Towards Practical Node Classification for Attributed Graphs: Improving Effectiveness/Scalability and Benchmarking Graph Neural Network-based Methods

Submitted to

Graduate School of Information Science and Technology

Osaka University

January 2023

Seiji MAEKAWA

List of Publications

Journals (Peer-reviewed)

1. [Under review of Information Systems, Minor revision] Seiji Maekawa, Yuya Sasaki, George Fletcher, Makoto Onizuka. GenCAT: Generating Attributed Graphs with Controlled Relationships between Classes, Attributes, and Topology, 40 pages.

2. Seiji Maekawa, Koh Takeuchi, Makoto Onizuka. New Attributed Graph Clustering by Bridging Attribute and Topology Spaces. Information Processing Society of Japan, Vol.28, pp.427-435, August 2020.

International Conferences (Peer-reviewed)

1. Seiji Maekawa, Dan Zhang, Hannah Kim, Sajjadur Rahman and Estevam Hruschka. Low-resource Interactive Active Labeling for Fine-tuning Language Models, Findings of Conference on Empirical Methods in Natural Language Processing (EMNLP), 13 pages, December 2022.

2. Seiji Maekawa, Koki Noda, Yuya Sasaki, Makoto Onizuka. Beyond Real-world Benchmark Datasets: An Empirical Study of Node Classification with GNNs, in Proceedings of NeurIPS Datasets and Benchmarks Track, 12 pages, November 2022.

3. Seiji Maekawa, Yuya Sasaki, George Fletcher, Makoto Onizuka. GNN Transformation Framework for Improving Efficiency and Scalability. in Proceedings of The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLP-KDD), 16 pages, September 2022.

4. Seiji Maekawa, Yuya Sasaki, George Fletcher, Makoto Onizuka. Benchmarking GNNs with GenCAT Workbench. in Proceedings of The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD) Demo track, 4 pages, September 2022.

5. Seiji Maekawa, Santi Saeyor, Takeshi Sakaki, Makoto Onizuka, Effective Candidate Selection and Interpretable Interest Extraction for Follower Prediction on Social Media. in Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI-IAT), pp.120-127, December 2021.

6. Yuya Ogawa, Seiji Maekawa, Yuya Sasaki, Yasuhiro Fujiwara, Makoto Onizuka. Adaptive Node Embedding Propagation for Semi-Supervised Classification. in Proceedings of The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD), pp. 417-433, September 2021.

7. Hiroto Yamaguchi, Yuya Ogawa, Seiji Maekawa, Yuya Sasaki, Makoto Onizuka. Controlling Internal Structure of Communities on Graph Generator. in Proceedings of 2020 IEEE/ACM ASONAM Demos and Exhibitions Track, Vol.1, pp.937-940, December 2020.

8. Seiji Maekawa, Jianpeng Zhang, George Fletcher, Makoto Onizuka. General Generator for Attributed Graphs with Community Structure. in The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD) Graph Embedding and Mining Workshop, September 2019.

Domestic Conference paper (not Peer-reviewed)

1. 前川 政司, 佐々木 勇和, George Fletcher, 鬼塚 真. コミュニティ構造を制御可能な属性付きグラフ生成, 情報処理学会第 83 回全国大会（IPSJ 2021）, 2 pages, March 2021.

2. 小川 裕也, 前川 政司, 佐々木 勇和, 藤原 靖宏, 鬼塚 真. 適応的なノード埋め込みの伝搬による半教師ありノード分類モデル. 情報処理学会第 83 回全国大会（IPSJ 2021）, 2 pages, March 2021.

3. 山口 寛人, 前川 政司, 佐々木 勇和, 鬼塚 真. 時系列グラフにおける着目ノードに特化したリンク予測. 情報処理学会第 83 回全国大会（IPSJ 2021）, 2 pages, March 2021.

4. 前川 政司, 佐々木 勇和, George Fletcher, 鬼塚 真. コミュニティ構造を制御する属性付きグラフ生成, 第 13 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2021）, 8 pages, March 2021.

5. 小川 裕也, 前川 政司, 佐々木 勇和, 藤原 靖宏, 鬼塚 真. 半教師ありノード分類のための適応的ノード埋め込み伝搬ニューラルネットワーク. 第 13 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2021）, 8 pages, March 2021.

6. 山口 寛人, 前川 政司, 佐々木 勇和, 鬼塚 真. 時系列グラフを活用する着目ノードに特化したリンク予測. 第 13 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2021）, 6 pages, March 2021.

7. 前川 政司, George Fletcher, 鬼塚 真, コミュニティ構造を考慮した属性付きグラフ汎用生成機構, 第 11 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2019）, 8 pages, March 2019. [学生プレゼンテーション賞]

8. 小川 裕也, 前川 政司, 竹内 孝, 佐々木 勇和, 鬼塚 真. 隣接性と構造類似性を考慮したグラフクラスタリング. 第 11 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2019）, 7 pages, March 2019.

9. 前川 政司, 竹内 孝, 佐々木 勇和, 鬼塚 真. 属性付きグラフのための非線形関数を用いた接合加重非負値行列分解, 第 10 回データ工学と情報マネジメントに関するフォーラム（DEIM Forum 2018）, 7 pages, March 2018.

Abstract

Graphs appear everywhere in many application domains such as web page links, social networks, computer vision, and gene expressions. Graph processing attracts broad attention and node classification is one of the hottest topics in the graph machine learning field. Machine Learning (ML) methods including Graph Neural Networks (GNNs) are powerful tools for node classification. However, to apply such ML methods to practical applications, several issues remain: hardly leveraging class structure, limited scalability, no comprehensive evaluation, limited data types, limited support for time-series of graphs, and no general-purpose pre-trained models.

The first three limitations regarding effectiveness, scalability, and evaluation are fundamental to all the limitations since basic techniques and frameworks which address the three limitations can be extended to more complicated and practical settings, i.e., graphs with multiple node/edge types and/or time-series information. As for general-purpose pre-trained models, effectiveness, scalability, and evaluation are essential to ensure the model capability, adequate learning within a reasonable time, and the generalizability of the model, respectively. In this sense, overcoming the effectiveness, scalability, and evaluation limitations will be beneficial in overcoming the other limitations. Hence, we addressed these three limitations in this thesis.

This thesis consists of five chapters. First, we describe the research background and discuss prior research works and their limitations in Chapter 1. In Chapter 2, we consider the clustering problem of attributed graphs in which we need to leverage the class structure, i.e., the relationship between the classes, attributes, and topology, in order to achieve high-quality performance. Note that graph clustering can be regarded as unsupervised node classification by assuming that nodes with the same label form a community. Our challenge is how we design an effective clustering method that captures the complicated relationship between the topology and the attributes

in real-world graphs. We propose NAGC, a new attributed graph clustering method that bridges the attribute space and the topology space. The feature of NAGC is two-fold; 1) NAGC learns a projection function between the topology space and the attribute space so as to capture their complicated relationship, and 2) NAGC leverages the positive unlabeled learning to take the effect of partially observed positive edges into the cluster assignment. We conducted experiments extensively to validate that NAGC performs higher than or comparable to prior arts regarding the clustering quality.

In Chapter 3, we propose a framework that automatically transforms non-scalable GNNs into precomputation-based GNNs which are efficient and scalable for large-scale graphs. The advantages of our framework are two-fold; 1) it transforms various non-scalable GNNs to scalable ones so that the transformed ones scale well to large-scale graphs by separating local feature aggregation from weight learning in their graph convolution, 2) it efficiently executes precomputation on GPU for large-scale graphs by decomposing their edges into small disjoint and balanced sets. Through extensive experiments with large-scale graphs, we demonstrate that the transformed GNNs run faster in training time than the original GNNs while achieving competitive accuracy to the state-of-the-art GNNs. Consequently, our transformation framework provides simple and efficient baselines for future research on scalable GNNs.

