `https://www.tensorflow.org/`

Definition 5.1. (Kernel density estimation [232]). Given a set of n IID samples $\{x_1, \ldots, x_n\}$ drawn from a distribution with an unknown probability density function f, the kernel density estimation of f at point τ is defined as follows.

$$\widetilde{f}(\tau) = \frac{1}{na} \sum_{i=1}^{n} f_k \left(\frac{\tau - x_i}{a} \right) \tag{5.9}$$

where $\widetilde{f}$ is the estimated probability density function, f_k is the kernel function, and a is a non-negative bandwidth.

Moreover, we assume the kernel function in KDE is the Gaussian kernel $f_k(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$. However, computing the CDF of a Gaussian distribution is non-trivial. Following [233], we leverage a tractable approximation of the Gaussian Q-function as follows.

$$Q(\tau) = F_k(\tau) = \int_{\tau}^{\infty} f_k(x)\, dx \approx e^{-\alpha\tau^2 - \beta\tau - \gamma} \tag{5.10}$$

where $f_k(x) == \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is a Gaussian distribution with zero mean, $\alpha = 0.4920$, $\beta = 0.2887$, and $\gamma = 1.1893$ [234]. The overall workflow of estimating $\mathrm{P}[\hat{y} = 1]$ is as follows.

- For any node i, get its prediction probability $\widehat{y}_{i,1}$ with respect to class 1.
- Estimate the CDF $\mathrm{P}[\hat{y} = 1]$ using a Gaussian KDE with bandwidth a by $\mathrm{P}[\hat{y} = 1] = \frac{1}{n} \sum_{i=1}^{n} \exp\left(-\alpha \left(\frac{0.5 - \widehat{y}_{i,1}}{a} \right)^2 - \beta \left(\frac{0.5 - \widehat{y}_{i,1}}{a} \right) - \gamma \right)$, where $\alpha = 0.4920$, $\beta = 0.2887$, $\gamma = 1.1893$, and $\exp(x) = e^x$.

Note that $\mathrm{P}[\hat{y} = 1 | s = 1]$ can be estimated with a similar procedure with minor modifications. The only modifications needed are: (1) get the prediction probability of nodes with $s = 1$ and (2) compute the CDF using the Gaussian Q-function over nodes with $s = 1$ rather than all nodes in the graph.

5.5 INSTANTIATION #2: INDIVIDUAL FAIRNESS ON GRAPH NEURAL NETWORKS

We provide another instantiation of the FATE framework by attacking individual fairness on graph neural networks. Here, we consider the same surrogate graph learning model (i.e., a 2-layer linear GCN) and the same lower-level loss function (i.e., cross-entropy loss) as described in Section 5.4. To attack individual fairness, we define the upper-level bias function following the principles in [17]. The fairness-related auxiliary information matrix $\mathbf{F}$ is defined

as the oracle symmetric pairwise node similarity matrix $\mathbf{S}$ (i.e., $\mathbf{F} = \mathbf{S}$), where $\mathbf{S}[i, j]$ measures the similarity between node i and node j. Kang et al. [17] define the overall individual bias to be $\mathrm{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)$. Assuming that $\mathbf{Y}$ is the output of an optimization-based graph learning model, $\mathbf{Y}$ can be viewed as a function with respect to the input graph $\mathcal{G}$, which makes $\mathrm{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)$ differentiable with respect to $\mathcal{G}$. Thus, the bias function $b(\cdot)$ can be naturally defined as the overall individual bias of the input graph $\mathcal{G}$, i.e., $b(\mathbf{Y}, \Theta^*, \mathbf{S}) = \mathrm{Tr}\left(\mathbf{Y}^T \mathbf{L_S} \mathbf{Y}\right)$.

5.6 FURTHER DISCUSSIONS ABOUT FATE

In this part, we provide additional discussions about FATE, including (1) other graph learning models from the optimization perspective, (2) the relationship between fairness attacks and the impossibility theorem of fairness, and (3) the relationship between FATE and Metattack, which is another meta learning-based attacking framework on graph neural networks.

5.6.1 Graph Learning from the Optimization Perspective: More Discussions

Here, we discuss four additional non-parameterized graph learning models from the optimization perspective, including PageRank, spectral clustering, matrix factorization-based completion, and first-order LINE.

Model #1: PageRank. It is one of the most successful random walk based ranking algorithm to measure node importance. Mathematically, PageRank solves the linear system

$$\mathbf{r} = c\mathbf{P}\mathbf{r} + (1 - c)\,\mathbf{e} \tag{5.11}$$

where c is the damping factor, $\mathbf{P}$ is the propagation matrix, and $\mathbf{e}$ is the teleportation vector. In PageRank, the propagation matrix $\mathbf{P}$ is often defined as the row-normalized adjacency matrix of a graph $\mathcal{G}$, and the teleportation vector is a uniform distribution $\frac{1}{n}\mathbf{1}$ with $\mathbf{1}$ being a vector filled with 1. Equivalently, given a damping factor c and a teleportation vector $\mathbf{e}$, the PageRank vector $\mathbf{Y} = \mathbf{r}$ can be learned by minimizing the following loss function

$$\min_{\mathbf{r}} c\mathbf{r}^T (\mathbf{I} - \mathbf{P})\,\mathbf{r} + (1 - c)\,\|\mathbf{r} - \mathbf{e}\|_2^2 \tag{5.12}$$

where $c\left(\mathbf{r}^T (\mathbf{I} - \mathbf{P})\,\mathbf{r}\right)$ is a smoothness term, and $(1 - c)\,\|\mathbf{r} - \mathbf{e}\|_2^2$ is a query-specific term. To attack the fairness of PageRank with FATE, the attacker could attack a surrogate PageRank with different choices of damping factor c and/or teleportation vector $\mathbf{e}$.

Model #2: Spectral clustering. It aims to assign nodes into several clusters, such that the intra-cluster connectivity are maximized, and the inter-cluster connectivity are minimized. To find k clusters of nodes, spectral clustering finds a soft cluster membership matrix $\mathbf{Y} = \mathbf{C}$ with orthonormal columns by minimizing the following loss function

$$\min_{\mathbf{C}} \ \mathrm{Tr}\left(\mathbf{C}^T\mathbf{L}\mathbf{C}\right) \quad \text{s.t.}\mathbf{C}^T\mathbf{C} = \mathbf{I} \tag{5.13}$$

where $\mathbf{L}$ is the (normalized) graph Laplacian of the input graph $\mathcal{G}$. It is worth noting that the columns of $\mathbf{C}$ is equivalent to the eigenvectors of $\mathbf{L}$ associated with the smallest k eigenvalues. To attack the fairness of spectral clustering with FATE, the attacker might attack a surrogate spectral clustering with different number of clusters k.

Model #3: Matrix factorization-based completion. Suppose we have a bipartite graph $\mathcal{G}$ with n_1 users, n_2 items, and m interactions between users and items. Matrix factorization-based completion aims to learn two low-rank matrices: an $n_1 \times z$ matrix $\mathbf{U}$ and an $n_2 \times z$ matrix $\mathbf{V}$, such that the following loss function will be minimized

$$\min_{\mathbf{U},\mathbf{V}} \ \| \, \mathrm{proj}_{\Omega}\left(\mathbf{R} - \mathbf{U}\mathbf{V}^T\right) \|_F^2 + \lambda_1 \|\mathbf{U}\|_F^2 \lambda_2 + \|\mathbf{V}\|_F^2 \tag{5.14}$$

where $\mathbf{A} = \begin{pmatrix} \mathbf{0}_{n_1} & \mathbf{R} \\ \mathbf{R}^T & \mathbf{0}_{n_2} \end{pmatrix}$ with $\mathbf{0}_{n_1}$ being an $n_1 \times n_1$ square matrix filled with 0, $\Omega = \{(i,j) \,|\, (i,j) \text{ is observed}\}$ is the set of observed interaction between any user i and any item j, $\mathrm{proj}_{\Omega}(\mathbf{Z})[i,j]$ equals to $\mathbf{Z}[i,j]$ if $(i,j) \in \Omega$ and 0 otherwise, and λ_1 and λ_2 are two hyperparameters for regularization. To attack the fairness of matrix factorization-based completion with FATE, the attacker could attack a surrogate model with different number of latent factors z.

Model #4: First-order LINE. It is a skip-gram based node embedding model. The key idea of first-order LINE is to map each node into a h-dimensional space such that the dot product of the embeddings of any two connected nodes will be small. To achieve this goal, first-order LINE essentially optimizes the following loss function

$$\max_{\mathbf{H}} \ \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{A}[i,j]\left(\log g\left(\mathbf{H}[j,:]\,\mathbf{H}[i,:]^T\right) + k\mathbb{E}_{j'\sim P_n}\left[\log g\left(-\mathbf{H}[j',:]\,\mathbf{H}[i,:]^T\right)\right]\right) \tag{5.15}$$

where $\mathbf{H}$ is the embedding matrix with $\mathbf{H}[i,:]$ being the h-dimensional embedding of node i, $g(x) = 1/(1+e^{-x})$ is the sigmoid function, k is the number of negative samples, and P_n is the distribution for negative sampling where the sampling probability for node i is proportional to the power of 0.75 of its degree $\deg_i^{0.75}$. For a victim first-order LINE, the attacker could

attack a surrogate first-order LINE with different dimension h in the embedding space and/or a different number of negative samples g.

Remarks. Note that, for a non-parameterized graph learning model (e.g., PageRank, spectral clustering, matrix completion, first-order LINE), we have $\Theta = \{\mathbf{Y}\}$ which is the set of learning results. For example, we have $\Theta = \{\mathbf{r}\}$ for PageRank, $\Theta = \{\mathbf{C}\}$ for spectral clustering, $\Theta = \{\mathbf{U}, \mathbf{V}\}$ and $\Theta = \{\mathbf{H}\}$ for LINE (1st). For parameterized graph learning models (e.g., GCN), Θ refers to the set of learnable weights, e.g., $\Theta = \left\{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\right\}$ for an L-layer GCN.

5.6.2 Relationship between Fairness Attacks and the Impossibility Theorem of Fairness.

The impossibility theorem [235] implies that some fairness definitions may not be satisfied at the same time. However, this may not always be regarded as fairness attacks. More specifically, the impossibility theorem proves that two fairness definitions (e.g., statistical parity and predictive parity) cannot be fully satisfied at the same time, i.e., biases for two fairness definitions are both zero. However, there is no formal theoretical guarantees that ensuring one fairness definition will *always* amplify the bias of another fairness definition. Such formal guarantees might be non-trivial and beyond the scope of this work. The main goal of FATE is to provide insights into the adversarial robustness of fair graph learning and shed light on designing robust and fair graph learning in future studies.

5.6.3 Relationship between FATE and Metattack.

FATE bears subtle differences with Metattack [81], which also utilizes meta learning for adversarial attacks. Note that Metattack aims to degrade the utility of a graph neural network by maximizing the task-specific utility loss (e.g., cross entropy for node classification) in the upper-level optimization problem. Different from Metattack, FATE aims to attack the fairness instead of utility by setting the upper-level optimization problem as maximizing a bias function rather than a utility loss.

5.7 EXPERIMENTS

In this part, we conduct experiments to answer the following research questions:

- *RQ1.* How effective is FATE in exacerbating bias?
- *RQ2.* How effective is FATE in maintaining utility of the downstream task?

Table 5.1: Statistics of the datasets to evaluate FATE.

Dataset	# Nodes	# Edges	# Features	Sensitive Attribute	Label
Pokec-z	7,659	20,550	276	Region	Working field
Pokec-n	6,185	15,321	265	Region	Working field
Bail	18,876	311,870	17	Race	Bail decision

5.7.1 Experimental Settings

Hardware and software specifications. All codes are programmed in Python 3.8.13 and PyTorch 1.12.1. All experiments are performed on a Linux server with 2 Intel Xeon Gold 6240R CPUs and 4 Nvidia Tesla V100 SXM2 GPUs with 32 GB memory.

Dataset descriptions. We use three widely-used public benchmark datasets for fair graph learning: *Pokec-z*, *Pokec-n*, and *Bail*. For each dataset, we use a fixed random seed to split the dataset into training, validation, and test sets with the split ratio being 50%, 25%, and 25%, respectively. The statistics of the datasets, including the number of nodes (# Nodes), the number of edges (# Edges), the number of features (# Features), the sensitive attribute (Sensitive Attribute), and the label (Label), are summarized in Table 5.1.

- *Pokec-z* and *Pokec-n* are two datasets collected from the Slovakian social network *Pokec*, each of which represents a sub-network of a province. Each node in these datasets is a user belonging to two major regions of the corresponding provinces, and each edge is the friendship relationship between two users. The sensitive attribute is the user region, and the label is the working field of a user.

- *Bail* is a similarity graph of criminal defendants during 1990 – 2009. Each node is a defendant during this time period. Two nodes are connected if they share similar past criminal records and demographics. The sensitive attribute is the race of the defendant, and the label is whether the defendant is on bail or not.

Baseline methods. We compare FATE with several baseline methods, including *Random*, *DICE* [236], and *FA-GNN* [85]. Descriptions of the baseline methods are as follows.

- *Random* is a heuristic approach that randomly inserts edges to the input graph.

- *DICE* [236] attacks the utility of a graph learning algorithm by randomly deleting edges within a community and randomly inserting edges across different communities. Similar to [80, 85], the community is defined as the group of nodes with the same class label.

- *FA-GNN* [85] aims to attack the fairness of a graph neural network by inserting adversarial edges that connect nodes in consideration of their class labels and sensitive attribute values.

Evaluation metrics. In our experiments, we aim to evaluate how effective FATE is in (1) attacking the fairness (for effective fairness attacks) and (2) maintaining the utility of node classification (for deceptive fairness attacks). First, to evaluate the performance of FATE in attacking the group fairness, we evaluate the effectiveness using Δ_{SP}, which is defined as follows.

$$\Delta_{\text{SP}} = |P\left[\widehat{y} = 1 \mid s = 1\right] - P\left[\widehat{y} = 1 \mid s = 0\right]| \tag{5.16}$$

where s is the sensitive attribute value of a node, and $\widehat{y}$ is the predicted class label of a node. To evaluate the performance of FATE in attacking the individual fairness, we evaluate the effectiveness using the INFORM bias (Bias) measure [17], which is defined as follows.

$$\text{Bias} = \sum_{i \in \mathcal{V}_{\text{test}}} \sum_{j \in \mathcal{V}_{\text{test}}} \mathbf{S}\left[i, j\right] \|\mathbf{Y}\left[i, :\right] - \mathbf{Y}\left[j, :\right]\|_F^2 \tag{5.17}$$

where $\mathcal{V}_{\text{test}}$ is the set of test nodes, and $\mathbf{S}$ is the oracle pairwise node similarity matrix. The intuition of Equation (5.17) is to measure the individual bias as the squared difference between the learning results of two test nodes, weighted by their pairwise similarity. Second, to evaluate the performance of FATE in maintaining the utility, we use micro F1 score (Micro F1), macro F1 score (Macro F1), and AUC score (AUC).

Detailed parameter settings. Here we provide detailed parameter settings for poisoning the input graph and training the victim model in our experiments.

A – Poisoning the input graph. During poisoning attacks, we set a fixed random seed to control the randomness. The random seed used for each dataset in attacking group/individual fairness are summarized in Table 5.2.

- *Surrogate model training.* We run all methods with a perturbation rate from 0.05 to 0.25 with a step size of 0.05. For FA-GNN [85], we follow its official implementation and use the same surrogate 2-layer GCN [7] with 16 hidden dimensions for poisoning attack.[40] The surrogate GCN in FA-GNN is trained for 500 epochs with a learning rate $1e-2$, weight decay $5e-4$, and dropout rate 0.5. For FATE, we use a 2-layer linear GCN [231] with 16 hidden dimensions for poisoning attacks. And the surrogate linear GCN in FATE is trained for 100 epochs with a learning rate $1e-2$, weight decay $5e-4$, and dropout rate 0.5.

[40] `https://github.com/mengcao327/attack-gnn-fairness`

Table 5.2: Parameter settings on the random seed for all baseline methods in poisoning attacks (Random Seed) and the number of steps for poisoning attacks in FATE (Attacking Steps).

Dataset	Fairness Definition	Attacking Steps	Random Seed
Pokec-n	Statistical parity	3	25
	Individual fairness	3	45
Pokec-z	Statistical parity	3	25
	Individual fairness	5	15
Bail	Statistical parity	3	25
	Individual fairness	3	5

- *Graph topology manipulation.* For Random and DICE, we use the implementations provided in the deeprobust package with the default parameters to add the adversarial edges.[41] For FA-GNN, we add adversarial edges that connect two nodes with different class labels and different sensitive attributes, which provides the most promising performance as shown in [85]. For FATE, suppose we poison the input graph in p ($p > 1$) attacking steps. Then the per-iteration attacking budget in Algorithm 5.1 is set as $\delta_1 = 1$ and $\delta_i = \frac{r|\mathcal{E}|-1}{p-1}$, $\forall i \in \{2, \ldots, p\}$, where r is the perturbation rate, and $|\mathcal{E}|$ is the number of edges. Detailed choices of p for each dataset in attacking group/individual fairness are summarized in Table 5.2.

B – Training the victim model. We use a fixed list of random seed (i.e., $[0, 1, 2, 42, 100]$) to train each victim model 5 times and report the mean and standard deviation. Regarding the victim models in attacking group fairness, we train a 2-layer GCN [7] for 400 epochs and a 2-layer FairGNN [12] for 2000 epochs to evaluate the efficacy of fairness attacks. The hidden dimension, learning rate, weight decay, and dropout rate of GCN and FairGNN are set to 128, $1e-3$, $1e-5$, and 0.5, respectively. The regularization parameters in FairGNN, namely α and β, are set to 100 and 1 for all datasets, respectively. Regarding the victim models in attacking individual fairness, we train a 2-layer GCN [7] and a 2-layer INFORM-GNN [17, 59] for 400 epochs. The hidden dimension, learning rate, weight decay, and dropout rate of GCN and INFORM-GNN are set to 128, $1e-3$, $1e-5$, and 0.5, respectively. The regularization parameter in INFORM-GNN is set to 0.1 for all datasets.

[41] `https://deeprobust.readthedocs.io/`

Table 5.3: Effectiveness of attacking group fairness on GCN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for micro F1 score (Micro F1) and Δ_{SP}. Bold font indicates the success of fairness attack (i.e., Δ_{SP} is increased after fairness attack) with the highest micro F1 score. Underlined cell indicates the failure of fairness attack (i.e., Δ_{SP} is decreased after fairness attack).

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)
Pokec-n	0.00	67.5 ± 0.3	7.1 ± 0.4	67.5 ± 0.3	7.1 ± 0.4	67.5 ± 0.3	7.1 ± 0.4	67.5 ± 0.3	7.1 ± 0.4	67.5 ± 0.3	7.1 ± 0.4
	0.05	68.0 ± 0.3	6.2 ± 0.8	67.6 ± 0.2	6.8 ± 0.3	67.8 ± 0.1	3.3 ± 0.4	$\mathbf{67.9 \pm 0.4}$	$\mathbf{9.3 \pm 1.2}$	$\mathbf{67.9 \pm 0.4}$	$\mathbf{9.3 \pm 1.2}$
	0.10	66.8 ± 0.8	7.3 ± 0.7	66.1 ± 0.5	6.6 ± 1.1	66.0 ± 0.2	11.5 ± 0.6	$\mathbf{68.2 \pm 0.6}$	$\mathbf{9.8 \pm 1.5}$	$\mathbf{68.2 \pm 0.6}$	$\mathbf{9.8 \pm 1.5}$
	0.15	66.7 ± 0.4	8.1 ± 0.4	65.6 ± 0.4	7.7 ± 0.8	66.0 ± 0.4	15.6 ± 3.0	$\mathbf{68.0 \pm 0.3}$	$\mathbf{11.5 \pm 1.0}$	$\mathbf{68.0 \pm 0.3}$	$\mathbf{11.5 \pm 1.0}$
	0.20	66.3 ± 0.7	8.6 ± 1.8	64.2 ± 0.4	3.4 ± 0.9	65.8 ± 0.1	18.4 ± 0.7	$\mathbf{68.2 \pm 0.5}$	$\mathbf{12.0 \pm 1.8}$	$\mathbf{68.2 \pm 0.5}$	$\mathbf{12.0 \pm 1.8}$
	0.25	66.2 ± 0.6	8.5 ± 0.8	63.4 ± 0.2	6.3 ± 0.8	66.6 ± 0.2	23.3 ± 0.5	$\mathbf{68.3 \pm 0.4}$	$\mathbf{12.1 \pm 2.1}$	$\mathbf{68.3 \pm 0.4}$	$\mathbf{12.1 \pm 2.1}$
Pokec-z	0.00	68.4 ± 0.4	6.6 ± 0.9	68.4 ± 0.4	6.6 ± 0.9	68.4 ± 0.4	6.6 ± 0.9	68.4 ± 0.4	6.6 ± 0.9	68.4 ± 0.4	6.6 ± 0.9
	0.05	68.8 ± 0.4	6.4 ± 0.6	67.4 ± 0.5	6.6 ± 0.3	68.1 ± 0.3	2.2 ± 0.4	$\mathbf{68.7 \pm 0.4}$	$\mathbf{6.7 \pm 1.4}$	$\mathbf{68.7 \pm 0.4}$	$\mathbf{6.7 \pm 1.4}$
	0.10	$\mathbf{68.7 \pm 0.3}$	$\mathbf{8.0 \pm 0.6}$	66.5 ± 0.2	6.3 ± 0.8	67.7 ± 0.4	13.5 ± 0.9	$\mathbf{68.7 \pm 0.6}$	$\mathbf{7.5 \pm 0.7}$	$\mathbf{68.7 \pm 0.6}$	$\mathbf{7.5 \pm 0.7}$
	0.15	67.9 ± 0.3	9.1 ± 0.8	65.9 ± 0.8	5.5 ± 1.3	66.6 ± 0.4	16.9 ± 2.6	$\mathbf{69.0 \pm 0.8}$	$\mathbf{8.5 \pm 1.1}$	$\mathbf{69.0 \pm 0.8}$	$\mathbf{8.5 \pm 1.1}$
	0.20	$\mathbf{68.5 \pm 0.4}$	$\mathbf{9.3 \pm 1.0}$	62.9 ± 0.7	8.7 ± 1.0	66.1 ± 0.2	25.4 ± 1.3	$\mathbf{68.5 \pm 0.6}$	$\mathbf{8.8 \pm 1.1}$	$\mathbf{68.5 \pm 0.6}$	$\mathbf{8.8 \pm 1.1}$
	0.25	68.3 ± 0.5	7.3 ± 0.5	63.9 ± 0.4	6.0 ± 1.0	65.5 ± 0.6	22.3 ± 2.8	$\mathbf{68.5 \pm 1.1}$	$\mathbf{8.6 \pm 2.5}$	$\mathbf{68.5 \pm 1.1}$	$\mathbf{8.6 \pm 2.5}$
Bail	0.00	93.1 ± 0.2	8.0 ± 0.2	93.1 ± 0.2	8.0 ± 0.2	93.1 ± 0.2	8.0 ± 0.2	93.1 ± 0.2	8.0 ± 0.2	93.1 ± 0.2	8.0 ± 0.2
	0.05	$\mathbf{92.7 \pm 0.2}$	$\mathbf{8.1 \pm 0.0}$	91.6 ± 0.2	8.5 ± 0.1	91.7 ± 0.1	10.0 ± 0.4	92.6 ± 0.1	8.6 ± 0.1	92.5 ± 0.1	8.6 ± 0.1
	0.10	92.2 ± 0.2	7.8 ± 0.2	90.3 ± 0.1	8.5 ± 0.1	90.5 ± 0.0	10.3 ± 0.4	$\mathbf{92.4 \pm 0.1}$	$\mathbf{8.9 \pm 0.1}$	$\mathbf{92.4 \pm 0.1}$	$\mathbf{8.6 \pm 0.1}$
	0.15	91.9 ± 0.2	7.8 ± 0.1	89.2 ± 0.1	7.7 ± 0.1	90.0 ± 0.2	8.4 ± 0.2	92.2 ± 0.2	9.1 ± 0.1	$\mathbf{92.3 \pm 0.1}$	$\mathbf{9.1 \pm 0.1}$
	0.20	91.6 ± 0.2	7.8 ± 0.1	88.3 ± 0.1	8.3 ± 0.1	89.7 ± 0.1	7.4 ± 0.4	92.2 ± 0.2	9.3 ± 0.1	$\mathbf{92.3 \pm 0.1}$	$\mathbf{9.3 \pm 0.2}$
	0.25	91.4 ± 0.1	8.3 ± 0.1	87.8 ± 0.0	7.8 ± 0.1	89.8 ± 0.2	5.2 ± 0.2	$\mathbf{92.1 \pm 0.1}$	$\mathbf{9.1 \pm 0.2}$	$\mathbf{92.1 \pm 0.1}$	$\mathbf{9.1 \pm 0.3}$

5.7.2 Attacking Statistical Parity on Graph Neural Networks

Settings. We compare FATE with *Random*, *DICE* [236], and *FA-GNN* [85], under the same settings as in Section 5.4. That is, (1) the fairness definition to be attacked is statistical parity, and (2) the downstream task is semi-supervised node classification with binary class label and binary sensitive attribute. The experiments are conducted on 3 real-world datasets, i.e., *Pokec-n*, *Pokec-z*, and *Bail*. Similar to existing works [12, 85], we use the 50%/25%/25% splits for train/validation/test sets. For all baseline methods, the victim models are set to GCN [7] and FairGNN [12], which guarantees statistical parity through adversarial learning. For each dataset, we use a fixed random seed to learn the poisoned graph corresponding to each baseline method. Then we train the victim model 5 times with different random seeds. For a fair comparison, we only attack the adjacency matrix in all experiments.

Effectiveness results. For FATE, we conduct fairness attacks via both edge flipping (FATE-flip in Table 5.3 and edge addition (FATE-add in Table 5.3). For all other baseline methods, edges are only added. The effectiveness of fairness attacks on GCN are presented in Table 5.3. From the table, we have the following key observations. (1) FATE-flip and FATE-add are the only methods that consistently succeed in fairness attacks, while all other baseline methods might fail in some cases (indicated by the underlined Δ_{SP}) because of the decrease in Δ_{SP}. (2) FATE-flip and FATE-add not only amplify Δ_{SP} consistently, but also achieve the best micro F1 score on node classification, which makes FATE-flip and FATE-add more deceptive than all baseline methods. Notably, FATE-flip and FATE-add are able to even increase micro F1

Table 5.4: Effectiveness of attacking group fairness on FairGNN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for micro F1 score (Micro F1) and Δ_{SP}. Bold font indicates the success of fairness attack (i.e., Δ_{SP} is increased after attack) with the highest micro F1 score. Underlined cell indicates the failure of fairness attack (i.e., Δ_{SP} is decreased after attack).

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)	Micro F1 (↑)	Δ_{SP} (↑)
Pokec-n	0.00	68.2 ± 0.4	6.7 ± 2.0	68.2 ± 0.4	6.7 ± 2.0	68.2 ± 0.4	6.7 ± 2.0	68.2 ± 0.4	6.7 ± 2.0	68.2 ± 0.4	6.7 ± 2.0
	0.05	67.4 ± 0.8	8.2 ± 2.5	66.2 ± 0.6	9.9 ± 1.7	66.7 ± 1.2	2.8 ± 1.3	$\mathbf{68.4 \pm 0.2}$	$\mathbf{8.9 \pm 1.8}$	$\mathbf{68.4 \pm 0.2}$	$\mathbf{8.9 \pm 1.8}$
	0.10	67.5 ± 0.5	8.3 ± 1.5	66.3 ± 0.4	9.5 ± 2.1	66.6 ± 0.5	5.9 ± 1.3	$\mathbf{68.5 \pm 0.4}$	$\mathbf{9.5 \pm 1.4}$	$\mathbf{68.5 \pm 0.4}$	$\mathbf{9.5 \pm 1.4}$
	0.15	65.9 ± 0.6	10.4 ± 2.3	65.4 ± 0.6	9.2 ± 2.6	64.8 ± 1.6	9.0 ± 3.3	$\mathbf{68.5 \pm 0.8}$	$\mathbf{10.5 \pm 2.6}$	$\mathbf{68.5 \pm 0.8}$	$\mathbf{10.5 \pm 2.6}$
	0.20	65.4 ± 0.5	10.0 ± 1.5	65.0 ± 0.4	4.4 ± 2.5	65.2 ± 0.2	11.6 ± 2.6	$\mathbf{68.3 \pm 0.3}$	$\mathbf{10.7 \pm 2.3}$	$\mathbf{68.3 \pm 0.3}$	$\mathbf{10.7 \pm 2.3}$
	0.25	65.8 ± 1.1	7.5 ± 1.9	63.8 ± 0.3	5.4 ± 1.8	64.8 ± 0.8	14.2 ± 2.3	$\mathbf{68.5 \pm 0.3}$	$\mathbf{9.1 \pm 3.6}$	$\mathbf{68.5 \pm 0.3}$	$\mathbf{9.1 \pm 3.6}$
Pokec-z	0.00	68.7 ± 0.3	7.0 ± 0.9	68.7 ± 0.3	7.0 ± 0.9	68.7 ± 0.3	7.0 ± 0.9	68.7 ± 0.3	7.0 ± 0.9	68.7 ± 0.3	7.0 ± 0.9
	0.05	67.3 ± 0.6	8.7 ± 2.8	67.5 ± 0.4	8.5 ± 1.3	67.1 ± 1.0	1.7 ± 1.3	$\mathbf{68.7 \pm 0.4}$	$\mathbf{8.0 \pm 0.9}$	$\mathbf{68.7 \pm 0.4}$	$\mathbf{8.0 \pm 0.9}$
	0.10	67.1 ± 0.2	8.6 ± 2.7	66.1 ± 0.3	7.0 ± 2.9	65.9 ± 0.8	6.8 ± 1.7	$\mathbf{68.5 \pm 0.5}$	$\mathbf{9.0 \pm 1.8}$	$\mathbf{68.5 \pm 0.5}$	$\mathbf{9.0 \pm 1.8}$
	0.15	66.8 ± 0.8	8.9 ± 2.2	65.2 ± 0.5	6.6 ± 1.4	64.9 ± 0.9	10.0 ± 1.7	$\mathbf{68.7 \pm 0.5}$	$\mathbf{9.5 \pm 2.2}$	$\mathbf{68.7 \pm 0.5}$	$\mathbf{9.5 \pm 2.2}$
	0.20	66.8 ± 0.7	8.6 ± 3.0	63.7 ± 0.6	6.6 ± 2.9	64.6 ± 0.8	14.2 ± 3.1	$\mathbf{68.8 \pm 0.2}$	$\mathbf{10.4 \pm 1.6}$	$\mathbf{68.8 \pm 0.2}$	$\mathbf{10.4 \pm 1.6}$
	0.25	66.4 ± 0.4	7.9 ± 2.8	63.4 ± 0.4	6.0 ± 2.8	64.0 ± 1.1	14.0 ± 2.0	$\mathbf{68.5 \pm 0.3}$	$\mathbf{10.3 \pm 2.1}$	$\mathbf{68.5 \pm 0.3}$	$\mathbf{10.3 \pm 2.1}$
Bail	0.00	93.9 ± 0.1	8.4 ± 0.2	93.9 ± 0.1	8.4 ± 0.2	93.9 ± 0.1	8.4 ± 0.2	93.9 ± 0.1	8.4 ± 0.2	93.9 ± 0.1	8.4 ± 0.2
	0.05	90.6 ± 1.2	8.3 ± 0.2	89.1 ± 1.2	8.3 ± 0.3	89.1 ± 2.0	10.8 ± 1.1	$\mathbf{93.6 \pm 0.1}$	$\mathbf{9.2 \pm 0.2}$	$\mathbf{93.6 \pm 0.1}$	$\mathbf{9.1 \pm 0.2}$
	0.10	90.1 ± 2.0	8.5 ± 0.6	88.1 ± 1.8	8.2 ± 0.3	87.3 ± 2.2	12.2 ± 1.2	$\mathbf{93.4 \pm 0.1}$	$\mathbf{9.3 \pm 0.2}$	$\mathbf{93.4 \pm 0.1}$	$\mathbf{9.3 \pm 0.2}$
	0.15	90.0 ± 2.0	8.1 ± 0.5	86.9 ± 2.0	8.1 ± 0.5	87.8 ± 2.0	10.9 ± 2.1	$\mathbf{93.3 \pm 0.1}$	$\mathbf{9.2 \pm 0.3}$	$\mathbf{93.3 \pm 0.1}$	$\mathbf{9.2 \pm 0.3}$
	0.20	89.2 ± 2.4	8.4 ± 0.7	85.3 ± 2.7	8.2 ± 0.4	86.0 ± 2.7	11.7 ± 2.4	$\mathbf{93.1 \pm 0.2}$	$\mathbf{9.3 \pm 0.3}$	93.0 ± 0.1	9.4 ± 0.2
	0.25	88.8 ± 2.3	8.2 ± 0.7	85.3 ± 3.3	7.9 ± 0.5	87.0 ± 1.9	8.5 ± 2.6	$\mathbf{93.0 \pm 0.1}$	$\mathbf{9.2 \pm 0.4}$	$\mathbf{93.0 \pm 0.2}$	$\mathbf{9.3 \pm 0.3}$

score on all datasets, while other baseline methods attack the graph neural networks at the expense of utility (micro F1 score). (3) Though FA-GNN could make the model more biased in some cases, it (1) degrades the utility which makes the fairness attacks less deceptive, and (2) cannot guarantee consistent success in fairness attacks on all three datasets as shown by the underlined Δ_{SP}.