In Chapter 4, we propose an evaluation framework using synthetic graphs for graph machine learning methods. First, we propose GenCAT, an attributed graph generator for controlling those relationships, which has the following advantages; 1) GenCAT generates graphs with user-specified node degrees and flexibly controls the relationship between nodes and labels by incorporating the connection proportion for each node to classes. 2) Generated attribute values follow user-specified distributions, and users can flexibly control the correlation between the attributes and labels. 3) Graph generation scales linearly to the number of edges. GenCAT is the first generator to support all three of these practical features. Through extensive experiments, we demonstrate that GenCAT can efficiently generate high-quality complex attributed graphs with user-controlled relationships between labels, attributes, and topology. Second, we conduct extensive experiments with a synthetic graph generator that can generate graphs having controlled characteristics for fine-grained analysis. Our empirical studies clarify the strengths and weaknesses of GNNs from four major characteristics of real-world graphs with the class labels of nodes, i.e., 1) class size distributions (balanced vs.

imbalanced), 2) edge connection proportions between classes (homophilic vs. heterophilic), 3) attribute values (biased vs. random), and 4) graph sizes (small vs. large). In addition, to foster future research on GNNs, we publicly release our codebase that allows users to evaluate various GNNs with various graphs. We hope this work offers interesting insights for future research.

Finally, Chapter 5 summarizes this thesis and discusses our future work.

Contents

Chapter 1

Introduction

1.1 Background

Graph is a ubiquitous structure that occurs in many domains such as citation networks [121], web page networks [34], social networks [35], computer vision [52], and gene expressions [15, 65]. Real-world graphs usually have attributes on nodes. Actually, the graph databases support attributed graphs or property graphs [36, 101]. We demonstrate an intuitive example of attributed graphs in Figure 1.1, where each node (paper) has its attributes (bag-of-words) and nodes are connected by citation relationships.

Node classification, a task of predicting the labels of nodes (i.e., assignments of nodes to classes) by using a partially labeled network, is one of the hottest topics since it has wide applications. In this thesis, we call a set of nodes with the same label a *class* and assume that nodes in a class tend to share similar attributes. For example, it is beneficial to predict categories of papers, e.g., Machine Learning, Databases, and Natural Language Processing as we show in Figure 1.1, in order to reduce human efforts in manually labeling papers. Another example is web page networks where nodes indicate web pages, edges indicate hyperlinks between pages, attributes indicate the texts of pages, and class labels indicate the topics of pages. Also, in social networks, nodes indicate user accounts, edges indicate friend/follow relationships between accounts, attributes indicate user information such as user descriptions, and class labels indicate communities of users. As for computer vision, nodes are objects in an image, a pair of objects has an edge if they are neighboring, attributes indicate node factors extracted from an

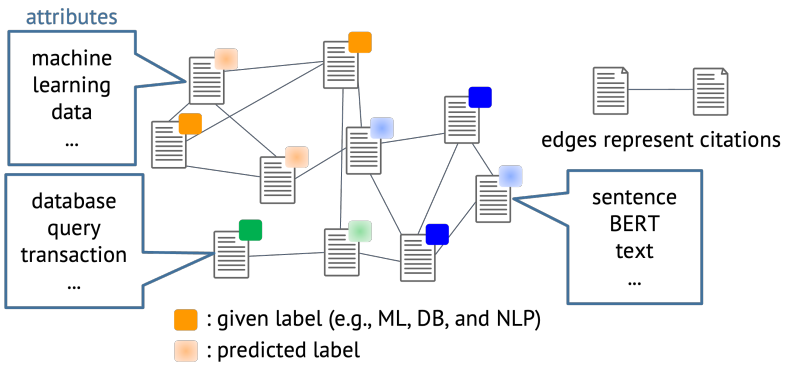


Figure 1.1: Example of node classification on an attributed graph. Nodes indicate papers, edges indicate citations between papers, attributes indicate the bag-of-words of papers, and classes indicate the research fields of papers, e.g., machine learning (ML), databases (DB), and natural language processing (NLP).

image, and class labels indicate categories of objects such as humans, bikes, and cars. As for gene expressions, nodes indicate genes, a pair of genes has an edge if there is a high probability of linkage between the two genes, attributes indicate what genes consist of, and class labels indicate the categories of the functional pathway of nodes. To accurately predict class labels, most methods aim to leverage the interplay between class labels, node attributes, and topology, which we call *class structure*.

Over the past few decades, various methods were proposed to solve a classification task. While non-ML methods, e.g., rule-based systems, were proposed, they require tremendous human effort to improve prediction quality, leading to impractical costs. Graph machine learning methods, e.g., Graph Neural Networks (GNNs), are powerful tools for node classification in terms of prediction quality. However, towards applying such graph machine learning methods to practical applications, several limitations remain: 1) hardly leveraging class structure, i.e., existing methods do not effectively leverage the relationship between the topology and attributes, leading to poor prediction quality, 2) limited scalability, i.e., existing methods do not scale well to large-scale graphs, 3) no comprehensive evaluation, i.e., existing methods have been assessed on a limited variety of graphs, leading to inadequate analysis, 4) limited data types, i.e., existing methods that support various

node types and edge types obtain marginal performance gain compared to methods that ignore such data types, 5) limited support for the time-series of graphs, i.e., existing methods that support the time-series of graphs do not effectively utilize node attributes and suffer from poor scalability, and 6) no general-purpose pre-trained models, i.e., in the graph domain, there are no pre-trained models for various downstream tasks while pre-trained models achieve great success in other domains such as natural language processing and computer vision.

The first three limitations regarding effectiveness, scalability, and evaluation are fundamental to all the limitations since basic techniques and frameworks which address the three limitations can be extended to more complicated and practical settings, i.e., graphs with multiple node/edge types and/or time-series information. For example, some nodes represent people and others represent cities or products in a single graph. As for edges, some edges indicate friend relationships between people, and others indicate purchasing relationships between people and products. Also, those relationships can vary over time. In other words, we focus on static homogeneous graphs, i.e., graphs with a single node/edge type and no time-series information. In fact, the state-of-the-art method for heterogeneous graphs [77] is designed based on GAT [105] which can be applied to only static homogeneous graphs. Also, the state-of-the-art method for graphs with time-series information [53, 108] is designed based on random walk sampling techniques over graphs which is originally proposed for static homogeneous graphs. As for general-purpose pre-trained models, effectiveness, scalability, and evaluation are essential to ensure the model capability, adequate learning within a reasonable time, and the generalizability of the model, respectively. In this sense, overcoming the effectiveness, scalability, and evaluation limitations for homogeneous graphs will be beneficial in overcoming the other limitations. Hence, we addressed these three limitations in this thesis.

1.2 Limitations of Existing Works

To address the limitations of existing works, we discuss them in order.

1.2.1 Hardly Leveraging Class Structure

Traditional graph machine learning methods for node classification and/or clustering [41, 55] typically focus on utilizing only the topology structure, i.e., they aim to capture the relationship between the topology and classes (or communities). This means that they fail to incorporate the interplay between classes, attributes, and topology. Recently few methods [49, 129] have been proposed to capture the class structure in a given attributed graph. However, it is still an open question how we can effectively leverage the virtue of attributed graphs to accurately predict the class labels of nodes. Concretely, existing attributed graph machine learning methods suffer from two limitations. First, they ignore non-linear relationship between the topology and attributes. The second limitation is that they ignore the possibility of missing positive edges, i.e., edges that exist in the real world but are not observed, even though datasets typically do not reflect real-world phenomena completely.

1.2.2 Limited Scalability

With the proliferation of the Web, Internet, and sensor devices, large-scale graphs have been collected. However, most existing graph machine learning methods have not been designed to handle such graphs with over a hundred million nodes and billion edges despite the demand for analyzing large-scale graphs in practice [29, 47]. Hence, it is important to develop efficient and scalable methods for large-scale graphs. Analyzing large-scale graphs is one of the hottest topics and several existing studies have addressed the scalability problem of node classification [37, 82, 111]. However, they still suffer from two major issues. First, though they have proposed their specific algorithms that scale well to large-scale graphs, it is laborsome to apply the same idea to other algorithms that have been widely studied. Second, precomputation schemes that several scalable methods adopt are still not scalable to large-scale graphs since they need to put complete graphs on the GPU memory.

1.2.3 No Comprehensive Evaluation

Several existing studies [30, 38] have addressed benchmarking the performance of graph machine learning methods. A study [30] supports a variety of graph tasks, i.e., node classification, graph classification, link prediction,

and graph regression. However, since only one or two datasets are used for each task, no deep analysis for node classification is provided. Another study [38] focuses on GNNs for materials chemistry. Due to the characteristics of the data, the study considers only graph regression. As a result, the comprehensive evaluation of the methods on a node classification task is challenging since most methods are assessed on well-known but limited benchmark datasets. Consequently, the existing evaluation lacks fine-grained analysis from various characteristics of graphs. Moreover, graph machine learning methods including GNNs have been broadly studied and many works have proposed tons of those algorithms. This makes the above issue more serious. While existing synthetic graph generators are used to mitigate the shortage of datasets, they cannot generate realistic graphs. To clarify the strengths and weaknesses of graph machine learning methods, we require a variety of realistic graphs to evaluate graph machine learning methods.

Our fundamental solutions regarding the first three limitations would provide benefit to future research addressing the rest of the limitations. We will discuss their more details in Section 5.2.

1.3 Contributions and Organization of Thesis

In this thesis, we propose techniques to address the aforementioned limitations against practical applications. Concretely, we propose a method that effectively combines the topology and attributes in order to predict the class labels of nodes in Chapter 2. In Chapter 3, we propose a framework that improves the efficiency of existing graph machine learning methods. Then, we propose an evaluation framework where users can investigate how their own methods work on various graphs in Chapter 4. Finally, in Chapter 5, we conclude this thesis and discuss future work. In the rest of this section, we briefly describe the summaries of our techniques proposed in Chapters 2, 3, and 4, respectively.

1.3.1 Class Structure-aware Graph Clustering Method

Towards effective approaches capturing the class structure in attributed graphs, we address an attributed graph clustering problem for the first step, which can be regarded as unsupervised node classification. Note that we assume that nodes with the same label form a community in this problem, i.e., graph

clustering is a special (and simple) case of node classification with this assumption. To achieve high-quality clustering results, it is necessary to effectively utilize both the topology and attribute information[1]. To address the limitations discussed in Section 1.2.1, Chapter 2, which is based on our research works published in [98, 139], describes the design and implementation of an effective attributed graph clustering method. We show experiments demonstrating that our method outperforms existing methods and balances the effects of the topology and attributes to achieve high-quality clustering results in Section 2.5.

1.3.2 Improving the Scalability of GNNs with Transformations and Graph Decomposition

In Chapter 3, we focus on the scalability of graph machine learning methods while we address improving their effectiveness in Chapter 2. Since deep neural network-based methods have recently achieved state-of-the-art performance on node classification, we focus on node classification with GNNs. To address the scalability issue discussed in Section 1.2.2, in Chapter 3, which is based on our research work published in [81], we propose 1) a framework that automatically transforms non-scalable GNNs into scalable GNNs and 2) a block-wise precomputation scheme that optimally decomposes large-scale graphs into small and balanced blocks each of which can fit into GPU memory. We experimentally validate that our transformation procedure and optimized block-wise precomputation scheme are quite effective in Section 3.4.

1.3.3 Comprehensive Evaluation of GNNs with Flexible Synthetic Graph Generator

While in Chapters 2 and 3 we address the effectiveness and scalability problems of graph machine learning methods, respectively, we address the evaluation problem of graph machine learning methods in Chapter 4. Similarly to Chapter 3, we focus on GNNs due to their promising performance. To address the evaluation problem aforementioned in Section 1.2.3, in Chapter 4,

[1]Since the essential difference between node classification and clustering is whether the loss function includes a supervised loss or not, graph machine learning models for clustering can be applied to node classification with a small modification.

which is based on our research works published in [78, 79, 80, 136, 137, 138], we propose a flexible graph generator supporting various characteristics of graphs and conduct empirical studies of GNNs by using the graph generator. Through extensive experiments, we provide insightful takeaways for future research on GNNs in Section 4.5. We hope the use of our evaluation framework using the flexible graph generator will reduce the burden of comparing existing GNNs and developing/evaluating new algorithms.

Chapter 2

Class Structure-aware Attributed Graph Clustering by Bridging Attribute and Topology Spaces

2.1 Introduction

As we discussed in Chapter 1, graphs in the real world usually have attributes on nodes. Actually, the graph databases support attributed graphs or property graphs [36, 101]. However, most graph clustering techniques [57, 87, 115] do not leverage the attributes of nodes since their design is limited to simple graphs without having attributes. Therefore, these techniques cannot extract precise clusters without leveraging the attributes. There are emerging researches that tackle the clustering problem for attributed graphs [7, 50, 91, 116, 133] and the representation learning problem[1] for attributed graphs, such as ANRL [129] and AANE [49]. Despite the considerable improvements made by those methods, they have not effectively leveraged the virtue of attributed graphs. There are two fundamental aspects of the attributed graphs we should consider. First, the topology and the attributes of real-world graphs have a complicated relationship with each other, i.e., the class structure, because they are obtained from different viewpoints in

[1]We will discuss the relationship between representation learning and clustering in Section 2.4.

real-world but these viewpoints are correlated. Therefore, we need to balance the effects of the topology and the attributes for each cluster independently: some clusters are formed due primarily to the graph topology while some others are formed due primarily to the attributes. Note that we consider a more complicated relationship between the topology and attributes than the relationship that existing methods assume since they cannot control the effect of the topology to the attributes for each cluster. Second, typical graphs consist only of partially observed positive edges, i.e., edges that exist in the real world but are not observed. This is because real-world graphs follow the open world assumption: "absence of information is interpreted as unknown information, not as negative" [58]. For example, a social graph may not reflect precisely the social connections in the real world: we can only observe positive connections between people such as "likes" and "friendships", but cannot observe negative ones [46].

We take the above two aspects into account and propose NAGC, a New Attributed Graph Clustering method by bridging the attribute space and topology space and by taking the effect of partially observed positive edges. To achieve high clustering quality, 1) NAGC learns a projection function between the topology space and the attribute space so as to capture their complicated relationship. The projection function consists of a rescale function and a transfer matrix that balances the effect from the attribute space to the topology space for each cluster independently, and 2) NAGC leverages PU (positive-unlabeled) learning [31, 46, 74] to take the effect of partially observed positive edges into the cluster assignment. To the best of our knowledge, our method is the first method that learns the representation of attributed graphs 1) by capturing the complex relationship between the topology and the attributes and 2) by applying the PU learning to the missing positive edges. Our method can precisely capture clustering results by revealing the relationship between the topology and the attributes in real-world graphs.

We extensively performed experiments for various clustering methods and representation learning methods over various real datasets with ground truth. We also performed a micro benchmark to validate the effectiveness of learning projection function and PU learning to the clustering quality. With these experiments, we confirm that our method performs higher than or comparable to the existing methods in terms of the clustering quality. In addition, we confirm that NAGC actually captures complicated relationships between attribute space and topology space by visualizing the transfer matrix. We

also confirm that our method is stable against the hyperparameter selection.

The rest of this chapter is organized as follows. We introduce fundamental techniques for our method, Non-negative Matrix Factorization, Symmetric Non-negative Matrix Factorization, and Biased Matrix Completion in Section 2.2. We propose our method in Section 2.3. Section 2.4 addresses the details of the related work. Section 2.5 gives the purpose and results of the evaluations. Finally, we conclude this chapter at Section 2.6.

2.2 Preliminaries

Notation: We denote a matrix and its i-th row vector as upper boldface $\boldsymbol{X}$ and under boldface $\boldsymbol{x}_i$. The set of non-negative real numbers is $\mathbb{R}_+$. We denote an *attributed graph* $G = (\boldsymbol{A}, \boldsymbol{X})$, where $\boldsymbol{A} \in \mathbb{R}_+^{n \times n}$ is an adjacency matrix and $\boldsymbol{X} \in \mathbb{R}_+^{n \times d}$ is an attribute matrix. For simple descriptions, we additionally denote the topology of a graph $G_{topology} = (V, E)$ comprising a set of nodes $V = \{1, 2, \ldots, n\}$ and edges $E = \{(i, j)\} \subseteq [n] \times [n]$. This corresponds to an adjacency matrix $\boldsymbol{A}$. $||\cdot||_{\mathcal{F}}$ and $||\cdot||_*$ are Frobenius norm and the nuclear norm, respectively. We use $\odot$ and $\oslash$ to denote element-wise multiplication and element-wise division, respectively.