The effectiveness results of fairness attacks on FairGNN are presented in Table 5.4. We have the following observations. (1) Even though the surrogate model is linear GCN without fairness consideration, FairGNN, which ensures statistical parity on graph neural networks, cannot mitigate the bias caused by fairness attacks and is vulnerable to fairness attack. (2) FATE-flip and FATE-add are effective and the most deceptive method in fairness attacks. (3) FATE-flip and FATE-add are the only methods that consistently succeed in effective and deceptive fairness attacks, while all other baseline methods degrade the utility and might fail in some cases (indicated by the underlined Δ_{SP}).

All in all, FATE consistently succeeds in fairness attacks while being the most deceptive (i.e., highest micro F1 score), regardless of the victim model.

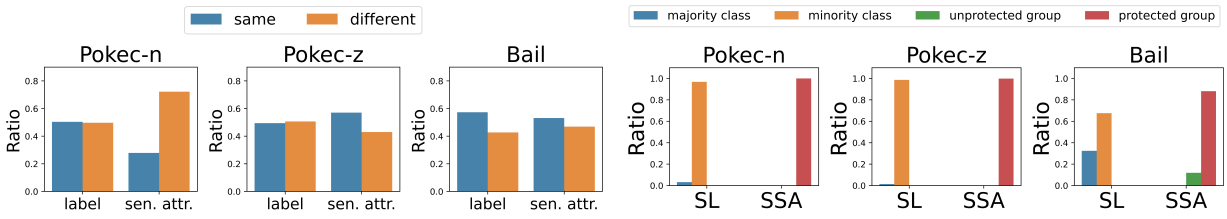
Performance evaluation under other utility metrics. We provide additional evaluation results of the utility using macro F1 score and AUC score. From Tables 5.5 and 5.6, we can see that both metrics are less impacted by different perturbation rates. Thus, it provide extra evidence that FATE framework can achieve deceptive fairness attacks with comparable or even better utility on semi-supervised node classification.

Table 5.5: Macro F1 score and AUC score of attacking group fairness on GCN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for macro F1 score (Macro F1) and AUC score (AUC). Bold font indicates the highest macro F1 score or AUC score.

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)
Pokec-n	0.00	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5
	0.05	65.7 ± 0.3	$\mathbf{70.4 \pm 0.4}$	64.9 ± 0.2	69.8 ± 0.3	64.9 ± 0.2	$\mathbf{70.4 \pm 0.2}$	$\mathbf{66.0 \pm 0.3}$	70.3 ± 0.6	$\mathbf{66.0 \pm 0.3}$	70.3 ± 0.6
	0.10	64.6 ± 0.4	69.6 ± 0.3	63.4 ± 0.6	67.7 ± 0.3	64.1 ± 0.3	70.0 ± 0.1	$\mathbf{66.1 \pm 0.6}$	$\mathbf{70.4 \pm 0.6}$	$\mathbf{66.1 \pm 0.6}$	$\mathbf{70.4 \pm 0.6}$
	0.15	65.1 ± 0.4	69.6 ± 0.1	62.8 ± 0.7	67.3 ± 0.4	64.3 ± 0.6	69.1 ± 0.5	$\mathbf{66.1 \pm 0.2}$	$\mathbf{70.6 \pm 0.6}$	$\mathbf{66.1 \pm 0.2}$	$\mathbf{70.6 \pm 0.6}$
	0.20	64.5 ± 0.5	69.1 ± 0.1	60.9 ± 0.2	64.6 ± 0.2	63.5 ± 0.2	68.0 ± 0.2	$\mathbf{66.4 \pm 0.3}$	$\mathbf{70.7 \pm 0.4}$	$\mathbf{66.4 \pm 0.3}$	$\mathbf{70.7 \pm 0.4}$
	0.25	64.5 ± 0.6	68.8 ± 0.1	59.7 ± 0.6	63.6 ± 0.6	65.0 ± 0.2	69.5 ± 0.3	$\mathbf{66.3 \pm 0.3}$	$\mathbf{70.6 \pm 0.6}$	$\mathbf{66.3 \pm 0.3}$	$\mathbf{70.6 \pm 0.6}$
Pokec-z	0.00	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3
	0.05	68.5 ± 0.4	74.5 ± 0.4	67.2 ± 0.5	74.2 ± 0.3	67.9 ± 0.3	74.5 ± 0.2	$\mathbf{68.6 \pm 0.4}$	$\mathbf{75.2 \pm 0.4}$	$\mathbf{68.6 \pm 0.4}$	$\mathbf{75.2 \pm 0.4}$
	0.10	68.5 ± 0.3	74.8 ± 0.3	66.3 ± 0.2	72.9 ± 0.1	67.5 ± 0.5	73.8 ± 0.3	$\mathbf{68.6 \pm 0.6}$	$\mathbf{75.2 \pm 0.3}$	$\mathbf{68.6 \pm 0.6}$	$\mathbf{75.2 \pm 0.3}$
	0.15	67.8 ± 0.3	74.4 ± 0.3	65.3 ± 1.0	70.0 ± 0.8	66.1 ± 0.6	72.7 ± 0.2	$\mathbf{68.9 \pm 0.7}$	$\mathbf{75.3 \pm 0.2}$	$\mathbf{68.9 \pm 0.7}$	$\mathbf{75.3 \pm 0.2}$
	0.20	68.2 ± 0.4	74.5 ± 0.6	62.6 ± 0.6	68.0 ± 0.9	66.1 ± 0.2	71.9 ± 0.1	$\mathbf{68.4 \pm 0.5}$	$\mathbf{75.1 \pm 0.3}$	$\mathbf{68.4 \pm 0.5}$	$\mathbf{75.1 \pm 0.3}$
	0.25	68.0 ± 0.4	74.0 ± 0.4	63.4 ± 0.4	68.6 ± 0.6	65.3 ± 0.6	71.2 ± 0.3	$\mathbf{68.4 \pm 1.1}$	$\mathbf{74.4 \pm 1.4}$	$\mathbf{68.4 \pm 1.1}$	$\mathbf{74.4 \pm 1.4}$
Bail	0.00	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1
	0.05	$\mathbf{92.0 \pm 0.2}$	95.3 ± 0.2	90.6 ± 0.3	93.8 ± 0.2	90.8 ± 0.1	94.4 ± 0.2	91.8 ± 0.1	$\mathbf{97.1 \pm 0.1}$	91.7 ± 0.1	$\mathbf{97.1 \pm 0.2}$
	0.10	91.4 ± 0.2	94.7 ± 0.3	89.2 ± 0.1	92.2 ± 0.3	89.5 ± 0.1	93.5 ± 0.1	$\mathbf{91.6 \pm 0.2}$	$\mathbf{96.9 \pm 0.1}$	$\mathbf{91.6 \pm 0.2}$	$\mathbf{96.9 \pm 0.1}$
	0.15	91.1 ± 0.2	94.2 ± 0.2	87.8 ± 0.2	91.1 ± 0.2	88.7 ± 0.3	92.5 ± 0.2	91.4 ± 0.2	$\mathbf{96.9 \pm 0.1}$	$\mathbf{91.5 \pm 0.1}$	$\mathbf{96.9 \pm 0.1}$
	0.20	90.7 ± 0.2	94.1 ± 0.1	86.9 ± 0.1	90.2 ± 0.2	88.4 ± 0.1	92.2 ± 0.1	91.3 ± 0.2	$\mathbf{96.8 \pm 0.1}$	$\mathbf{91.4 \pm 0.2}$	$\mathbf{96.8 \pm 0.1}$
	0.25	90.4 ± 0.2	93.4 ± 0.3	86.2 ± 0.1	89.2 ± 0.3	88.5 ± 0.2	92.0 ± 0.1	91.2 ± 0.1	$\mathbf{96.8 \pm 0.1}$	$\mathbf{91.3 \pm 0.2}$	$\mathbf{96.8 \pm 0.1}$

Table 5.6: Macro F1 score and AUC score of attacking group fairness on FairGNN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for macro F1 score (Macro F1) and AUC score (AUC). Bold font indicates the highest macro F1 score or AUC score.

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)
Pokec-n	0.00	65.6 ± 0.3	70.4 ± 0.5	65.6 ± 0.3	70.4 ± 0.5	65.6 ± 0.3	70.4 ± 0.5	65.6 ± 0.3	70.4 ± 0.5	65.6 ± 0.3	70.4 ± 0.5
	0.05	64.3 ± 0.6	68.3 ± 1.1	64.2 ± 0.8	68.4 ± 1.4	63.6 ± 0.7	68.2 ± 0.5	$\mathbf{65.8 \pm 0.5}$	$\mathbf{70.7 \pm 0.4}$	$\mathbf{65.8 \pm 0.5}$	$\mathbf{70.7 \pm 0.4}$
	0.10	63.8 ± 0.2	67.3 ± 1.1	63.4 ± 0.8	67.4 ± 0.6	63.9 ± 0.4	68.3 ± 0.2	$\mathbf{66.0 \pm 0.7}$	$\mathbf{70.8 \pm 0.5}$	$\mathbf{66.0 \pm 0.7}$	$\mathbf{70.8 \pm 0.5}$
	0.15	63.5 ± 0.2	67.8 ± 0.4	60.7 ± 0.7	65.2 ± 1.2	63.1 ± 0.6	67.2 ± 0.5	$\mathbf{65.8 \pm 1.0}$	$\mathbf{70.8 \pm 0.5}$	$\mathbf{65.8 \pm 1.0}$	$\mathbf{70.8 \pm 0.5}$
	0.20	63.1 ± 0.6	67.8 ± 1.1	60.2 ± 1.0	63.7 ± 1.1	62.3 ± 0.6	66.7 ± 0.9	$\mathbf{65.7 \pm 0.7}$	$\mathbf{70.4 \pm 0.5}$	$\mathbf{65.7 \pm 0.7}$	$\mathbf{70.4 \pm 0.5}$
	0.25	62.4 ± 0.3	66.8 ± 0.8	57.3 ± 0.4	61.8 ± 0.7	62.4 ± 1.4	67.6 ± 1.3	$\mathbf{65.1 \pm 1.2}$	$\mathbf{70.1 \pm 0.5}$	$\mathbf{65.1 \pm 1.2}$	$\mathbf{70.1 \pm 0.5}$
Pokec-z	0.00	68.4 ± 0.4	75.1 ± 0.3	68.4 ± 0.4	75.1 ± 0.3	68.4 ± 0.4	75.1 ± 0.3	68.4 ± 0.4	75.1 ± 0.3	68.4 ± 0.4	75.1 ± 0.3
	0.05	66.3 ± 0.9	73.5 ± 0.9	66.4 ± 0.8	72.8 ± 1.2	66.5 ± 1.4	72.6 ± 1.4	$\mathbf{68.4 \pm 0.4}$	$\mathbf{74.7 \pm 0.9}$	$\mathbf{68.4 \pm 0.4}$	$\mathbf{74.7 \pm 0.9}$
	0.10	66.0 ± 0.7	72.9 ± 1.1	64.9 ± 0.8	71.4 ± 0.4	65.2 ± 0.9	71.3 ± 1.7	$\mathbf{68.2 \pm 0.8}$	$\mathbf{75.3 \pm 0.8}$	$\mathbf{68.2 \pm 0.8}$	$\mathbf{75.3 \pm 0.8}$
	0.15	66.0 ± 0.8	71.8 ± 2.1	63.9 ± 0.8	68.5 ± 1.5	63.4 ± 1.5	70.0 ± 1.8	$\mathbf{68.3 \pm 0.5}$	$\mathbf{75.2 \pm 0.6}$	$\mathbf{68.3 \pm 0.5}$	$\mathbf{75.2 \pm 0.6}$
	0.20	65.6 ± 0.9	71.9 ± 1.4	62.1 ± 1.1	67.9 ± 1.8	63.7 ± 0.9	68.9 ± 1.6	$\mathbf{68.3 \pm 0.3}$	$\mathbf{75.5 \pm 0.3}$	$\mathbf{68.3 \pm 0.3}$	$\mathbf{75.5 \pm 0.3}$
	0.25	65.0 ± 0.7	71.2 ± 1.7	61.9 ± 0.4	66.7 ± 1.1	62.8 ± 1.8	69.4 ± 1.5	$\mathbf{68.0 \pm 0.5}$	$\mathbf{75.3 \pm 0.3}$	$\mathbf{68.0 \pm 0.5}$	$\mathbf{75.3 \pm 0.3}$
Bail	0.00	93.3 ± 0.2	97.4 ± 0.1	93.3 ± 0.2	97.4 ± 0.1	93.3 ± 0.2	97.4 ± 0.1	93.3 ± 0.2	97.4 ± 0.1	93.3 ± 0.2	97.4 ± 0.1
	0.05	89.5 ± 1.5	92.8 ± 1.8	87.8 ± 1.3	90.9 ± 1.5	87.8 ± 2.2	91.2 ± 1.8	$\mathbf{93.0 \pm 0.1}$	$\mathbf{97.3 \pm 0.1}$	$\mathbf{93.0 \pm 0.1}$	$\mathbf{97.3 \pm 0.1}$
	0.10	89.1 ± 2.2	92.7 ± 2.5	86.7 ± 2.0	90.1 ± 1.0	85.6 ± 2.7	90.5 ± 1.7	$\mathbf{92.7 \pm 0.1}$	$\mathbf{97.1 \pm 0.1}$	$\mathbf{92.7 \pm 0.1}$	$\mathbf{97.1 \pm 0.1}$
	0.15	88.8 ± 2.2	92.4 ± 2.5	85.2 ± 2.3	88.0 ± 2.8	86.1 ± 2.4	90.3 ± 2.2	$\mathbf{92.6 \pm 0.1}$	$\mathbf{97.0 \pm 0.1}$	$\mathbf{92.6 \pm 0.1}$	$\mathbf{97.0 \pm 0.1}$
	0.20	87.8 ± 2.8	91.6 ± 2.5	83.2 ± 3.1	86.5 ± 3.6	84.1 ± 3.0	89.0 ± 1.5	$\mathbf{92.5 \pm 0.2}$	$\mathbf{97.0 \pm 0.1}$	92.3 ± 0.1	$\mathbf{97.0 \pm 0.1}$
	0.25	87.5 ± 2.6	91.5 ± 2.6	83.3 ± 3.9	87.3 ± 3.7	85.1 ± 2.3	89.6 ± 1.3	$\mathbf{92.3 \pm 0.1}$	$\mathbf{97.0 \pm 0.1}$	$\mathbf{92.3 \pm 0.2}$	$\mathbf{97.0 \pm 0.1}$



(a) (b)

Figure 5.1: Attacking statistical parity with FATE-flip. (a) Ratios of flipped edges that connect two nodes with same/different label or sensitive attribute (sen. attr.). (b) SL (abbreviation for same label) refers to the ratios of flipped edges whose two endpoints are both from the same class. SSA (abbreviation for same sensitive attribute) refers to the ratios of flipped edges whose two endpoints are both from the same demographic group. Majority/minority classes are determined by splitting the training nodes based on their class labels. The protected group is the demographic group with fewer nodes.