2.2.1 Non-negative Matrix Factorization

Given the number of clusters for topology and attributes, $k_1, k_2 \ll \min\{d, n\}$, respectively, we suppose a cluster assignment matrix $\boldsymbol{U} \in \mathbb{R}_+^{n \times k_1}$ and an attribute factor matrix $\boldsymbol{V} \in \mathbb{R}_+^{d \times k_2}$. Let us denote a transfer matrix $\boldsymbol{H} \in \mathbb{R}_+^{k_1 \times k_2}$ that represents the relationship between topology and attributes. Non-negative Matrix Tri-Factorization (NMTF) [28], which is a novel extension of Non-negative Matrix Factorization (NMF) [69], estimates local optimal parameters $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{H}$ by minimizing a non-convex loss:

$$\min_{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H} \geq 0} ||\boldsymbol{X} - \boldsymbol{U}\boldsymbol{H}\boldsymbol{V}^\top||_{\mathcal{F}}^2. \tag{2.1}$$

NMF is treated as a special case of NMTF where $\boldsymbol{H}$ is set to an identity matrix. Compared with the original NMF, which estimates $\boldsymbol{U}$, $\boldsymbol{V}$, NMTF generates more precise model by introducing transfer matrix $\boldsymbol{H}$. However, NMTF is limited to consider only linear relationships between topology and attributes.

2.2.2 Symmetric Non-negative Matrix Factorization

The goal of graph clustering is to find a partition of nodes in a graph where the similarity between nodes is high within the same cluster and low across different clusters. Kuang et al. proposed Symmetric Non-negative Matrix Factorization (SNMF) [63, 64], and showed an interesting relationship among SNMF and graph clustering methods [88]. SNMF estimates a cluster assignment matrix $\boldsymbol{U}$ by minimizing a non-convex loss function that uses an adjacency matrix $\boldsymbol{A}$ as input:

$$\min_{\boldsymbol{U} \geq 0} ||\boldsymbol{A} - \boldsymbol{U}\boldsymbol{U}^\top||_{\mathcal{F}}^2. \tag{2.2}$$

Thanks to the non-negative constraint, we can obtain a clustering result by assigning i-th node to the k_1'-th cluster that has the largest value in $\boldsymbol{u}_i$, that means $k_1' = \text{argmax}_l\{u_{i,l} \mid l = (1, \ldots, k)\}$. We don't need to apply additional clustering techniques such as k-means to the node vectors.

2.2.3 Biased Matrix Completion

Hsieh et al. [46] considered a matrix completion problem when only a subset of positive relationships is observed, such as recommender systems and social networks where only "likes" or "friendships" are observed. The problem is an instance of PU learning, i.e. learning from only positive and unlabeled examples that has been studied in the classification problems. They introduced the ρ-weighted loss for a bipartite graph $G' = (V', E')$ comprising a set of nodes $V' = \{\{1, 2, \ldots, n\}, \{1, 2, \ldots, m\}\}$ and edges $E =' \{(i, j)\} \subseteq [n] \times [m]$:

$$\ell_\rho(z_{i,j}) = \rho 1_{(i,j)\in E'}(z_{i,j} - 1)^2 + (1 - \rho)1_{(i,j)\notin E'} z_{i,j}^2, \tag{2.3}$$

where $\rho = [0, 1]$, $1_{(i,j)\in E'}(\cdot)$, and $1_{(i,j)\notin E'}(\cdot)$ are a bias weight, an indicator function for positive edges, and an indicator function for unlabeled edges, respectively. This loss can change a weight for reconstruction errors among positive and unlabeled edges. When we set $\rho = 0.5$, it treats the positive and unlabeled entities equally. With this loss, they proposed a biased matrix completion as:

$$\min_{\boldsymbol{Z}:||\boldsymbol{Z}||_* \leq \lambda} \sum_{(i,j)\in E'} \rho(z_{i,j} - 1)^2 + \sum_{(i,j)\notin E'} (1 - \rho) z_{i,j}^2. \tag{2.4}$$

where $\lambda \geq 0$ is a hyperparameter.

Table 2.1: Definition of main symbols.

Variable	Explanation
$\boldsymbol{A} \in \mathbb{R}_+^{n \times n}$	adjacency matrix
$\boldsymbol{X} \in \mathbb{R}_+^{n \times m}$	attribute matrix
$\boldsymbol{U} \in \mathbb{R}_+^{n \times k_1}$	topology cluster assignment matrix
$\boldsymbol{V} \in \mathbb{R}_+^{m \times k_2}$	attribute factor matrix
$\boldsymbol{H} \in \mathbb{R}_+^{k_1 \times k_2}$	cluster assignment transfer matrix
$\boldsymbol{W} \in \mathbb{R}_+^{n \times n}$	mask matrix of $\boldsymbol{A}$
$k_1 \in \mathbb{N}$	number of clusters for topology
$k_2 \in \mathbb{N}$	number of clusters for attributes
$\lambda \geq 0$	balancing parameter between the topology and the attributes
$\rho = [0, 1]$	bias weight for $\boldsymbol{A}$
$t \in \mathbb{N}$	number of iterations

2.3 NAGC: New Attribute Graph Clustering

As we mentioned in Section 2.1, we need to consider two fundamental aspects of the attributed graphs: 1) the topology and the attributes of real-world graphs have a complicated relationship and 2) typical graphs usually have a subset of positive edges implying that there are missing positive edges. The novelty of NAGC is three-hold:

- We jointly decompose the topology (adjacency) matrix and the attribute matrix into factor matrices to represent the node features in the topology space and the attribute space. We employ SNMF and NMTF to decompose the adjacency matrix and attribute matrix, respectively.

- NAGC learns a projection function between the topology space and the attribute space. The projection function consists of a rescale function and a cluster assignment transfer matrix. The rescale function bridges the scale gap between the two spaces. The transfer matrix bridges the two spaces by balancing the effect from the attribute space to the topology space for each cluster independently. This transfer matrix

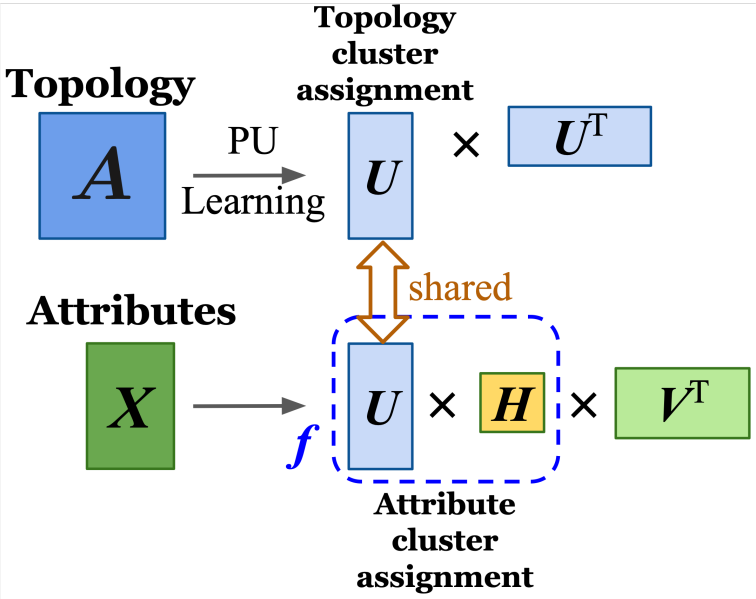


Figure 2.1: Illustration of NAGC. $\boldsymbol{A}$ and $\boldsymbol{X}$ are an adjacency matrix and an attributed matrix, respectively. $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{H}$ denote a topology cluster assignment, an attribute factor, and a cluster assignment transfer matrices, respectively. f is a rescale function.

increases the expressive power of our model as deep learning models typically increase their weight matrices.

- We leverages PU learning to take the effect of partially observed positive edges into the cluster assignment. That is, we put a larger bias to partially observed positive edges than unlabeled edges.

Table 2.1 lists the main symbols and their definitions.

2.3.1 NAGC Model

We formalize our method as a minimization problem of a non-convex loss as follows:

$$\min_{\boldsymbol{U},\boldsymbol{V},\boldsymbol{H}\geq 0} \mathcal{L}_{\rho}(\boldsymbol{A}-\boldsymbol{U}\boldsymbol{U}^{\top})+\frac{\lambda}{2}||\boldsymbol{X}-f(\boldsymbol{U}\boldsymbol{H})\boldsymbol{V}^{\top}||_{\mathcal{F}}^{2}. \tag{2.5}$$

Figure 2.1 depicts the design of NAGC. The adjacency matrix $\boldsymbol{A}$ is decomposed into $\boldsymbol{U}\boldsymbol{U}^\top$, so $\boldsymbol{U}$ represents the matrix of the topology cluster assignment. In contrast, the attribute matrix $\boldsymbol{X}$ is decomposed into $f(\boldsymbol{U}\boldsymbol{H})\boldsymbol{V}^\top$, so $f(\boldsymbol{U}\boldsymbol{H})$ represents the matrix of the attribute cluster assignment. By transforming the matrix of the topology cluster assignment ($\boldsymbol{U}$) to the matrix of the attribute cluster assignment ($f(\boldsymbol{U}\boldsymbol{H})$) with the rescale function f and transfer matrix $\boldsymbol{H}$, our method enables to capture the complex relationship between the topology cluster assignment and the attribute cluster assignment. In particular, $\boldsymbol{H}$ bridges the two assignments by balancing the effect from the attribute space to the topology space for each cluster independently. In addition, λ is a hyperparameter that balances globally the effects between the topology and the attribute for all clusters. f denotes an element-wise rescale function and we use the sigmoid function as f: $f(x) = \frac{1}{1+e^{-x}}$. There are two reasons we adopt the sigmoid function. First, the sigmoid function is differentiable so the parameter update rules of the existing method can be used with minor modifications. Second, the sigmoid function is one of the simplest functions for rescale function.

This choice can be generalized to any non-linear function. We use $\mathcal{L}_\rho(\boldsymbol{Z})$ to denote an error of the adjacency matrix $\boldsymbol{A}$ with ρ-weighted loss.

$$\mathcal{L}_\rho(\boldsymbol{Z}) = \sum_{(i,j)\in E} \rho(z_{i,j} - 1)^2 + (1-\rho) \sum_{(i,j)\notin E} z_{i,j}^2. \tag{2.6}$$

Note that, the number of clusters k_2 for attribution is not necessary the same as the number of clusters k_1 for topology, since we suppose that the cluster structure embedded in attributes differs from that in topology. NAGC can be seen as a generalized SNMF because NAGC simulates SNMF by setting $\lambda = 0$ and $\rho = 0.5$: no effects from attributes and partially observed positive edges.

2.3.2 Optimization

Since our loss is non-convex for $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{H}$, we derive a parameter estimation procedure that alternately updates each parameter by utilizing the method of Lagrange multipliers [28]. Following the standard theory of constrained optimization, we introduce Lagrangian multipliers $\boldsymbol{\alpha} \in \mathbb{R}^{n\times k_1}$, $\boldsymbol{\beta} \in \mathbb{R}^{m\times k_2}$, and $\boldsymbol{\gamma} \in \mathbb{R}^{k_1\times k_2}$ for the non-negative constraints $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H} \geq 0$.

We define the Lagrangian function of our proposed method as:

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H}; \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) \\
&= \mathcal{L}_\rho(\boldsymbol{A} - \boldsymbol{U}\boldsymbol{U}^\top) + \frac{\lambda}{2}||\boldsymbol{X} - f(\boldsymbol{U}\boldsymbol{H})\boldsymbol{V}^\top||_{\mathcal{F}}^2 \\
&\quad + \mathrm{Tr}(\boldsymbol{\alpha}^\top \boldsymbol{U}) + \mathrm{Tr}(\boldsymbol{\beta}^\top \boldsymbol{V}) + \mathrm{Tr}(\boldsymbol{\gamma}^\top \boldsymbol{H}). \qquad (2.7)
\end{aligned}$$

For each parameter, we derive partial differences of $\mathcal{L}$.

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}} &= -2\rho \boldsymbol{A}\boldsymbol{U} - \lambda\{(\boldsymbol{X}\boldsymbol{V}) \odot f'(\boldsymbol{U}\boldsymbol{H})\}\boldsymbol{H}^\top \\
&\quad + 2\rho(\boldsymbol{U}\boldsymbol{U}^\top \odot \boldsymbol{W})\boldsymbol{U} + 2(1-\rho)(\boldsymbol{U}\boldsymbol{U}^\top \odot \boldsymbol{W}')\boldsymbol{U} \\
&\quad + \lambda[\{f(\boldsymbol{U}\boldsymbol{H})\boldsymbol{V}^\top \boldsymbol{V}\} \odot f'(\boldsymbol{U}\boldsymbol{H})]\boldsymbol{H}^\top + \boldsymbol{\alpha}. \qquad (2.8) \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}} &= -\lambda \boldsymbol{X}^\top f(\boldsymbol{U}\boldsymbol{H}) + \lambda \boldsymbol{V} f(\boldsymbol{U}\boldsymbol{H})^\top f(\boldsymbol{U}\boldsymbol{H}) + \boldsymbol{\beta}. \qquad (2.9) \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{H}} &= -\lambda \boldsymbol{U}^\top \{f'(\boldsymbol{U}\boldsymbol{H}) \odot (\boldsymbol{X}\boldsymbol{V})\} \\
&\quad + \lambda \boldsymbol{U}^\top \{f'(\boldsymbol{U}\boldsymbol{H}) \odot f(\boldsymbol{U}\boldsymbol{H})\}\boldsymbol{V}^\top \boldsymbol{V} + \boldsymbol{\gamma}. \qquad (2.10)
\end{aligned}$$

$\boldsymbol{W} \in \mathbb{R}_+^{n \times n}$ is a mask matrix whose elements are set to $w_{i,j} = 1$ if $s_{i,j} \neq 0$ or $w_{i,j} = 0$ otherwise, and $\boldsymbol{W}' = 1 - \boldsymbol{W}$. The KKT complementarity conditions are: $\boldsymbol{\alpha} \odot \boldsymbol{U} = 0, \boldsymbol{\beta} \odot \boldsymbol{V} = 0, \boldsymbol{\gamma} \odot \boldsymbol{H} = 0$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}} = 0$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}} = 0$, and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{H}} = 0$. By satisfying these conditions, we can derive multiplicative update rules.

$$\begin{aligned}
\boldsymbol{U} &\leftarrow \boldsymbol{U} \odot [2\rho \boldsymbol{A}\boldsymbol{U} + \lambda\{(\boldsymbol{X}\boldsymbol{V}) \odot f'(\boldsymbol{U}\boldsymbol{H})\}\boldsymbol{H}^\top] \oslash \\
&\quad [2\rho(\boldsymbol{U}\boldsymbol{U}^\top \odot \boldsymbol{W})\boldsymbol{U} + 2(1-\rho)(\boldsymbol{U}\boldsymbol{U}^\top \odot \boldsymbol{W}')\boldsymbol{U} \\
&\quad + \lambda\{(f(\boldsymbol{U}\boldsymbol{H})\boldsymbol{V}^\top \boldsymbol{V}) \odot f'(\boldsymbol{U}\boldsymbol{H})\}\boldsymbol{H}^\top]. \qquad (2.11) \\
\boldsymbol{V} &\leftarrow \boldsymbol{V} \odot \{\boldsymbol{X}^\top f(\boldsymbol{U}\boldsymbol{H})\} \oslash \{\boldsymbol{V} f(\boldsymbol{U}\boldsymbol{H})^\top f(\boldsymbol{U}\boldsymbol{H})\}. \qquad (2.12) \\
\boldsymbol{H} &\leftarrow \boldsymbol{H} \odot [\boldsymbol{U}^\top \{f'(\boldsymbol{U}\boldsymbol{H}) \odot (\boldsymbol{X}\boldsymbol{V})\}] \\
&\quad \oslash [\boldsymbol{U}^T \{f'(\boldsymbol{U}\boldsymbol{H}) \odot f(\boldsymbol{U}\boldsymbol{H})\}\boldsymbol{V}^\top \boldsymbol{V}]. \qquad (2.13)
\end{aligned}$$

Our loss is convex with respect to $\boldsymbol{V}$ and $\boldsymbol{H}$, however, as mentioned in [63], the loss is a fourth-order non-convex function with respect to $\boldsymbol{U}$. That means, it is difficult to guarantee the monotonic convergence of our parameter estimation method; thus we expect a good convergence property that every limit point is a stationary point.