Effect of the perturbation rate. From Tables 5.3 and 5.4, we have the following observations. First, Δ_{SP} tends to increase when the perturbation rate increases, which demonstrates the effectiveness of FATE-flip and FATE-add for attacking fairness. Though in some cases Δ_{SP} might have a marginal decrease, FATE-flip and FATE-add still successfully attack the fairness compared with GCN trained on the benign graph by being larger to the Δ_{SP} when the perturbation rate (Ptb.) is 0.00. Second, FATE-flip and FATE-add are deceptive, meaning that the micro F1 scores on the poisoned graphs are close to or even higher than the micro F1 scores on the benign graph. In summary, across different perturbation rates, FATE-flip and FATE-add are effective (i.e., amplifying more bias with higher perturbation rate) and deceptive (i.e., achieving similar or even higher micro F1 score).

Analysis on the manipulated edges. To better understand which edges contribute the most to exacerbate the bias, we characterize the properties of the edges that are flipped by FATE (i.e., FATE-flip) when attacking statistical parity. The reason to only analyzing FATE-flip is that the majority of edges manipulated by FATE-flip on all three datasets is by addition (i.e., flipping from non-existing to existing), so such analysis also provides insights about the edges that are added by FATE-add. Figure 5.1b suggests that, if the two endpoints of a manipulated edge share the same class label or same sensitive attribute value, these two endpoints are most likely from the minority class and protected group. Combining Figures 5.1a and 5.1b, FATE would significantly increase the number of edges that are incident to nodes in the minority class and/or protected group.

Table 5.7: Effectiveness of attacking individual fairness on GCN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for micro F1 score (Micro F1) and INFORM bias (Bias). Bold font indicates the success of fairness attack (i.e., bias is increased after attack) with the highest micro F1 score. Underlined cell indicates the failure of fairness attack (i.e., bias is decreased after attack).

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)
Pokec-n	0.00	67.5 ± 0.3	0.9 ± 0.2	67.5 ± 0.3	0.9 ± 0.2	67.5 ± 0.3	0.9 ± 0.2	67.5 ± 0.3	0.9 ± 0.2	67.5 ± 0.3	0.9 ± 0.2
	0.05	67.6 ± 0.3	1.6 ± 0.3	66.9 ± 0.3	1.6 ± 0.2	$\mathbf{67.8 \pm 0.5}$	$\mathbf{1.9 \pm 0.2}$	$\mathbf{67.8 \pm 0.3}$	$\mathbf{1.2 \pm 0.4}$	67.6 ± 0.3	1.5 ± 0.6
	0.10	67.2 ± 0.5	1.4 ± 0.3	65.3 ± 0.7	1.1 ± 0.1	67.4 ± 0.4	1.2 ± 0.2	$\mathbf{67.9 \pm 0.4}$	$\mathbf{1.3 \pm 0.3}$	67.7 ± 0.4	1.6 ± 0.4
	0.15	67.2 ± 0.3	1.2 ± 0.4	63.9 ± 0.6	1.1 ± 0.2	66.1 ± 0.3	1.5 ± 0.3	$\mathbf{67.8 \pm 0.4}$	$\mathbf{1.2 \pm 0.2}$	67.6 ± 0.2	1.1 ± 0.3
	0.20	66.6 ± 0.3	1.1 ± 0.2	$\underline{63.8 \pm 0.1}$	$\underline{0.8 \pm 0.1}$	65.7 ± 0.6	1.5 ± 0.3	67.3 ± 0.4	1.1 ± 0.3	$\mathbf{68.2 \pm 1.0}$	$\mathbf{1.7 \pm 0.8}$
	0.25	66.7 ± 0.3	1.3 ± 0.4	$\underline{62.5 \pm 0.4}$	$\underline{0.6 \pm 0.0}$	65.2 ± 0.5	1.3 ± 0.4	$\mathbf{67.8 \pm 0.8}$	$\mathbf{1.4 \pm 0.7}$	$\mathbf{67.9 \pm 0.9}$	$\mathbf{1.4 \pm 0.7}$
Pokec-z	0.00	68.4 ± 0.4	2.6 ± 0.7	68.4 ± 0.4	2.6 ± 0.7	68.4 ± 0.4	2.6 ± 0.7	68.4 ± 0.4	2.6 ± 0.7	68.4 ± 0.4	2.6 ± 0.7
	0.05	$\mathbf{69.0 \pm 0.4}$	$\mathbf{3.4 \pm 0.5}$	67.1 ± 0.5	2.7 ± 1.0	68.1 ± 0.4	2.9 ± 0.3	68.7 ± 0.5	2.9 ± 0.5	68.7 ± 0.4	3.1 ± 1.0
	0.10	$\underline{68.7 \pm 0.1}$	$\underline{2.4 \pm 0.5}$	$\underline{66.3 \pm 0.6}$	$\underline{1.7 \pm 0.6}$	$\underline{68.2 \pm 0.5}$	$\underline{1.7 \pm 0.5}$	$\mathbf{69.0 \pm 0.6}$	$\mathbf{2.9 \pm 0.6}$	$\mathbf{69.0 \pm 0.5}$	$\mathbf{3.0 \pm 0.6}$
	0.15	67.9 ± 0.3	2.8 ± 0.3	$\underline{65.5 \pm 0.3}$	$\underline{1.4 \pm 0.3}$	$\underline{67.0 \pm 0.5}$	$\underline{1.3 \pm 0.2}$	68.6 ± 0.5	2.9 ± 0.6	$\mathbf{69.0 \pm 0.7}$	$\mathbf{2.7 \pm 0.4}$
	0.20	$\underline{67.9 \pm 0.3}$	$\underline{2.2 \pm 0.6}$	$\underline{64.2 \pm 0.4}$	$\underline{0.7 \pm 0.3}$	$\underline{66.1 \pm 0.1}$	$\underline{1.6 \pm 0.5}$	68.8 ± 0.4	3.0 ± 0.4	$\mathbf{69.2 \pm 0.4}$	$\mathbf{2.9 \pm 0.3}$
	0.25	$\underline{67.6 \pm 0.3}$	$\underline{1.9 \pm 0.3}$	$\underline{64.2 \pm 0.3}$	$\underline{0.5 \pm 0.1}$	$\underline{65.1 \pm 0.3}$	$\underline{1.9 \pm 0.6}$	69.1 ± 0.3	2.9 ± 0.7	$\mathbf{69.3 \pm 0.3}$	$\mathbf{2.7 \pm 0.6}$
Bail	0.00	93.1 ± 0.2	7.2 ± 0.6	93.1 ± 0.2	7.2 ± 0.6	93.1 ± 0.2	7.2 ± 0.6	93.1 ± 0.2	7.2 ± 0.6	93.1 ± 0.2	7.2 ± 0.6
	0.05	92.1 ± 0.3	8.0 ± 1.9	$\underline{91.8 \pm 0.1}$	$\underline{7.1 \pm 1.1}$	$\underline{91.2 \pm 0.2}$	$\underline{5.6 \pm 0.7}$	$\mathbf{93.0 \pm 0.3}$	7.8 ± 1.0	92.9 ± 0.2	7.7 ± 1.0
	0.10	91.6 ± 0.1	7.3 ± 1.2	$\underline{90.3 \pm 0.1}$	$\underline{6.1 \pm 0.6}$	$\underline{90.3 \pm 0.1}$	$\underline{5.1 \pm 0.4}$	$\mathbf{93.0 \pm 0.1}$	$\mathbf{8.0 \pm 0.7}$	92.9 ± 0.2	7.9 ± 0.8
	0.15	$\underline{91.3 \pm 0.1}$	$\underline{6.5 \pm 0.9}$	$\underline{89.4 \pm 0.0}$	$\underline{4.8 \pm 0.1}$	$\underline{89.8 \pm 0.1}$	$\underline{5.2 \pm 0.1}$	$\mathbf{93.1 \pm 0.1}$	$\mathbf{8.2 \pm 0.6}$	93.0 ± 0.2	7.8 ± 0.8
	0.20	$\underline{91.2 \pm 0.2}$	$\underline{6.6 \pm 0.6}$	$\underline{88.5 \pm 0.1}$	$\underline{4.0 \pm 0.4}$	$\underline{89.3 \pm 0.1}$	$\underline{5.3 \pm 0.4}$	$\mathbf{93.1 \pm 0.1}$	$\mathbf{7.9 \pm 0.6}$	$\mathbf{93.1 \pm 0.1}$	$\mathbf{8.2 \pm 0.6}$
	0.25	$\underline{90.9 \pm 0.1}$	$\underline{6.8 \pm 0.8}$	$\underline{87.4 \pm 0.3}$	$\underline{3.6 \pm 0.5}$	$\underline{88.9 \pm 0.1}$	$\underline{5.4 \pm 0.3}$	92.9 ± 0.1	7.6 ± 0.5	$\mathbf{93.0 \pm 0.2}$	$\mathbf{7.8 \pm 0.7}$

5.7.3 Attacking Individual Fairness on Graph Neural Networks

Settings. To showcase the ability of FATE on attacking individual fairness (Section 5.5), we further compare FATE with the same set of baseline methods (*Random*, *DICE* [236], *FA-GNN* [85]) on the same set of datasets (*Pokec-n*, *Pokec-z*, *Bail*). We follow the settings as in Section 5.5. We use the 50%/25%/25% splits for train/validation/test sets with GCN [7] and INFORM-GNN [17] being the victim models. Please note that INFORM-GNN is an individually fair graph neural network that regularizes the individual bias measure defined in Section 5.5. For each dataset, we use a fixed random seed to learn the poisoned graph corresponding to each baseline method. Then we train the victim model 5 times with different random seeds. And each entry in the oracle pairwise node similarity matrix is computed by the cosine similarity of the corresponding rows in the adjacency matrix. That is, $\mathbf{S}[i,j] = \cos(\mathbf{A}[i,:], A[j,:])$, where $\cos()$ is the function to compute cosine similarity. For a fair comparison, we only attack the adjacency matrix in all experiments.

Effectiveness results. We test FATE with both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods only add edges. Regarding the effectiveness results on GCN (Table 5.7), we have two key observations. (1) FATE-flip and FATE-add are effective – they are the only methods that could consistently attack individual fairness, whereas all other baseline methods mostly fail to attack individual fairness. (2) FATE-flip and FATE-add are deceptive – they achieve comparable or even better utility on all datasets

Table 5.8: Effectiveness of attacking individual fairness on InFoRM-GNN. Fate poisons the graph via both edge flipping (Fate-flip) and edge addition (Fate-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for micro F1 score (Micro F1) and InFoRM bias (Bias). Bold font indicates the success of fairness attack (i.e., bias is increased after attack) with the highest micro F1 score. Underlined cell indicates the failure of fairness attack (i.e., bias is decreased after attack).

Dataset	Ptb.	Random		DICE		FA-GNN		Fate-flip		Fate-add	
		Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)	Micro F1 (↑)	Bias (↑)
Pokec-n	0.00	68.0 ± 0.4	0.5 ± 0.1	68.0 ± 0.4	0.5 ± 0.1	68.0 ± 0.4	0.5 ± 0.1	68.0 ± 0.4	0.5 ± 0.1	68.0 ± 0.4	0.5 ± 0.1
	0.05	67.3 ± 0.5	0.5 ± 0.0	67.0 ± 0.2	0.5 ± 0.0	68.3 ± 0.2	0.5 ± 0.0	$\mathbf{68.4 \pm 0.4}$	$\mathbf{0.6 \pm 0.1}$	68.3 ± 0.4	0.5 ± 0.1
	0.10	67.0 ± 0.2	0.5 ± 0.1	65.5 ± 0.6	0.5 ± 0.1	67.2 ± 0.2	0.4 ± 0.0	68.3 ± 0.6	0.5 ± 0.1	$\mathbf{68.4 \pm 0.5}$	$\mathbf{0.6 \pm 0.1}$
	0.15	66.7 ± 0.5	0.5 ± 0.1	63.8 ± 0.3	0.4 ± 0.0	66.1 ± 0.2	0.4 ± 0.0	$\mathbf{68.3 \pm 0.6}$	$\mathbf{0.6 \pm 0.1}$	68.1 ± 0.7	0.6 ± 0.1
	0.20	66.9 ± 0.3	0.4 ± 0.1	63.7 ± 0.2	0.3 ± 0.0	66.5 ± 0.2	0.4 ± 0.0	67.9 ± 0.8	0.5 ± 0.1	$\mathbf{68.1 \pm 0.7}$	$\mathbf{0.6 \pm 0.1}$
	0.25	66.6 ± 0.5	0.5 ± 0.0	62.2 ± 0.5	0.2 ± 0.1	65.1 ± 0.2	0.4 ± 0.0	$\mathbf{68.7 \pm 0.3}$	$\mathbf{0.6 \pm 0.0}$	68.5 ± 0.8	0.6 ± 0.1
Pokec-z	0.00	68.4 ± 0.5	0.5 ± 0.0	68.4 ± 0.5	0.5 ± 0.0	68.4 ± 0.5	0.5 ± 0.0	68.4 ± 0.5	0.5 ± 0.0	68.4 ± 0.5	0.5 ± 0.0
	0.05	68.9 ± 0.2	0.6 ± 0.1	67.0 ± 0.6	0.6 ± 0.1	68.1 ± 0.7	0.5 ± 0.1	68.7 ± 0.7	0.7 ± 0.1	$\mathbf{68.9 \pm 0.5}$	$\mathbf{0.6 \pm 0.0}$
	0.10	67.9 ± 0.2	0.6 ± 0.1	66.3 ± 0.4	0.5 ± 0.1	68.0 ± 0.6	0.5 ± 0.0	$\mathbf{68.9 \pm 0.6}$	$\mathbf{0.6 \pm 0.0}$	68.8 ± 0.6	0.6 ± 0.0
	0.15	67.6 ± 0.3	0.6 ± 0.1	65.3 ± 0.4	0.4 ± 0.1	66.8 ± 0.3	0.5 ± 0.1	$\mathbf{69.1 \pm 0.5}$	$\mathbf{0.6 \pm 0.0}$	69.0 ± 0.7	0.6 ± 0.1
	0.20	67.7 ± 0.5	0.6 ± 0.1	63.9 ± 0.6	0.3 ± 0.0	66.4 ± 0.6	0.4 ± 0.1	69.1 ± 0.2	0.6 ± 0.0	$\mathbf{69.3 \pm 0.3}$	$\mathbf{0.6 \pm 0.0}$
	0.25	66.8 ± 0.4	0.5 ± 0.1	64.5 ± 0.3	0.2 ± 0.0	65.3 ± 0.4	0.4 ± 0.0	68.9 ± 0.7	0.6 ± 0.0	$\mathbf{69.4 \pm 0.4}$	$\mathbf{0.6 \pm 0.0}$
Bail	0.00	92.8 ± 0.1	1.7 ± 0.1	92.8 ± 0.1	1.7 ± 0.1	92.8 ± 0.1	1.7 ± 0.1	92.8 ± 0.1	1.7 ± 0.1	92.8 ± 0.1	1.7 ± 0.1
	0.05	91.9 ± 0.1	0.4 ± 0.0	91.5 ± 0.1	1.6 ± 0.0	91.3 ± 0.1	1.5 ± 0.1	92.8 ± 0.3	1.7 ± 0.1	92.7 ± 0.1	1.6 ± 0.1
	0.10	91.7 ± 0.1	0.3 ± 0.0	90.4 ± 0.1	1.4 ± 0.1	90.4 ± 0.2	1.5 ± 0.1	92.8 ± 0.1	1.6 ± 0.0	92.8 ± 0.1	1.6 ± 0.0
	0.15	91.5 ± 0.1	0.3 ± 0.0	89.7 ± 0.1	1.4 ± 0.0	90.0 ± 0.1	1.7 ± 0.1	92.8 ± 0.0	1.6 ± 0.1	92.8 ± 0.1	1.6 ± 0.0
	0.20	91.5 ± 0.1	0.3 ± 0.0	88.6 ± 0.2	1.3 ± 0.1	$\mathbf{89.1 \pm 0.1}$	$\mathbf{1.7 \pm 0.1}$	92.8 ± 0.1	1.6 ± 0.0	92.7 ± 0.1	1.5 ± 0.1
	0.25	91.1 ± 0.2	0.3 ± 0.0	87.6 ± 0.2	1.3 ± 0.1	$\mathbf{88.9 \pm 0.1}$	$\mathbf{1.8 \pm 0.1}$	92.6 ± 0.1	1.6 ± 0.1	92.7 ± 0.0	1.6 ± 0.1

compared with the utility on the benign graph. Hence, Fate framework is able to achieve effective and deceptive attacks to exacerbate individual bias.