Algorithm 1 shows the algorithm for our method. Since non-convex minimization problems have multiple local minima, we apply k-means to the

Algorithm 1: NAGC-U algorithm

Input: $\boldsymbol{A}, \boldsymbol{X}, k_1, k_2, \lambda, t$
Output: clustering result C

Preprocess: $\boldsymbol{A}, \boldsymbol{X}$
Initialize: $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{H}$
while $t' < t$ **do**
  # alternately update parameters
  $\boldsymbol{U}^{(t'+1)} \leftarrow$ update $(\boldsymbol{U}^{(t')})$ by Eq. (2.11)
  $\boldsymbol{V}^{(t'+1)} \leftarrow$ update $(\boldsymbol{V}^{(t')})$ by Eq. (2.12)
  $\boldsymbol{H}^{(t'+1)} \leftarrow$ update $(\boldsymbol{H}^{(t')})$ by Eq. (2.13)
while $n' < n$ **do**
  # assign each node to the clusters
  $c_{n'} \leftarrow \text{argmax}_l\{u_{n',l} \mid l = (1, \ldots, k_1)\}$

attribute matrix $\boldsymbol{X}$ and use the result to initialize $\boldsymbol{U}$ and $\boldsymbol{V}$. $\boldsymbol{H}$ is initialized by random values in the same way as the standard NMTF. There are two variations of our method, NAGC-U and NAGC-UH. They obtain clusters based on the topology cluster assignment $\boldsymbol{U}$ and attribute cluster assignment $\boldsymbol{UH}$, respectively.

2.3.3 Computational Complexity

Let t be the number of iterations in the matrix decomposition. Since the cost for SNMF is $O(n^2kt)$ [63, 64], the cost for updating rules of NAGC is equal to $O((n^2 + dn)kt)$ where $k = \max(k_1, k_2)$ and $k \ll n$ in general. For example, the operation of NAGC finishes in 3 seconds for WebKB which is a small size dataset, in 60 seconds for Citeseer which is a middle size dataset, and in 250 seconds for Flickr which is a relatively large size dataset[2].

Limitations. Since NAGC depends quadratically on the number of nodes, it does not scale to large-scale graphs, e.g., graphs with million nodes. While we address the effectiveness of a graph clustering method in this chapter, we will address the scalability and efficiency problems[3] in the next chapter.

[2]The experiments are implemented on Python3.

[3]Though we focus on graph neural networks in the next chapter, they can be used as clustering methods by transforming their loss functions into unsupervised fashions.

2.4 Related Work

There are many clustering methods for attributed graphs [7, 50, 91, 116, 133] and representation learning techniques for attributed graphs [49, 119, 129]. Most of the representation learning for attributed graphs are influenced by node embedding techniques [18, 41, 93, 104, 122].

2.4.1 Attributed Graph Clustering

Matrix decomposition-based. SNMF is recently extended to consider both the topology and the attributes for discovering clusters of data entities. DANMF [122] is a deep autoencoder-like nonnegative matrix decomposition method, which is extended from SNMF. It learns hierarchical mappings between the original network and the final community assignment based on a deep autoencoder-like architecture. CDE [71] shares the same design with ours in that it decomposes topology matrix and attribute matrix by using NMF, but it is also orthogonal to ours in that 1) it newly introduces a community structure embedding matrix (distance matrix from node to node in cluster space) used as a topology matrix, whereas 2) our approach learns a projection function between the topology space and the attribute space and also leverages PU learning. TLSC [128] is based on generative models and it does usually not perform better than NMF-based approaches. Indeed, the experiments reported in [71] show that CDE performs higher than TLSC in terms of the NMI measure. JWNMF [50] factorizes both the topology and the attribute matrices at the same time, however, the clustering quality is not high since it does not use the transfer matrix between the topology space and attribute space: the transfer matrix effectively balances the effect between those two spaces.

Distance-based. SA-Cluster [132] and its efficient version Inc-Cluster [133] are attributed graph clustering methods expanded from distance-based graph clustering. The key idea is to embed node attributes as new nodes into the graph. A unified distance for the augmented graph is defined by the random walk process, and the graph is partitioned by k-medoids. It is hard to apply these methods to large graphs since the augmented steps increase the size of the graph considerably.

Bayesian-based. BAGC/GBAGC [116, 117] learns a posterior distribution over the model parameters. This method assumes that the nodes in the same cluster should have a common multinomial distribution for each

node attribute and a Bernoulli distribution for node connections. The attributed graph clustering problem can be formulated as a probabilistic generative model. PAICAN [18] performs anomaly detection and clustering on the attributed graph at the same time. PAICAN explicitly models partial anomalies by generalizing the ideas of Degree Corrected Stochastic Block Models [54] and Bernoulli Mixture Models.

2.4.2 Representation Learning

The representation learning generates node embeddings[4] for attributed graphs, which is a way of representing nodes as vectors, similarly to word embedding techniques such as word2vec [84]. The embeddings can be used for various tasks such as clustering, link prediction, and classification.

ANRL [129] combines a neighbor enhancement autoencoder and an attribute-aware skip-gram model for learning node features that preserve the attributes and the network structure. It controls the topology effect by tuning parameters for the contribution of the neighbor enhancement autoencoder and for the window size on random walk. AANE [49] uses a hyperparameter λ that balances globally the effects between the topology and the attribute for all clusters. Notice that ANRL and AANE use hyperparameters to control the effect of the topology to the attributes for all clusters, however, they cannot control it for each cluster independently. This is because they simply propagate the attributes by random walk on graph. TADW [119] employs NMTF to decompose the topology matrix into the product of two factor matrices and text feature matrix. TADW is not robust to the text feature since factors are not extracted from the text feature.

Graph Convolutional Networks [61], that is a semi-supervised learning method for a graph, has obtained considerable attention from machine learning and data mining fields due to its high performance in classifying graph nodes. However, this approach needs a subset of true cluster labels on nodes, and thus its goal is different from that of the attributed graph clustering.

2.5 Experiments

The purpose of our experiments is to answer the following questions:

[4]Representation learning techniques are closely related to clustering techniques since graph clustering can be regarded as feature clustering on node embedding space.

Q1 Does NAGC perform higher than former methods? (Section 2.5.4)

Q2 Does NAGC capture the complicated relationship between the topology and the attributes? (Section 2.5.5)

Q3 How largely the parameters affect the performance? (Section 2.5.6)

In detail, the first purpose of the experiments is to evaluate the clustering quality of NAGC[5] compared with various methods: representation learning methods for attributed graphs (AANE [49], ANRL [129]), an attributed graph clustering method (JWNMF [50]), graph clustering methods without using attributes (METIS [57], DANMF [122]), and a typical attribute-based clustering method (k-means). We used publicly available codes for those methods. As for the representation learning methods, we learned the node representation by using the same setting used in each paper. Then, we obtain clustering results by applying k-means to the learned representation by taking the same approach used in [41, 122]. We also evaluate a simple graph clustering method without using attributes, METIS [55], and an attribute-based clustering method, k-means, so that how much only the topology or attributes of the graphs contribute to the clustering quality. We use two variations of our method, NAGC-U and NAGC-UH, based on the topology cluster assignment and the attribute cluster assignment, respectively. We perform five restarts for each method and report the average of the results for all the above experiments.

The second purpose is to evaluate how effectively the transfer matrix bridges the two spaces to capture their complicated relationship, because the transfer matrix is designed to balance the effect from the attribute space to the topology space for each cluster independently.

The third purpose of our experiments is to investigate the details of the quality improvement achieved by our method: we evaluate the effectiveness of PU learning and the effect of the hyperparameters.

2.5.1 Datasets

We choose seven real-world datasets with ground truth in our experiments. They cover wide variety of graph types and sizes. They are used in the related papers of the attributed graph clustering. The graph types of our

[5]The source code of NAGC is available at https://github.com/seijimaekawa/NAGC.

datasets (web graph, blogs, Wikipedia, citation networks, social network) cover more than half of the categories used in SNAP[6] graph data archive. Also, the graph sizes are from small to relatively large (the number of nodes from 877 to 7,564 and the number of edges from 1480 to 239,365).