For attacking individual fairness on InFoRM-GNN (Table 5.8), we get the following observations. (1) For *Pokec-n* and *Pokec-z*, Fate-flip and Fate-add are effective because they are the only methods that could consistently attack individual fairness across different perturbation rates; Fate-flip and Fate-add are deceptive by achieving comparable or higher micro F1 scores on the poisoned graph compared with the micro F1 scores on the benign graph (when perturbation rate is 0.00). (2) For *Bail*, almost all methods fail the fairness attacks. A possible reason is that the adjacency matrix $\mathbf{A}$ of *Bail* is essentially a similarity graph, which causes the pairwise node similarity matrix $\mathbf{S}$ being close to the adjacency matrix $\mathbf{A}$. Even though Fate (and other baseline methods) add adversarial edges to attack individual fairness, regularizing the individual bias defined by $\mathbf{S}$ not only helps to ensure individual fairness, but also provides useful supervision signal in learning a representative node representation due to the similarity between $\mathbf{S}$ and $\mathbf{A}$. (3) Compared with the results in Table 5.7 where GCN is the victim model, InFoRM-GNN is more robust against fairness attacks against individual fairness due to smaller individual bias in Table 5.8.

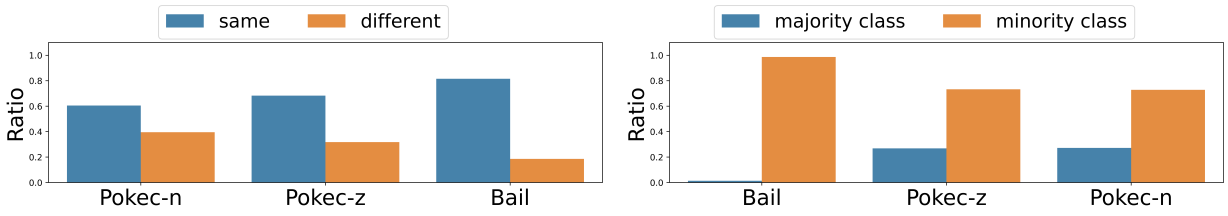
Performance evaluation under other utility metrics. We provide additional results on evaluating the utility of Fate with macro F1 score and AUC score. From Tables 5.9 and 5.10, we can draw a conclusion that Fate can achieve comparable or even better macro F1 scores and AUC scores for both GCN and InFoRM-GNN across different perturbation rates,

Table 5.9: Macro F1 score and AUC score of attacking individual fairness on GCN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for macro F1 score (Macro F1) and AUC score (AUC). Bold font indicates the highest macro F1 score or AUC score.

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)
Pokec-n	0.00	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5	65.3 ± 0.3	69.9 ± 0.5
	0.05	65.2 ± 0.3	70.1 ± 0.2	64.4 ± 0.4	69.5 ± 0.2	65.6 ± 0.6	$\mathbf{71.1 \pm 0.2}$	$\mathbf{65.7 \pm 0.4}$	70.1 ± 0.6	65.5 ± 0.3	70.2 ± 0.8
	0.10	65.2 ± 0.3	69.6 ± 0.5	62.5 ± 0.3	66.9 ± 0.2	65.4 ± 0.6	70.2 ± 0.3	65.5 ± 0.3	70.2 ± 0.7	$\mathbf{65.8 \pm 0.5}$	$\mathbf{70.7 \pm 0.6}$
	0.15	65.4 ± 0.2	69.4 ± 0.3	60.9 ± 0.5	64.8 ± 0.4	64.6 ± 0.2	69.4 ± 0.1	$\mathbf{65.6 \pm 0.4}$	$\mathbf{70.0 \pm 0.5}$	65.4 ± 0.1	69.8 ± 0.7
	0.20	64.9 ± 0.2	69.6 ± 0.3	61.3 ± 0.2	65.5 ± 0.2	63.7 ± 0.5	69.0 ± 0.1	65.2 ± 0.3	69.7 ± 0.6	$\mathbf{65.6 \pm 0.6}$	$\mathbf{70.2 \pm 0.7}$
	0.25	64.7 ± 0.1	69.4 ± 0.2	60.0 ± 0.3	63.5 ± 0.4	63.3 ± 0.5	68.4 ± 0.3	65.4 ± 0.6	69.7 ± 0.7	$\mathbf{65.6 \pm 0.8}$	$\mathbf{69.8 \pm 0.8}$
Pokec-z	0.00	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3	68.2 ± 0.4	75.1 ± 0.3
	0.05	$\mathbf{68.7 \pm 0.4}$	75.0 ± 0.4	67.0 ± 0.5	73.8 ± 0.5	68.0 ± 0.4	75.1 ± 0.5	68.5 ± 0.5	$\mathbf{75.4 \pm 0.2}$	68.5 ± 0.3	75.2 ± 0.4
	0.10	68.5 ± 0.1	75.1 ± 0.5	66.0 ± 0.5	72.5 ± 0.3	67.9 ± 0.6	74.4 ± 0.5	$\mathbf{68.8 \pm 0.5}$	75.5 ± 0.3	$\mathbf{68.8 \pm 0.4}$	$\mathbf{75.6 \pm 0.2}$
	0.15	67.5 ± 0.4	74.4 ± 0.3	65.1 ± 0.4	71.0 ± 0.5	66.8 ± 0.4	72.6 ± 0.2	68.4 ± 0.5	75.5 ± 0.4	$\mathbf{68.8 \pm 0.7}$	$\mathbf{75.6 \pm 0.3}$
	0.20	67.5 ± 0.4	74.7 ± 0.4	63.9 ± 0.3	69.4 ± 0.4	66.1 ± 0.1	71.8 ± 0.2	68.7 ± 0.5	75.5 ± 0.3	$\mathbf{69.0 \pm 0.4}$	$\mathbf{75.6 \pm 0.3}$
	0.25	67.2 ± 0.3	74.1 ± 0.3	63.5 ± 0.4	68.5 ± 0.3	64.8 ± 0.4	70.5 ± 0.4	68.9 ± 0.3	75.6 ± 0.2	$\mathbf{69.1 \pm 0.3}$	$\mathbf{75.7 \pm 0.3}$
Bail	0.00	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1	92.3 ± 0.2	97.4 ± 0.1
	0.05	91.2 ± 0.3	94.8 ± 0.2	90.9 ± 0.1	94.0 ± 0.2	90.3 ± 0.2	94.1 ± 0.2	$\mathbf{92.3 \pm 0.4}$	$\mathbf{97.3 \pm 0.1}$	92.1 ± 0.3	$\mathbf{97.3 \pm 0.1}$
	0.10	90.6 ± 0.1	94.2 ± 0.3	89.0 ± 0.1	91.9 ± 0.2	89.1 ± 0.1	92.9 ± 0.3	$\mathbf{92.3 \pm 0.1}$	$\mathbf{97.3 \pm 0.4}$	92.2 ± 0.2	$\mathbf{97.3 \pm 0.1}$
	0.15	90.3 ± 0.1	94.1 ± 0.2	88.0 ± 0.0	90.9 ± 0.2	88.6 ± 0.2	92.4 ± 0.3	$\mathbf{92.4 \pm 0.1}$	$\mathbf{97.3 \pm 0.0}$	92.3 ± 0.2	$\mathbf{97.3 \pm 0.1}$
	0.20	90.2 ± 0.0	93.9 ± 0.1	87.1 ± 0.2	90.4 ± 0.2	87.9 ± 0.2	91.8 ± 0.2	$\mathbf{92.4 \pm 0.1}$	$\mathbf{97.3 \pm 0.0}$	$\mathbf{92.4 \pm 0.2}$	$\mathbf{97.3 \pm 0.1}$
	0.25	90.9 ± 0.1	93.5 ± 0.2	85.9 ± 0.3	89.6 ± 0.3	87.6 ± 0.1	91.6 ± 0.2	$\mathbf{92.2 \pm 0.2}$	97.2 ± 0.1	$\mathbf{92.2 \pm 0.2}$	$\mathbf{97.3 \pm 0.1}$

Table 5.10: Macro F1 score and AUC score of attacking individual fairness on INFORM-GNN. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add), while all other baseline methods poison the graph via edge addition. Higher is better (↑) for macro F1 score (Macro F1) and AUC score (AUC). Bold font indicates the highest macro F1 score or AUC score.

Dataset	Ptb.	Random		DICE		FA-GNN		FATE-flip		FATE-add	
		Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)	Macro F1 (↑)	AUC (↑)
Pokec-n	0.00	65.4 ± 0.4	70.5 ± 0.8	65.4 ± 0.4	70.5 ± 0.8	65.4 ± 0.4	70.5 ± 0.8	65.4 ± 0.4	70.5 ± 0.8	65.4 ± 0.4	70.5 ± 0.8
	0.05	65.1 ± 0.3	69.9 ± 0.2	64.5 ± 0.4	69.3 ± 0.2	65.6 ± 0.3	$\mathbf{70.8 \pm 0.1}$	$\mathbf{65.9 \pm 0.4}$	70.7 ± 0.8	65.8 ± 0.5	70.5 ± 0.9
	0.10	64.9 ± 0.2	69.6 ± 0.5	62.7 ± 0.4	67.0 ± 0.2	64.8 ± 0.4	69.8 ± 0.3	65.8 ± 0.5	70.3 ± 1.0	$\mathbf{66.0 \pm 0.4}$	$\mathbf{70.9 \pm 1.1}$
	0.15	64.8 ± 0.4	69.6 ± 0.4	60.7 ± 0.2	64.8 ± 0.2	64.4 ± 0.1	69.2 ± 0.3	65.7 ± 0.6	$\mathbf{70.3 \pm 0.7}$	$\mathbf{65.8 \pm 0.4}$	$\mathbf{70.3 \pm 0.9}$
	0.20	65.1 ± 0.2	69.5 ± 0.3	61.0 ± 0.1	65.4 ± 0.3	63.4 ± 0.4	69.0 ± 0.2	65.5 ± 0.8	70.2 ± 0.9	$\mathbf{65.6 \pm 0.6}$	$\mathbf{70.5 \pm 0.7}$
	0.25	64.6 ± 0.3	69.6 ± 0.2	59.8 ± 0.3	63.4 ± 0.2	63.6 ± 0.3	68.6 ± 0.2	$\mathbf{66.0 \pm 0.5}$	$\mathbf{70.8 \pm 0.3}$	65.9 ± 0.5	70.4 ± 0.8
Pokec-z	0.00	68.3 ± 0.4	75.2 ± 0.2	68.3 ± 0.4	75.2 ± 0.2	68.3 ± 0.4	75.2 ± 0.2	68.3 ± 0.4	75.2 ± 0.2	68.3 ± 0.4	75.2 ± 0.2
	0.05	68.6 ± 0.2	75.1 ± 0.3	66.9 ± 0.5	73.4 ± 0.4	67.8 ± 0.6	75.0 ± 0.3	68.6 ± 0.7	75.1 ± 0.6	$\mathbf{68.7 \pm 0.4}$	$\mathbf{75.4 \pm 0.3}$
	0.10	67.6 ± 0.2	74.3 ± 0.4	66.1 ± 0.4	72.1 ± 0.8	67.7 ± 0.6	73.9 ± 0.6	$\mathbf{68.6 \pm 0.6}$	$\mathbf{75.6 \pm 0.3}$	$\mathbf{68.6 \pm 0.6}$	75.5 ± 0.3
	0.15	67.2 ± 0.3	74.1 ± 0.4	64.9 ± 0.5	71.2 ± 0.3	66.7 ± 0.3	72.3 ± 0.1	$\mathbf{68.9 \pm 0.4}$	$\mathbf{75.4 \pm 0.4}$	$\mathbf{68.9 \pm 0.6}$	$\mathbf{75.4 \pm 0.4}$
	0.20	67.3 ± 0.6	74.4 ± 0.4	63.9 ± 0.6	69.4 ± 0.5	66.0 ± 0.5	71.7 ± 0.2	69.0 ± 0.2	$\mathbf{75.5 \pm 0.2}$	$\mathbf{69.2 \pm 0.4}$	75.4 ± 0.4
	0.25	66.3 ± 0.4	73.9 ± 0.4	63.9 ± 0.6	68.9 ± 0.4	65.0 ± 0.5	70.8 ± 0.2	68.8 ± 0.7	75.6 ± 0.3	$\mathbf{69.3 \pm 0.4}$	$\mathbf{75.8 \pm 0.1}$
Bail	0.00	91.9 ± 0.1	97.2 ± 0.0	91.9 ± 0.1	97.2 ± 0.0	91.9 ± 0.1	97.2 ± 0.0	91.9 ± 0.1	97.2 ± 0.0	91.9 ± 0.1	97.2 ± 0.0
	0.05	91.0 ± 0.1	94.2 ± 0.2	90.5 ± 0.1	93.9 ± 0.1	90.4 ± 0.1	94.2 ± 0.1	$\mathbf{92.0 \pm 0.0}$	$\mathbf{97.1 \pm 0.1}$	91.9 ± 0.2	97.0 ± 0.2
	0.10	90.7 ± 0.2	93.9 ± 0.3	89.3 ± 0.1	92.4 ± 0.3	89.4 ± 0.2	93.3 ± 0.1	$\mathbf{92.0 \pm 0.1}$	$\mathbf{97.0 \pm 0.0}$	91.9 ± 0.1	$\mathbf{97.0 \pm 0.0}$
	0.15	90.5 ± 0.1	93.8 ± 0.3	88.4 ± 0.1	91.4 ± 0.3	88.8 ± 0.2	92.4 ± 0.1	$\mathbf{92.0 \pm 0.1}$	$\mathbf{97.0 \pm 0.0}$	91.9 ± 0.2	$\mathbf{97.0 \pm 0.1}$
	0.20	90.5 ± 0.2	93.7 ± 0.2	87.3 ± 0.2	90.5 ± 0.3	87.8 ± 0.1	91.8 ± 0.1	$\mathbf{92.0 \pm 0.1}$	$\mathbf{96.9 \pm 0.0}$	91.9 ± 0.1	96.8 ± 0.1
	0.25	90.1 ± 0.2	93.4 ± 0.3	85.9 ± 0.2	89.1 ± 0.5	87.4 ± 0.1	91.4 ± 0.1	91.8 ± 0.1	96.8 ± 0.1	$\mathbf{91.9 \pm 0.1}$	$\mathbf{96.9 \pm 0.0}$



(a) (b)

Figure 5.2: Attacking individual fairness with FATE-flip. (a) Ratios of flipped edges that connect two nodes with same/different label. (b) Ratios of flipped edges whose two endpoints are both from the majority/minority class. Majority/minority classes are formed by splitting the training nodes based on their class labels.

which are consistent with our findings when evaluating the utility with micro F1 score. It further proves the ability of FATE on deceptive fairness attacks in the task of semi-supervised node classification.

Effect of the perturbation rate. From Tables 5.7 and Table 5.8, we obtain similar observations as in Section 5.7.2 for *Bail* when the victim model is GCN. While for all other cases, the correlation between the perturbation rate (Ptb.) and the individual bias is weaker. One possible reason is that the individual bias is computed using the pairwise node similarity matrix, which is not impacted by poisoning the adjacency matrix, and the discrepancy between the oracle pairwise node similarity matrix and the benign graph is larger. Since the individual bias is computed using the oracle pairwise node similarity matrix rather than the benign/poisoned adjacency matrix, a higher perturbation rate to poison the adjacency matrix may have less impact on the computation of individual bias.

Analysis on the manipulated edges. Since the majority of edges manipulated by FATE-flip is through addition, we only analyze FATE-flip here. And we believe this analysis also provides insights about the properties of the edges manipulated by FATE-add. From Figure 5.2, we can find out that FATE tends to manipulate edges from the same class (especially from the minority class). In this way, FATE would find edges that could increase the individual bias to make the fairness attacks effective and improve the utility of the minority class to make the fairness attack deceptive.

5.7.4 Tranferability of Fairness Attacks by FATE

For all aforementioned evaluation results, both the surrogate model (linear GCN) and the victim models (i.e., GCN, FairGNN, INFORM-GNN) are convolutional aggregation-based graph neural networks. In this part, we test the transferability of FATE by generating poisoned

graphs on the convolutional aggregation-based surrogate model (i.e., linear GCN) and testing on graph attention network (GAT), which is a non-convolutional aggregation-based graph neural network.

More specifically, we train a graph attention network (GAT) with 8 attention heads for 400 epochs. The hidden dimension, learning rate, weight decay, and dropout rate of GAT are set to 64, $1e-3$, $1e-5$, and 0.5, respectively.

The results of attacking statistical parity and individual fairness with GAT as the victim model are shown in Table 5.11. Even though the surrogate model used by the attacker is a convolutional aggregation-based linear GCN, from the table, it is clear that FATE can consistently succeed in (1) effective fairness attack by increasing Δ_{SP} and the individual bias (Bias) and (2) deceptive attack by offering comparable or even better micro F1 score (Micro F1) when the victim model is not a convolutional aggregation-based model (i.e., GAT). Thus, it shows that the adversarial edges flipped/added by FATE is able to transfer to graph neural networks with different type of aggregation function.

Table 5.11: Transferability of attacking statical parity and individual fairness with FATE on GAT. FATE poisons the graph via both edge flipping (FATE-flip) and edge addition (FATE-add). Higher is better (↑) for micro F1 score (Micro F1) Δ_{SP} and INFORM bias (Bias).

Attacking Statistical Parity							
Method	**Ptb.**	**Pokec-n**		**Pokec-z**		**Bail**	
		Micro F1 (↑)	**Bias (↑)**	**Micro F1 (↑)**	**Δ_{SP} (↑)**	**Micro F1 (↑)**	**Δ_{SP} (↑)**
FATE-flip	0.00	63.8 ± 5.3	4.0 ± 3.2	68.2 ± 0.5	8.6 ± 1.1	89.7 ± 4.2	7.5 ± 0.6
	0.05	63.9 ± 5.5	6.4 ± 5.1	68.3 ± 0.4	10.5 ± 1.3	90.1 ± 3.8	8.1 ± 0.6
	0.10	63.6 ± 5.3	7.9 ± 6.7	67.8 ± 0.4	11.2 ± 1.7	90.3 ± 3.2	8.5 ± 0.6
	0.15	63.7 ± 5.3	7.5 ± 6.1	68.2 ± 0.6	11.2 ± 1.5	90.2 ± 2.7	8.8 ± 0.3
	0.20	64.1 ± 5.6	7.7 ± 6.3	67.8 ± 0.6	11.1 ± 0.9	90.0 ± 2.7	8.7 ± 0.6
	0.25	63.6 ± 5.2	8.5 ± 7.0	68.0 ± 0.4	11.5 ± 1.2	89.9 ± 3.0	8.8 ± 0.5
FATE-add	0.00	63.8 ± 5.3	4.0 ± 3.2	68.2 ± 0.5	8.6 ± 1.1	89.7 ± 4.2	7.5 ± 0.6
	0.05	63.9 ± 5.5	6.4 ± 5.1	68.3 ± 0.4	10.5 ± 1.3	90.2 ± 3.7	8.1 ± 0.7
	0.10	63.6 ± 5.3	7.9 ± 6.7	67.8 ± 0.4	11.2 ± 1.7	90.3 ± 3.2	8.5 ± 0.6
	0.15	63.7 ± 5.3	7.5 ± 6.1	68.2 ± 0.6	11.2 ± 1.5	90.3 ± 2.6	8.8 ± 0.3
	0.20	64.1 ± 5.6	7.7 ± 6.3	67.8 ± 0.6	11.1 ± 0.9	90.1 ± 2.6	8.8 ± 0.5
	0.25	63.6 ± 5.2	8.5 ± 7.0	68.0 ± 0.4	11.5 ± 1.2	89.9 ± 2.9	8.8 ± 0.5
Attacking Individual Fairness							
Method	**Ptb.**	**Pokec-n**		**Pokec-z**		**Bail**	
		Micro F1 (↑)	**Bias (↑)**	**Micro F1 (↑)**	**Bias (↑)**	**Micro F1 (↑)**	**Bias (↑)**
FATE-flip	0.00	63.8 ± 5.3	0.4 ± 0.2	68.2 ± 0.5	0.5 ± 0.1	89.7 ± 4.2	2.5 ± 1.2
	0.05	63.6 ± 5.3	0.5 ± 0.2	68.2 ± 0.8	0.6 ± 0.1	90.0 ± 4.2	2.7 ± 1.1
	0.10	63.7 ± 5.3	0.5 ± 0.2	67.8 ± 0.5	0.6 ± 0.1	90.0 ± 4.0	2.8 ± 1.3
	0.15	63.7 ± 5.4	0.5 ± 0.2	68.2 ± 0.5	0.6 ± 0.2	90.2 ± 3.6	2.8 ± 1.4
	0.20	63.5 ± 5.1	0.5 ± 0.2	68.5 ± 0.5	0.6 ± 0.2	90.2 ± 3.4	2.8 ± 1.2
	0.25	63.5 ± 5.1	0.5 ± 0.2	68.0 ± 0.6	0.6 ± 0.1	90.2 ± 3.1	2.7 ± 1.2
FATE-add	0.00	63.8 ± 5.3	0.4 ± 0.2	68.2 ± 0.5	0.5 ± 0.1	89.7 ± 4.2	2.5 ± 1.2
	0.05	63.9 ± 5.4	0.5 ± 0.2	68.2 ± 0.7	0.6 ± 0.1	90.0 ± 4.6	2.7 ± 1.4
	0.10	63.8 ± 5.4	0.5 ± 0.2	68.2 ± 0.5	0.6 ± 0.2	90.1 ± 4.0	2.8 ± 1.2
	0.15	63.8 ± 5.4	0.5 ± 0.2	68.3 ± 0.2	0.6 ± 0.2	90.1 ± 3.9	2.8 ± 1.2
	0.20	63.7 ± 5.3	0.5 ± 0.2	68.4 ± 0.3	0.6 ± 0.1	90.3 ± 3.2	2.8 ± 1.3
	0.25	63.7 ± 5.3	0.5 ± 0.2	68.4 ± 0.3	0.6 ± 0.1	90.2 ± 3.1	2.8 ± 1.2

CHAPTER 6: CONCLUSION AND FUTURE DIRECTIONS

In this chapter, we conclude our key research contributions and discuss the future research directions in fair graph mining.

6.1 KEY RESEARCH CONTRIBUTIONS

As the first batch of researchers to study fair graph mining, my thesis research aims to understand the *how* and *why* questions for fair graph mining and build an algorithmic foundation of fair graph mining. We study three research tasks associated with three key challenges, namely the auditing challenge, the debiasing challenge, and the safeguarding challenge, arisen from the tension of four foundational key properties (utility, fairness, robustness, and transparency). The main contributions of the thesis can be summarized as follows.

Task 1 – Auditing. We address the auditing task (utility vs. transparency) by understanding *how* the graph mining results would relate to the input graph.

Auditing PageRank. We offer the first solution named AURORA to audit PageRank by finding the influential graph elements, such as edges, nodes, and a subgraph. The key idea to measure the influence of graph elements is to compute the rate of change in a scalar-valued function (e.g., L_p norm) defined over the ranking results. We further prove the $(1 - 1/e)$ optimality and the linear time and space complexities of AURORA. The experiments demonstrate that AURORA is able to provide reasonable and intuitive information to find influential graph elements in linear time with respect to the number of nodes and edges.

Network derivative mining. We introduce the network derivative mining (N2N) problem. It can be helpful to various tasks like explainable graph mining, adversarial graph mining, and sensitive analysis. The goal of N2N problem is to construct a derivative network, each of whose edge quantifies the influence of the corresponding edge of the input graph on the mining results. We develop a generic N2N framework for network derivative mining and instantiate the N2N framework with three classic graph mining algorithms from the optimization perspective. For each graph mining algorithm, we design effective solver for constructing the derivative network in linear time. The experiments demonstrate that N2N can consistently (1) find high-influence edges, whose removal results in up to $10\times$ more deviation than the best competitor, and (2) scale linearly with respect to the graph size.

Uncertainty quantification. We develop the first frequentist-based distribution-free uncertainty quantification method JURYGCN on graph convolutional network. The key idea is to leverage

a jackknife estimator to construct a leave-one-out confidence interval for each node. We avoid model re-training after leaving the loss of a training node out by leveraging influence function with respect to the model parameters. We further discuss how to apply JuryGCN to active learning on node classification and semi-supervised node classification. Experimental results demonstrate that JuryGCN offers the best effectiveness and lowest memory usage than the competitors. Notably, by leveraging the influence function, JuryGCN achieves a 130× speed-up in running time compared with model re-training.

Task 2 – Debiasing. We address the debiasing task (utility vs. fairness) by understanding *how* to make the graph mining process and its results fair.

Individual fairness on graphs. We offer the first principled study of individual fairness on graph mining (InFoRM), including the bias measure, the debiasing algorithms, and the theoretical analysis on the cost of individual fairness. To measure the individual bias, we present a quantitative measure of individual bias on graph mining based on the mining results and the pairwise node similarity matrix. To mitigate the individual bias, we develop three mutually complementary algorithmic frameworks, namely debiasing the input graph, debiasing the mining model, and debiasing the mining results, with three instantiations (PageRank, spectral clustering, and LINE). Moreover, we characterize the cost of individual fairness and find out key factors in relation to the cost. The experiments demonstrate that InFoRM can consistently mitigate up to 97% of bias with small deviation to the vanilla mining results and comparable performance on the downstream tasks.

Degree fairness on graphs. We study how to ensure degree fairness on graph convolutional networks without introducing additional learnable parameters and changing the model architecture. Specifically, we reveal the mathematical root cause of degree unfairness by analyzing how the gradient matrix of weight parameters in GCN is computed. Guided by its computation, we design a pre-processing method RawlsGCN-Graph and an in-processing named RawlsGCN-Grad to mitigate the degree unfairness. The experiments demonstrate that RawlsGCN-Graph and RawlsGCN-Grad can mitigate up to 89% and 71% of the degree unfairness, respectively, with no additional memory consumption and up to 8× speed-up in time compared with the best competitor.

Group fairness with multiple sensitive attributes. We study how to satisfy group fairness on the intersectional groups defined by multiple sensitive attributes from the information-theoretic perspective. Specifically, we develop an end-to-end framework InfoFair that minimizes a variational representation of mutual information between the mining results and the vectorized sensitive attribute. The experiments demonstrate that InfoFair can effectively debias the classification results with respect to one or more sensitive attribute(s) with little sacrifice to the classification accuracy.

Task 3 – Safeguarding. We study *how robust* the fairness aspect of (fair) graph mining models is. Specifically, we aim to achieve fairness attacks on graph learning models, whose goal is to amplify the bias while maintaining the utility on the downstream task. Mathematically, we define the problem as a bi-level optimization problem such that effective and deceptive fairness attacks can be achieved in the upper-level optimization problem and lower-level optimization problem, respectively. Then we introduce a meta learning-based framework FATE to poison the input graph using the meta-gradient of the bias function with respect to the input graph. We instantiate FATE by attacking statistical parity and individual fairness on graph neural networks. The experiments demonstrate that FATE can consistently amplify bias and achieve comparable or even better utility compared with training graph neural networks on the benign graph.

6.2 VISION FOR THE FUTURE

While graph mining increasingly benefit our society, there is rising societal awareness that calls for the trustworthiness of these techniques. My current contributions have laid solid foundation in fair graph mining, which prepares me well for future investigations. In the near future, I will mainly continue investigating fair graph mining under more realistic but also more challenging settings (e.g., with noisy graph, in the online setting, and beyond equitable predictive outcomes). In the long run, I am interested in fundamental theories behind the interconnections among different aspects of trustworthiness and principled solutions to deploy graph mining in an open world.

6.2.1 Short-term Research Plan

In the near future, I will focus on the following three research directions, all of which are closely related to my thesis research about fair graph mining.

Fair graph mining with noisy data. Fair graph mining has been actively studied in recent years, existing techniques almost exclusively rely on learning debiased representations with observed graph and labels, while overlooking the fact that the observations could be noisy. The noisy observations brings several challenges in fair graph mining. First, noisy observations could result in a distributional shift (e.g., covariate shift, concept shift) between distributions of the training set and the test set, which might violate the fundamental assumption about the same distribution between training data and test data in various graph mining techniques. Second, noises in the observations could mislead the fair graph mining model toward sub-optimal training directions, resulting in unfairness in the testing

phase. To tackle these challenges, it is essential to investigate how to ensure fairness with noisy data. We will analyze the theoretical necessity of investigating fairness on the noisy data. Additionally, a possible solution is to regularize the distributional distance between the hidden representations of the training nodes and the test nodes in order to mitigate the impact of distributional shifts caused by noisy observations. We envision that a fair graph mining technique being resilient to noisy data would also facilitate research on the safeguarding task in the thesis.

Fair graph mining in the online setting. The real world is not static with nodes and edges appearing, disappearing, or re-appearing over time. For example, in a real-world graph-based recommender system, the users may have changing user profiles and preferences in the system over-time, rate new item(s) they purchased, or update the rating(s) they provided before. In a road network, a road might be closed due to maintenance or car accident, and new roads (i.e., new edges in the network) can appear after the construction is finished. Despite many previous efforts in fair graph mining, almost all existing techniques assume an offline setting without considering the real-world user feedback and graph dynamics. However, graphs at various timestamps could have different distributions of nodes and edges, and the users of graph mining models might provide user feedback on the mining results (e.g., whether to accept the model predictions) in the meanwhile. Therefore, simply applying the fair graph mining algorithms trained in offline setting would fail to accommodate the impact of graph dynamics and align with the value of users in terms of model fairness. Though a naive solution is to repeatedly apply fair graph mining techniques in offline setting whenever the input graph changes or the user provides feedback, it would impose prohibitively long running time and high computational cost, which is detrimental to the environmental well-being. Thus, it is interesting to investigate how to effectively and efficiently update the fair graph mining algorithm over time and with real-world interactions. It could further strengthen the research depth of the debiasing task in the thesis. And one possible solution is to (1) fine-tune the fair graph mining model when a new node/edge is received and/or (2) introduce a policy network so as to align the model predictions with user interests based on the feedback from end users.

Fair graph mining beyond equitable predictive outcomes. As Graph mining being increasingly deployed in high-stake domains like finance and healthcare, the confidence of model predictions is also needed along with the model predictions. One promising method to quantify model confidence is to construct a prediction set or interval that characterizes the plausible range of values covering the true outcome. However, existing works do not offer insights about the variations in the coverage of prediction sets/intervals among demographic groups or individuals, while only focusing on ensuring equitable predictive performance

measured by the model predictions (e.g., accuracy of the model predictions). For example, a highly confident disease risk prediction may offer good coverage guarantee on the population, who has access to essential medical resources and participates more frequently during data collection (e.g., people with high income level), but would provide uncertain and unreliable predictions with low coverage for others (e.g., people with low income level). This calls for a need to develop graph mining models that are equally confident about its predictions with respect to different demographic groups. A potential solution is to investigate conformal prediction with conditional coverage guarantee [237] on graphs. Specifically, we shall re-investigate whether the exchangeablity of the conformity score holds on graphs under conditional coverage guarantee. Then we may develop a fairness-aware re-weighting strategy to assign higher importance to nodes with high residual errors from the minority group. We believe that a rigorous study of fair graph mining with equal coverage guarantee would further benefit both all tasks in the thesis, i.e., prediction set construction for the auditing task, novel equal coverage graph mining model for the debiasing task, and robust fair graph mining toward uncertainty for the safeguarding task.