- WebKB[7] is a web graph of four universities: nodes indicate web pages, edges indicate hyperlinks between pages, the label for a node indicates the owner university of the page, and the attributes of a node represent the words that appear on the page.
- Polblog[8] is a network of hyperlinks between blogs on US politics: nodes indicate blog posts, edges indicate hyperlinks between blogs, the label of a node indicates whether the blog is liberal or conservative, and the attributes of a node represent the sources of the blogs.
- Wiki is a document network and the link among different nodes is the hyperlink in a web page. The label of a node indicates the category of the node. The attributes represent the TFIDF matrix of this dataset.
- Citeseer and Cora (see also footnote7 for detail) are citation networks. The label of a node corresponds to a research field of the paper. The attributes of a node consist of the words appeared in the paper.
- BlogCatalog is a blogger community network, where users interact with each other. The labels of nodes indicate the topics of bloggers, to which the bloggers register their blogs. The attributes of a node represent the keywords of their blogs.
- Flickr is an online community in which people can share photos and follow each other. The labels of nodes indicate communities to which the nodes belong. We use the tags attached on each image as the attribute information.

Table 2.2 summarizes the statistics of the datasets[9]. We also include the modularity [87] and the average entropy for each of the true cluster assignment: the modularity and the entropy represent the topological aspect and

[6]Stanford Large Network Dataset Collection: https://snap.stanford.edu/data/index.html

[7]http://linqs.cs.umd.edu/projects/projects/lbc/index.html

[8]http://www-personal.umich.edu/~mejn/netdata/

[9]As we mentioned in Section 1.1, we focus on homogeneous graphs, i.e., graphs we use have no edge types.

Table 2.2: The statistics of the datasets. Mod. and Ent. indicate the modularity and the average entropy, respectively.

Dataset	Node	Edge	Attribute	Label	Mod.	Ent.
WebKB	877	1480	1703	4	0.739	0.152
Polblog	1490	16630	7	2	0.405	0.379
Wiki	2405	12761	4973	17	0.524	0.320
Cora	2708	5278	1433	7	0.640	0.054
Citeseer	3312	4660	3703	6	0.544	0.039
BlogCatalog	5196	171743	8189	6	0.224	0.036
Flickr	7564	239365	12047	9	0.121	0.012

attribute aspect, respectively. Intuitively, higher modularity indicates there are dense connections in the same cluster but sparse connections between different clusters. Lower average entropy indicates there are similar attribute values in the same cluster but dissimilar attribute values between different clusters. Average entropy is defined as:

$$Average\_entropy = \sum_{i=1}^{m}\sum_{j=1}^{k}\frac{|C_j|}{nm}entropy(a_i, C_j) \tag{2.14}$$

where $entropy(a_i, C_j)$ is the information entropy of attribute a_i in cluster C_j. The values with respect to the modularity and the average entropy fall within the range of $[-1, 1]$ and the range of $[0, 1]$, respectively.

2.5.2 Measurements

The modularity and entropy are not suitable measurements for the clustering evaluation of attributed graphs, because the resulting clusters should take into account both aspects of the topology and attributes. We utilize two measures, Adjusted Rand Index (ARI) [51] and Adjusted Mutual Information (AMI) [107]. ARI is the corrected-for-chance version of the Rand index [51]. Such a correction for chance establishes a baseline by using the expected similarity of all pair-wise comparisons between clusterings specified by a random model. AMI corrects the effect of agreement solely due to chance between clusterings. Note that it is an adjusted version of Normalized

Mutual Information (NMI), similar to the way that ARI corrects the Rand index. They are typical measurements used for assessing the clustering quality with ground truth labels[10]. They are adjusted in a sense that random cluster assignments make ARI and AMI scores close to zero. On the other hand, non-adjusted measures such as NMI have a dependency between the number of clusters and the number of samples used to compute the measure. Therefore, the adjusted measures are more preferable for cluster evaluation.

2.5.3 Parameter Settings

We searched for optimum parameters, λ, k_2, and ρ for each dataset and used them in our experiments. λ is chosen from the set $\{10^{-10}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 1, 10, 100, 1000\}$ by following the settings used in [50]. The model does not work well when $\lambda > 100$.

Let k be the number of true clusters for each dataset. k_2 is chosen from the set $\{k, 5, 7, 10, 15, 20\}$ for NAGC-U[11] so that we can learn the model more precisely than when we use k. ρ is chosen from the set $\{0.5, 0.55, 0.75, 0.95, 0.995\}$. To mitigate the different scales between $\boldsymbol{A}$ and $\boldsymbol{X}$, we normalize $\boldsymbol{A}$ by multiplying each element of $\boldsymbol{A}$ with $\frac{|\boldsymbol{X}|}{|\boldsymbol{A}|}$. The iterate computation of our method converges very fast (usually in 100 iterations) so the number of the iterations t is fixed at 100 in all the experiments.

2.5.4 Clustering Quality

Table 2.3 shows the results of evaluating the clustering quality. NAGC-U is obtained from the topology cluster assignment and NAGC-UH is obtained from the attribute cluster assignment. The last column (Avg.) indicates the average for all datasets. NAGC achieves the best performance not only in the average results, but also in six datasets (out of seven) in ARI measurement and three datasets in AMI measurement. The benefit of NAGC is that it balances and combines the effects of both the topology and the attributes, as we can see that NAGC is always better than METIS and k-means. Moreover, NAGC generally works well regardless of the entropy of the datasets (see Table 2.2). In particular, NAGC performs better than other methods even

[10]We choose AMI since NMI is not adjusted for chance. Note that the chance rates of ARI and AMI are 0.

[11]We do the same by replacing k_2 with k_1 for NAGC-UH.

Table 2.3: Clustering performance of different datasets. The boldface font represents the best performance for each dataset.

Dataset		WebKB	Polblog	Wiki	Cora	Citeseer	BlogCatalog	Flickr	Avg.
ARI	NAGC-U	**0.992**	**0.646**	0.315	0.336	0.269	0.200	0.136	**0.413**
	NAGC-UH	0.802	0.100	**0.367**	0.360	**0.303**	**0.259**	**0.141**	0.333
	AANE	0.974	0.003	0.137	0.221	0.184	0.174	0.086	0.254
	ANRL	0.990	0.000	0.212	**0.439**	0.296	0.252	0.053	0.320
	JWNMF	0.908	0.513	0.132	0.309	0.077	0.149	0.133	0.317
	DANMF	0.850	0.556	0.174	0.249	0.084	0.129	0.067	0.301
	METIS	0.909	0.575	0.187	0.246	0.155	0.143	0.069	0.326
	k-means	0.274	0.000	0.040	0.062	0.169	0.000	0.000	0.078
AMI	NAGC-U	**0.987**	**0.547**	0.440	0.374	0.266	0.260	0.199	**0.439**
	NAGC-UH	0.742	0.078	**0.459**	0.404	0.290	0.310	0.168	0.351
	AANE	0.962	0.006	0.447	0.336	0.216	0.289	0.172	0.347
	ANRL	0.984	0.000	0.378	**0.491**	**0.349**	**0.337**	0.095	0.376
	JWNMF	0.899	0.442	0.260	0.232	0.087	0.215	**0.202**	0.334
	DANMF	0.853	0.484	0.292	0.334	0.133	0.194	0.105	0.342
	METIS	0.889	0.471	0.299	0.336	0.173	0.186	0.105	0.351
	k-means	0.292	0.000	0.174	0.117	0.207	0.005	0.001	0.114

when the entropy is large (WebKB, Polblog, Wiki). NAGC also performs best when the attributes do not effectively contribute to the clustering result, such as when k-means works poorly (Flickr)[12]. This behavior implies that NAGC selectively chooses the effect from the attribute space to the topology space for each cluster independently.

In contrast, ANRL generally works well when the entropy is small (Cora, Citeseer, BlogCatalog) but not otherwise[13]. ANRL learns the representation equally from all the attributes, so it does not control the effect of each attribute cluster independently to the topology cluster assignment: ANRL is even worse than METIS for Polblog and Flickr.

To investigate more on the difficulty of the attributed graph clustering, we show that the topology and the attributes of real-world graphs have different cluster assignments. Table 2.4 gives the modularity and the average entropy

[12]k-means decides the centroids of clusters by treating all nodes equally so it does not work well when most elements of the attribute matrix are zero. Actually, 99% of the nodes are assigned to a single cluster in Flickr.