6.2.2 Long-term Research Plan

In the long run, the ultimate goal of my research is to achieve *reliable* graph mining *in an open world* that could benefit every aspect of human-being. This goal would require future research along two key directions: model trustworthiness and task complexity.

Model trustworthiness: Interconnections among aspects of trustworthiness. To date, major efforts in trustworthy graph mining aim to guarantee one aspect of trustworthiness (e.g., transparency, fairness, robustness, privacy), while overlooking the interconnections among them. Without deep understanding on their implicit relationships, trustworthy graph mining models would not be trustworthy and conscious enough to adapt the learned knowledge on unknown factors that do not exist previously. Thus, in the long term, it is important to understand the interconnections among different aspects of trustworthiness, which could help make a graph mining model to be trustworthy from multiple perspectives rather than respecting to only one aspect. For example, following our work FATE, we shall theoretically analyze how to certify the fairness of graph mining models with respect to bounded perturbations on the input graph. Moreover, it is also important to explore the potential applications of such general trustworthy graph mining models, such as medical diagnosis and policy making.

Task complexity: Graph mining in the wild. Current graph mining algorithms are primarily developed and evaluated in a closed or domain-specific environment. However, the

real world is often complicated and keeps evolving. As a consequence, existing graph mining algorithms could be vulnerable to the real world with noises, distributional shifts, new users, and unobserved domains. For my long-term future research, I would like to study graph mining in the wild. Compared with graph mining in a closed-world setting, graph mining in the wild should be capable of accommodating the incoming new edges, nodes, and domains that do not exist in the history. To adapt graph mining in such open world, it might rely on (1) a multi-modal learning module that learn knowledge from data in multiple modalities (e.g., text, images, social relationships), (2) a domain-invariant learning framework which enables the knowledge to be transferred across domains, and (3) a lifelong learning framework which continually learns knowledge while keeping useful historical knowledge without catastrophic forgetting. Though being challenging, a solution to open-world graph mining can facilitate the deployment of graph mining algorithms in a complicated real-world environment.

REFERENCES

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," Stanford InfoLab, Tech. Rep., 1999.

[2] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[3] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in Neural Information Processing Systems*, pp. 849–856, 2002.

[4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 701–710.

[5] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *The World Wide Web Conference*, 2015, pp. 1067–1077.

[6] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 855–864.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[9] S. Barocas, M. Hardt, and A. Narayanan, "Fairness in machine learning," *Nips tutorial*, vol. 1, p. 2017, 2017.

[10] A. Rahmattalabi, P. Vayanos, A. Fulginiti, E. Rice, B. Wilder, A. Yadav, and M. Tambe, "Exploring algorithmic fairness in robust graph covering problems," *Advances in Neural Information Processing Systems*, vol. 32, pp. 15 776–15 787, 2019.

[11] S. Tsioutsiouliklis, E. Pitoura, P. Tsaparas, I. Kleftakis, and N. Mamoulis, "Fairness-aware pagerank," in *Proceedings of the Web Conference 2021*, 2021, pp. 3815–3826.

[12] E. Dai and S. Wang, "Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 680–688.

[13] X. Lin, J. Kang, W. Cong, and H. Tong, "Bemap: Balanced message passing for fair graph neural network," *arXiv preprint arXiv:2306.04107*, 2023.

[14] X. Tang, H. Yao, Y. Sun, Y. Wang, J. Tang, C. Aggarwal, P. Mitra, and S. Wang, "Investigating and mitigating degree-related biases in graph convoltuional networks," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 2020, pp. 1435–1444.

[15] P. Li, Y. Wang, H. Zhao, P. Hong, and H. Liu, "On dyadic fairness: Exploring and mitigating bias in graph connections," in *International Conference on Learning Representations*, 2021.

[16] J. Kang, Q. Zhou, and H. Tong, "Jurygcn: Quantifying jackknife uncertainty on graph convolutional networks," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 742–752.

[17] J. Kang, J. He, R. Maciejewski, and H. Tong, "Inform: Individual fairness on graph mining," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 379–389.

[18] J. Kang, Y. Zhu, Y. Xia, J. Luo, H. Tong, and X. Wang, "Rawlsgcn: Towards rawlsian difference principle on graph convolutional network," in *Proceedings of the ACM Web Conference 2022*, 2022.

[19] J. Kang, T. Xie, X. Wu, R. Maciejewski, and H. Tong, "Infofair: Information-theoretic intersectional fairness," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 1455–1464.

[20] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[21] T. H. Haveliwala, "Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 784–796, 2003.

[22] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 2006, pp. 613–622.

[23] A. Y. Ng, A. X. Zheng, and M. I. Jordan, "Link analysis, eigenvectors and stability," in *International Joint Conference on Artificial Intelligence*, 2001, pp. 903–910.

[24] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[25] U. Brandes, M. Gaertler, and D. Wagner, "Experiments on graph clustering algorithms," in *European Symposium on Algorithms*, 2003, pp. 568–579.

[26] L. Donetti and M. A. Munoz, "Detecting network communities: A new systematic and efficient algorithm," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2004, no. 10, p. P10012, 2004.

[27] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 554–560.

[28] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[29] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[30] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," *Advances in neural information processing systems*, vol. 14, 2001.

[31] C. Ding, X. He, and H. D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 606–610.

[32] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 2008, pp. 263–272.

[33] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[34] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[35] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[36] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1025–1035.

[37] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[38] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, vol. 2, no. 2, pp. 109–127, 2021.

[39] Y. Zhou, H. Zheng, X. Huang, S. Hao, D. Li, and J. Zhao, "Graph neural networks: Taxonomy, advances, and trends," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 1, pp. 1–54, 2022.

[40] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, "Certifying and removing disparate impact," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 259–268.

[41] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel, "Fairness through awareness," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 214–226.

[42] M. J. Kusner, J. Loftus, C. Russell, and R. Silva, "Counterfactual fairness," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[43] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[44] Z. Tang, J. Zhang, and K. Zhang, "What-is and how-to for fairness in machine learning: A survey, reflection, and perspective," *arXiv preprint arXiv:2206.04101*, 2022.

[45] S. Tsioutsiouliklis, E. Pitoura, K. Semertzidis, and P. Tsaparas, "Link recommendations for pagerank fairness," in *Proceedings of the ACM Web Conference 2022*, 2022.

[46] M. Kleindessner, S. Samadi, P. Awasthi, and J. Morgenstern, "Guarantees for spectral clustering with fairness constraints," in *International Conference on Machine Learning*, 2019, pp. 3458–3467.

[47] A. Bose and W. Hamilton, "Compositional fairness constraints for graph embeddings," in *International Conference on Machine Learning*, 2019, pp. 715–724.

[48] T. A. Rahman, B. Surma, M. Backes, and Y. Zhang, "Fairwalk: Towards fair graph embedding," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 3289–3295.

[49] M. Buyl and T. De Bie, "Debayes: A bayesian method for debiasing network embeddings," in *International Conference on Machine Learning*, 2020, pp. 1220–1229.

[50] I. Spinelli, S. Scardapane, A. Hussain, and A. Uncini, "Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 3, pp. 344–354, 2021.

[51] O. D. Kose and Y. Shen, "Fast&fair: Training acceleration and bias mitigation for gnns," *Transactions on Machine Learning Research*, 2023.

[52] H. Zhu, G. Fu, Z. Guo, Z. Zhang, T. Xiao, and S. Wang, "Fairness-aware message passing for graph neural networks," *arXiv preprint arXiv:2306.11132*, 2023.

[53] Y. Wang, Y. Zhao, Y. Dong, H. Chen, J. Li, and T. Derr, "Improving fairness in graph neural networks via mitigating sensitive attribute leakage," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 1938–1948.

[54] A. Khajehnejad, M. Khajehnejad, M. Babaei, K. P. Gummadi, A. Weller, and B. Mirzasoleiman, "Crosswalk: Fairness-enhanced node representation learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 11 963–11 970.

[55] O. D. Kose and Y. Shen, "Fairness-aware selective sampling on attributed graphs," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 5682–5686.

[56] Y. Dong, N. Liu, B. Jalaian, and J. Li, "Edits: Modeling and mitigating data bias for graph neural networks," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1259–1269.

[57] Z. Song, Y. Ma, and I. King, "Individual fairness in dynamic financial networks," in *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*, 2022.

[58] P. Xu, Y. Zhou, B. An, W. Ai, and F. Huang, "Gfairhint: Improving individual fairness for graph neural networks via fairness hint," *arXiv preprint arXiv:2305.15622*, 2023.

[59] Y. Dong, J. Kang, H. Tong, and J. Li, "Individual fairness for graph neural networks: A ranking based approach," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021, pp. 300–310.

[60] T. Xie, Y. Ma, J. Kang, H. Tong, and R. Maciejewski, "Fairrankvis: A visual analytics framework for exploring algorithmic fairness in graph mining models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 368–377, 2021.

[61] C. Agarwal, H. Lakkaraju, and M. Zitnik, "Towards a unified framework for fair and stable graph representation learning," *arXiv preprint arXiv:2102.13186*, 2021.

[62] J. Ma, R. Guo, M. Wan, L. Yang, A. Zhang, and J. Li, "Learning fair node representations with graph counterfactual fairness," *arXiv preprint arXiv:2201.03662*, 2022.

[63] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *Advances in Neural Information Processing Systems*, vol. 6, 1993.

[64] J. Wu, J. He, and J. Xu, "Net: Degree-specific graph neural networks for node and graph classification," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 406–415.

[65] Z. Liu, T.-K. Nguyen, and Y. Fang, "Tail-gnn: Tail-node graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1109–1119.

[66] Y. Wang and T. Derr, "Degree-related bias in link prediction," in *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2022, pp. 757–758.

[67] Z. Liu, T.-K. Nguyen, and Y. Fang, "On generalized degree fairness in graph neural networks," *arXiv preprint arXiv:2302.03881*, 2023.

[68] F. Masrour, T. Wilson, H. Yan, P.-N. Tan, and A. Esfahanian, "Bursting the filter bubble: Fairness-aware network link prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 841–848.

[69] J. Rawls, *A theory of justice: Revised edition.* Harvard university press, 2020.

[70] A. Rahmattalabi, S. Jabbari, H. Lakkaraju, P. Vayanos, M. Izenberg, R. Brown, E. Rice, and M. Tambe, "Fair influence maximization: A welfare optimization approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 11 630–11 638.

[71] W. Zhang, J. C. Weiss, S. Zhou, and T. Walsh, "Fairness amidst non-iid graph data: A literature review," *arXiv preprint arXiv:2202.07170*, 2022.

[72] Y. Dong, J. Ma, C. Chen, and J. Li, "Fairness in graph mining: A survey," *arXiv preprint arXiv:2204.09888*, 2022.

[73] J. Kang and H. Tong, "Fair graph mining," in *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 2021, pp. 4849–4852.

[74] J. Kang and H. Tong, "Algorithmic fairness on graphs: methods and trends," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4798–4799.

[75] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[76] Z. Gyongyi and H. Garcia-Molina, "Web spam taxonomy," in *First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb 2005)*, 2005.

[77] A. Cheng and E. Friedman, "Manipulability of pagerank under sybil strategies," 2006.

[78] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[79] M. Sun, J. Tang, H. Li, B. Li, C. Xiao, Y. Chen, and D. Song, "Data poisoning attack against unsupervised node embedding methods," *arXiv preprint arXiv:1810.12881*, 2018.

[80] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2847–2856.

[81] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," *arXiv preprint arXiv:1902.08412*, 2019.

[82] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *International conference on machine learning*. PMLR, 2018, pp. 1115–1124.

[83] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 695–704.

[84] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, "Graph adversarial attack via rewiring," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021, pp. 1161–1169.

[85] H. Hussain, M. Cao, S. Sikdar, D. Helic, E. Lex, M. Strohmaier, and R. Kern, "Adversarial inter-group link injection degrades the fairness of graph neural networks," *arXiv preprint arXiv:2209.05957*, 2022.

[86] A. Y. Ng, A. X. Zheng, and M. I. Jordan, "Stable algorithms for link analysis," in *Proceedings of the 24th annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, 2001, pp. 258–266.

[87] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, K. Jain, V. Mirrokni, and S. Teng, "Robust pagerank and locally computable spam detection features," in *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*, 2008, pp. 69–76.

[88] D. F. Gleich and M. W. Mahoney, "Using local spectral methods to robustify graph-based learning algorithms," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 359–368.

[89] E. Estrada, "Network robustness to targeted attacks. the interplay of expansibility and degree distribution," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 52, no. 4, pp. 563–574, 2006.

[90] W. Ellens and R. E. Kooij, "Graph measures and network robustness," *arXiv preprint arXiv:1311.5064*, 2013.

[91] F. D. Malliaros, V. Megalooikonomou, and C. Faloutsos, "Fast robustness estimation in large social graphs: Communities and anomaly detection," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 942–953.

[92] S. Freitas, D. Yang, S. Kumar, H. Tong, and D. H. Chau, "Evaluating graph vulnerability and robustness using tiger," in *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 2021, pp. 4495–4503.

[93] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 3961–3967.

[94] A. Bojchevski and S. Günnemann, "Certifiable robustness to graph perturbations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[95] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 1399–1407.

[96] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: Deep insights into attack and defense," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019, pp. 4816–4823.

[97] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 66–74.

[98] X. Tang, Y. Li, Y. Sun, H. Yao, P. Mitra, and S. Wang, "Transferring robustness for graph neural network against poisoning attacks," in *Proceedings of the 13th international conference on web search and data mining*, 2020, pp. 600–608.

[99] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank): Defending against adversarial attacks on graphs," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 169–177.

[100] Z. Xu, B. Du, and H. Tong, "Graph sanitation with application to node classification," in *Proceedings of the ACM Web Conference 2022*, 2022.

[101] S. Freitas, D. Yang, S. Kumar, H. Tong, and D. H. Chau, "Graph vulnerability and robustness: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[102] J. Xu, J. Chen, S. You, Z. Xiao, Y. Yang, and J. Lu, "Robustness of deep learning models on graphs: A survey," *AI Open*, vol. 2, pp. 69–78, 2021.

[103] Y. Wang, Y. Yao, H. Tong, F. Xu, and J. Lu, "Discerning edge influence for network embedding," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 429–438.

[104] Y. Wang, Y. Yao, H. Tong, F. Xu, and J. Lu, "Auditing network embedding: An edge influence based approach," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[105] T. Xie, Y. Ma, H. Tong, M. T. Thai, and R. Maciejewski, "Auditing the sensitivity of graph-based ranking with visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1459–1469, 2020.

[106] Z. Chen, P. Li, H. Liu, and P. Hong, "Characterizing the influence of graph elements," *arXiv preprint arXiv:2210.07441*, 2022.

[107] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[108] F. Baldassarre and H. Azizpour, "Explainability techniques for graph convolutional networks," in *International Conference on Machine Learning (ICML) Workshops, 2019 Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[109] H. Yuan, J. Tang, X. Hu, and S. Ji, "Xgnn: Towards model-level explanations of graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020, pp. 430–438.

[110] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schütt, K.-R. Müller, and G. Montavon, "Higher-order explanations of graph neural networks via relevant walks," *arXiv preprint arXiv:2006.03589*, 2020.

[111] M. Vu and M. T. Thai, "Pgm-explainer: Probabilistic graphical model explanations for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 225–12 235, 2020.

[112] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *Advances in neural information processing systems*, vol. 33, pp. 19 620–19 631, 2020.

[113] M. S. Schlichtkrull, N. De Cao, and I. Titov, "Interpreting graph neural networks for nlp with differentiable edge masking," in *International Conference on Learning Representations*, 2021.

[114] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," in *International Conference on Machine Learning*, 2021, pp. 12 241–12 252.

[115] Y. Zhang, D. Defazio, and A. Ramesh, "Relex: A model-agnostic relational model explainer," in *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 2021, pp. 1042–1049.

[116] Q. Huang, M. Yamada, Y. Tian, D. Singh, and Y. Chang, "Graphlime: Local interpretable model explanations for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[117] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri, "Cf-gnnexplainer: Counterfactual explanations for graph neural networks," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 4499–4511.

[118] C. Agarwal, M. Zitnik, and H. Lakkaraju, "Probing gnn explainers: A rigorous theoretical and empirical analysis of gnn explanation methods," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 8969–8996.

[119] C. Agarwal, O. Queen, H. Lakkaraju, and M. Zitnik, "Evaluating explainability for graph neural networks," *Scientific Data*, vol. 10, no. 1, p. 144, 2023.

[120] B. Kang, J. Lijffijt, and T. De Bie, "Explaine: An approach for explaining network embedding-based link predictions," *arXiv preprint arXiv:1904.12694*, 2019.

[121] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[122] M. T. Ribeiro, S. Singh, and C. Guestrin, """ why should i trust you?""" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[123] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[124] H. Liu, Y. Wang, W. Fan, X. Liu, Y. Li, S. Jain, Y. Liu, A. K. Jain, and J. Tang, "Trustworthy ai: A computational perspective," *arXiv preprint arXiv:2107.06641*, 2021.

[125] Y. Dong, S. Wang, Y. Wang, T. Derr, and J. Li, "On structural explanation of bias in graph neural networks," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 316–326.

[126] Y. Dong, S. Wang, J. Ma, N. Liu, and J. Li, "Interpreting unfairness in graph neural networks via training node attribution," *arXiv preprint arXiv:2211.14383*, 2022.

[127] Y. Zhao, Y. Wang, and T. Derr, "Fairness and explainability: Bridging the gap towards fair model explanations," *arXiv preprint arXiv:2212.03840*, 2022.

[128] M. Gori and A. Pucci, "Itemrank: A random-walk based scoring algorithm for recommender engines," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, 2007, pp. 2766–2771.

[129] J. Weng, E.-P. Lim, J. Jiang, and Q. He, "Twitterrank: Finding topic-sensitive influential twitterers," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 2010, pp. 261–270.

[130] F. Radicchi, "Who is the best player ever? a complex network analysis of the history of professional tennis," *PloS one*, vol. 6, no. 2, p. e17249, 2011.

[131] R. Singh, J. Xu, and B. Berger, "Pairwise global alignment of protein interaction networks by matching neighborhood topology," in *Research in computational molecular biology*. Springer, 2007, pp. 16–31.

[132] J. J. Crofts and D. J. Higham, "Googling the brain: Discovering hierarchical and asymmetric network structures, with applications in neuroscience," *Internet Mathematics*, vol. 7, no. 4, pp. 233–254, 2011.

[133] L. Li, Y. Yao, J. Tang, W. Fan, and H. Tong, "Quint: on query-specific optimal networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 985–994.

[134] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International conference on machine learning*. PMLR, 2017, pp. 1885–1894.

[135] J. Ni, H. Tong, W. Fan, and X. Zhang, "Inside the atoms: ranking on a network of networks," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1356–1365.

[136] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau, "Mage: Matching approximate patterns in richly-attributed graphs," in *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 585–590.

[137] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[138] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, 2003.

[139] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Signed networks in social media," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2010, pp. 1361–1370.

[140] L. Takac and M. Zabovsky, "Data analysis in public social networks," in *International scientific conference and international workshop present day trends of innovations*, vol. 1, no. 6. Present Day Trends of Innovations Lamza Poland, 2012.

[141] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler, "Replacing the irreplaceable: Fast algorithms for team member recommendation," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 636–646.

[142] M. Ley, "The dblp computer science bibliography: Evolution, research issues, perspectives," in *String Processing and Information Retrieval: 9th International Symposium, SPIRE 2002 Lisbon, Portugal, September 11–13, 2002 Proceedings 9*. Springer, 2002, pp. 1–10.

[143] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: densification laws, shrinking diameters and possible explanations," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 177–187.

[144] D. E. Knuth, *The stanford graphbase: A platform for combinatorial computing*. Addison-Wesley Reading, 1993, vol. 37.

[145] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 2, 2007.

[146] L. Tang and H. Liu, "Graph mining applications to social network analysis," in *Managing and Mining Graph Data*. Springer, 2010, pp. 487–513.

[147] J. Kang, M. Wang, N. Cao, Y. Xia, W. Fan, and H. Tong, "Aurora: Auditing pagerank on large graphs," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 713–722.

[148] M. Wang, J. Kang, N. Cao, Y. Xia, W. Fan, and H. Tong, "Graph ranking auditing: Problem definition and fast solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 10, pp. 3366–3380, 2020.

[149] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[150] J. Kunegis, "Konect: The koblenz network collection," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 1343–1350.

[151] J. Leskovec and A. Krevl, "SNAP datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun 2014.

[152] C. Chen, R. Peng, L. Ying, and H. Tong, "Network connectivity optimization: Fundamental limits and effective algorithms," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 1167–1176.

[153] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 598–607.

[154] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals," *Chemistry of Materials*, vol. 31, no. 9, pp. 3564–3572, 2019.

[155] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire et al., "Eta prediction with graph neural networks in google maps," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, 2021.

[156] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian, "Bayesian graph neural networks with adaptive connection sampling," in *International conference on machine learning*, 2020, pp. 4094–4104.

[157] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 5829–5836.

[158] Z.-Y. Liu, S.-Y. Li, S. Chen, Y. Hu, and S.-J. Huang, "Uncertainty aware graph gaussian process for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 4957–4964.

[159] X. Zhao, F. Chen, S. Hu, and J.-H. Cho, "Uncertainty aware semi-supervised learning on graph data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 827–12 836, 2020.

[160] M. Stadler, B. Charpentier, S. Geisler, D. Zügner, and S. Günnemann, "Graph posterior network: Bayesian predictive uncertainty for node classification," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[161] R. G. Miller, "The jackknife-a review," *Biometrika*, vol. 61, no. 1, pp. 1–15, 1974.

[162] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya et al., "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Information Fusion*, vol. 76, pp. 243–297, 2021.

[163] J. Tukey, "Bias and confidence in not-quite large sample," *Annals of Mathematical Statistics*, vol. 29, p. 614, 1958.

[164] B. Efron, "Jackknife-after-bootstrap standard errors and influence functions," *Journal of the Royal Statistical Society: Series B (Methodological)*, 1992.

[165] R. F. Barber, E. J. Candes, A. Ramdas, and R. J. Tibshirani, "Predictive inference with the fackknife+," *The Annals of Statistics*, vol. 49, no. 1, pp. 486–507, 2021.

[166] A. Alaa and M. van Der Schaar, "Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions," in *International conference on machine learning*, 2020, pp. 165–174.

[167] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.

[168] P. Erdős, A. Rényi et al., "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[169] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social Networks*, vol. 5, no. 2, pp. 109–137, 1983.

[170] Q. Zhou, L. Li, X. Wu, N. Cao, L. Ying, and H. Tong, "Attent: Active attributed network alignment," in *Proceedings of the Web Conference 2021*, 2021, pp. 3896–3906.

[171] R. D. Cook and S. Weisberg, *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[172] L. T. Fernholz, *Von Mises calculus for statistical functionals*. Springer Science & Business Media, 2012, vol. 19.

[173] X. Wang, H. Liu, C. Shi, and C. Yang, "Be confident! towards trustworthy graph neural networks via confidence calibration," in *Advances in Neural Information Processing Systems*, 2021, pp. 23 768–23 779.

[174] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, 2008.

[175] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu, "Query-driven active surveying for collective classification," in *10th International Workshop on Mining and Learning with Graphs*, 2012, p. 1.

[176] H. Cai, V. W. Zheng, and K. C.-C. Chang, "Active learning for graph embedding," *arXiv preprint arXiv:1705.05085*, 2017.

[177] L. Gao, H. Yang, C. Zhou, J. Wu, S. Pan, and Y. Hu, "Active discriminative network representation learning," in *IJCAI International Joint Conference on Artificial Intelligence*, 2018, pp. 2142–2148.

[178] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," *arXiv preprint arXiv:1708.00489*, 2017.

[179] S. Hu, Z. Xiong, M. Qu, X. Yuan, M.-A. Côté, Z. Liu, and J. Tang, "Graph policy network for transferable active learning on graphs," *arXiv preprint arXiv:2006.13463*, 2020.

[180] Y. C. Ng, N. Colombo, and R. Silva, "Bayesian semi-supervised learning with graph gaussian processes," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[181] Y. Ma, R. Garnett, and J. Schneider, "σ-optimality for active learning on gaussian random fields," *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[182] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[183] P. Bedi and C. Sharma, "Community detection in social networks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 6, no. 3, pp. 115–135, 2016.

[184] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[185] D. Zhou, J. He, H. Yang, and W. Fan, "Sparc: Self-paced network representation for few-shot rare category characterization," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2807–2816.

[186] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," *Advances in neural information processing systems*, vol. 29, 2016.

[187] Y. Wu, L. Zhang, X. Wu, and H. Tong, "Pc-fairness: A unified framework for measuring causality-based fairness," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[188] F. Chierichetti, R. Kumar, S. Lattanzi, and S. Vassilvitskii, "Fair clustering through fairlets," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5029–5037, 2017.

[189] T. Kamishima, S. Akaho, H. Asoh, and J. Sakuma, "Enhancement of the neutrality in recommendation," in *Decisions@ RecSys*, 2012, pp. 8–14.

[190] S. Yao and B. Huang, "Beyond parity: Fairness objectives for collaborative filtering," *Advances in Neural Information Processing Systems*, vol. 30, pp. 2921–2930, 2017.

[191] S. Verma and J. Rubin, "Fairness definitions explained," in *Proceedings of the international workshop on software fairness*, 2018, pp. 1–7.

[192] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 538–543.

[193] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for graph similarity and subgraph matching," in *Proc. Ecol. Inference Conf.*, vol. 17, 2011.

[194] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[195] J. Kang and H. Tong, "N2n: Network derivative mining," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 861–870.

[196] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[197] S. Peng, G. Wang, and D. Xie, "Social influence analysis in social networking big data: Opportunities and challenges," *IEEE Network*, vol. 31, no. 1, pp. 11–17, 2016.

[198] S. Zhang, D. Zhou, M. Y. Yildirim, S. Alcorn, J. He, H. Davulcu, and H. Tong, "Hidden: Hierarchical dense subgraph detection with application to financial fraud detection," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, 2017, pp. 570–578.

[199] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd International ACM SIGIR conference on Research and Development in Information Retrieval*, 2019, pp. 165–174.

[200] T. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang, "Fairness without demographics in repeated loss minimization," in *International Conference on Machine Learning*, 2018, pp. 1929–1938.

[201] K. Guo, K. Zhou, X. Hu, Y. Li, Y. Chang, and X. Wang, "Orthogonal graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, 2022, pp. 3996–4004.

[202] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[203] C. Luo, D. Wu, and D. Wu, "A deep learning approach for credit scoring using credit default swaps," *Engineering Applications of Artificial Intelligence*, vol. 65, 2017.

[204] R. Berk, H. Heidari, S. Jabbari, M. Kearns, and A. Roth, "Fairness in criminal justice risk assessments: The state of the art," *arXiv preprint arXiv:1703.09207*, 2017.

[205] M. A. Ahmad, C. Eckert, and A. Teredesai, "Interpretable machine learning in healthcare," in *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*, 2018, pp. 559–560.

[206] M. B. Zafar, I. Valera, M. G. Rogriguez, and K. P. Gummadi, "Fairness constraints: Mechanisms for fair classification," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 962–970.

[207] S. B. Morris and R. E. Lobsenz, "Significance tests and confidence intervals for the adverse impact ratio," *Personnel Psychology*, vol. 53, no. 1, 2000.

[208] M. Kearns, S. Neel, A. Roth, and Z. S. Wu, "Preventing fairness gerrymandering: Auditing and learning for subgroup fairness," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2564–2572.

[209] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, 1948.

[210] R. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork, "Learning fair representations," in *International conference on machine learning*, 2013, pp. 325–333.

[211] B. H. Zhang, B. Lemoine, and M. Mitchell, "Mitigating unwanted biases with adversarial learning," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 335–340.

[212] A. Ghassami, S. Khodadadian, and N. Kiyavash, "Fairness in supervised learning: An information theoretic approach," in *2018 IEEE international symposium on information theory (ISIT)*, 2018, pp. 176–180.

[213] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm, "Mutual information neural estimation," in *International conference on machine learning*. PMLR, 2018, pp. 531–540.

[214] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *International Conference on Learning Representations*, 2019.

[215] S. Mukherjee, H. Asnani, and S. Kannan, "Ccmi: Classifier based conditional mutual information estimation," in *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 1083–1093.

[216] S. Bickel, M. Brückner, and T. Scheffer, "Discriminative learning under covariate shift," *Journal of Machine Learning Research*, vol. 10, no. 9, 2009.

[217] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017.

[218] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[219] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[220] F. Prost, H. Qian, Q. Chen, E. H. Chi, J. Chen, and A. Beutel, "Toward a better trade-off between performance and fairness with kernel-based distribution matching," *arXiv preprint arXiv:1910.11779*, 2019.

[221] Z. Jiang, X. Han, C. Fan, F. Yang, A. Mostafavi, and X. Hu, "Generalized demographic parity for group fairness," in *International Conference on Learning Representations*, 2022.

[222] C. F. P. Bureau, "CFPB targets unfair discrimination in consumer finance," https://www.consumerfinance.gov/about-us/newsroom/cfpb-targets-unfair-discrimination-in-consumer-finance/, 2022, [Online; accessed 13-April-2023].

[223] D. Solans, B. Biggio, and C. Castillo, "Poisoning attacks on algorithmic fairness," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part I*. Springer, 2021, pp. 162–177.

[224] N. Mehrabi, M. Naveed, F. Morstatter, and A. Galstyan, "Exacerbating algorithmic bias through fairness attacks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8930–8938.

[225] A. Chhabra, A. Singla, and P. Mohapatra, "Fairness degrading adversarial attacks against clustering algorithms," *arXiv preprint arXiv:2110.12020*, 2021.

[226] M.-H. Van, W. Du, X. Wu, and A. Lu, "Poisoning attacks on fair machine learning," in *International Conference on Database Systems for Advanced Applications*. Springer, 2022, pp. 370–386.

[227] L. T. Liu, S. Dean, E. Rolf, M. Simchowitz, and M. Hardt, "Delayed impact of fair machine learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3150–3158.

[228] J. Baumann, A. Hannák, and C. Heitz, "Enforcing group fairness in algorithmic decision making: Utility maximization under sufficiency," in *2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022, pp. 2315–2326.

[229] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[230] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[231] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[232] Y.-C. Chen, "A tutorial on kernel density estimation and recent advances," *Biostatistics & Epidemiology*, vol. 1, no. 1, pp. 161–187, 2017.

[233] J. Cho, G. Hwang, and C. Suh, "A fair classifier using kernel density estimation," *Advances in neural information processing systems*, vol. 33, pp. 15 088–15 099, 2020.

[234] M. López-Benítez and F. Casadevall, "Versatile, accurate, and analytically tractable approximation for the gaussian q-function," *IEEE Transactions on Communications*, vol. 59, no. 4, pp. 917–922, 2011.

[235] K. K. Saravanakumar, "The impossibility theorem of machine fairness–a causal perspective," *arXiv preprint arXiv:2007.06024*, 2020.

[236] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan, "Hiding individuals and communities in a social network," *Nature Human Behaviour*, vol. 2, no. 2, pp. 139–147, 2018.

[237] I. Gibbs, J. J. Cherian, and E. J. Candès, "Conformal prediction with conditional guarantees," *arXiv preprint arXiv:2305.12616*, 2023.