[13]The average entropy tends to be low when most elements of the attribute matrix are zero regardless of the correlation between the attributes and the true label. Considering the summation of all elements of the attribute matrix divided by the numbers of nodes and attributes, Flickr has the smallest value, 0.0280. It is much smaller than the average value of all datasets, 0.128. For this reason, Flickr has low entropy.

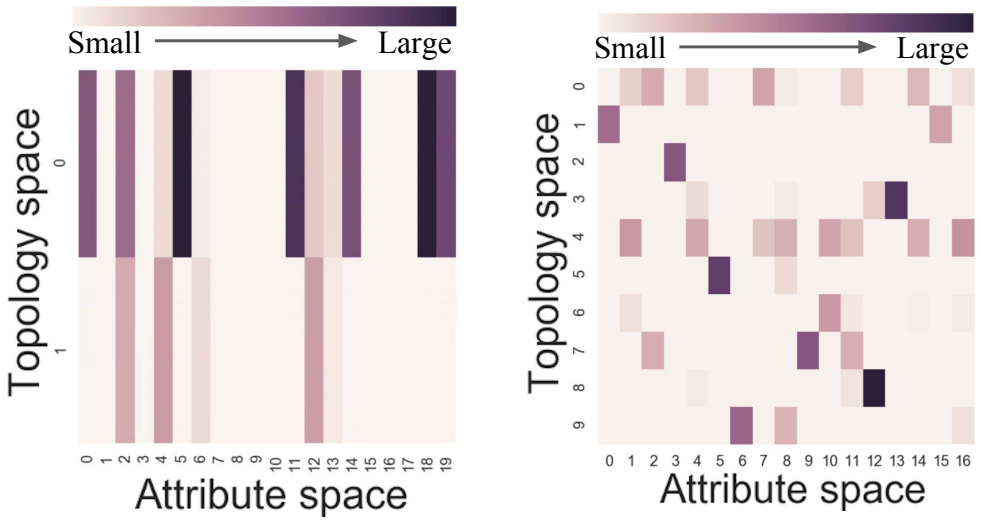
Table 2.4: Modularity and average entropy for WebKB dataset.

	Modularity	Entropy	ARI
NAGC-U	0.737	0.152	**0.992**
AANE	0.731	0.152	0.974
ANRL	0.737	0.152	0.990
JWNMF	**0.741**	0.153	0.908
DANMF	0.718	0.153	0.850
METIS	**0.741**	0.153	0.909
k-means	0.252	**0.146**	0.274

for the clustering result of WebKB. Our method achieves the highest ARI but does not achieve either the best modularity or the best average entropy. This result implies that we should not optimize the model only to either the topology or the attributes, but balance the effects between the topology and the attributes.

2.5.5 Bridging Topology and Attribute Spaces

One of the most important contributions is how effectively NAGC captures the complex relationship between the topology space and the attribute space. Fig.2.2 depicts the heatmaps of the transfer matrix $\boldsymbol{H}$ for Polblog and Wiki since $\boldsymbol{H}$ represents the relationship between the topology clusters and attribute clusters. Note that the size of $\boldsymbol{H}$ is $\mathbb{R}^{k_1 \times k_2}$ and the X and Y axes depict attribute clusters (k_2 dimensions) and topology clusters (k_1 dimensions), respectively. We choose Polblog and Wiki datasets in which NAGC-U and NAGC-UH achieve the best results for both ARI and AMI. The darker elements indicate there is a larger effect from the attribute cluster to the topology cluster, i.e., the topology and attribute clusters are strongly correlated if the elements are dark. . In detail, in Fig.2.2 (a), most attribute clusters (1,3,7-10,15-17) are colored in light color, this indicates that there is almost no effect from those attribute clusters to topology clusters. In other words, those attribute clusters do not have any correlations with topology clusters. Considering a web graph with word information as an example, there is no correlation between general words (such as “abstract” or “introduction” for



(a) Polblog (k_1=2, k_2=20)

(b) Wiki (k_1=10, k_2=17)

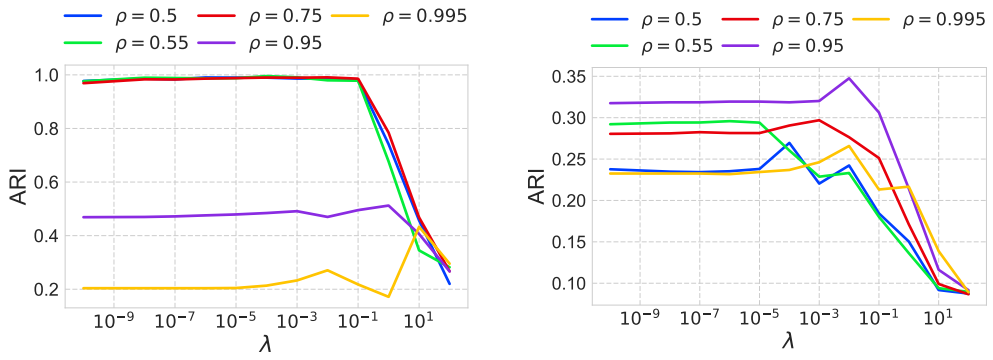
Figure 2.2: Heatmaps of the transfer matrix $\boldsymbol{H}$. The darker elements indicate there is larger effect from attribute cluster to topology cluster.

academic papers) and topology clusters. In contrast, the attribute cluster of 0,5,11,18,19 indicate theses attribute clusters effect mostly to topology cluster 0. In Fig.2.2 (b), the dark elements in the matrix, such as (5,5), (12,8), (13,3), show clear effect from an attribute cluster to a topology cluster. Also, we can observe that the topology cluster of 4 receives almost equal effects from multiple attribute cluster of 1,4,7,8,10,11,14,16.

These results validate that there is actually a complex relationship between topology space and attribute space and justify our motivation: we should learn a projection function between the topology space and the attribute space.

2.5.6 Hyperparameter Analysis

We discuss the effect of the hyperparameters of our method. Fig. 2.3 shows the effect of λ to the clustering results. Other parameters are fixed at the values when ARI becomes highest for each λ. There is a peak in each dataset ($\lambda = 10^{-4}$ on WebKB and $\lambda = 10^{-2}$ on Cora) which indicates that the effect



(a) WebKB (b) Cora

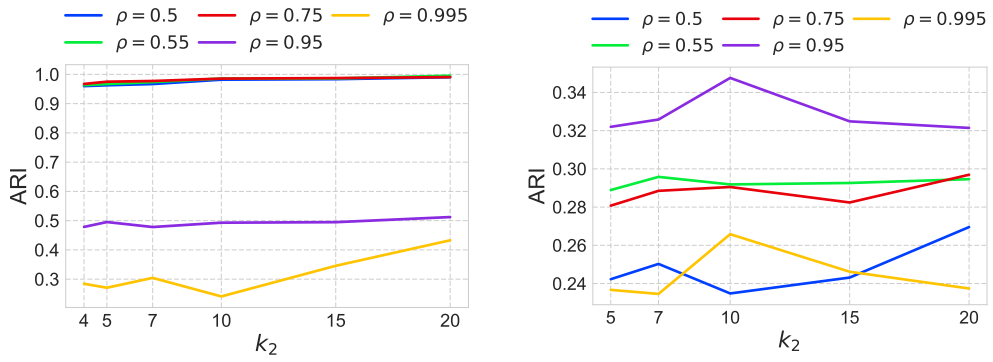
Figure 2.3: Effect of λ and ρ on ARI in our method for two datasets.

to the model is well balanced by λ between the topology and the attributes[14].

The effect of k_2 and ρ to ARI is shown in Fig. 2.4. Fig. 2.4a shows that ARI slightly increases when k_2 increases. ARI of the WebKB is enough high (almost 1.0) when $k_2 = 20$. Fig. 2.4b shows that there is a peak of ARI on Cora when $\rho = 0.95$ and $k_2 = 10$. From Fig. 2.3 and 2.4, we confirmed that ARI is stable against the selection of λ (when $\lambda < 0.1$) and k_2 in a wide range. Thus, in practice, we suppose our method would perform well when λ and k_2 may be simply chosen e.g., $\lambda = 0.01$ and $k_2 = k_1$.

As for the hyperparameter of PU learning, ρ has a large influence on the performance of our method as shown in Fig. 2.3 and 2.4. Note that, when $\rho = 0.5$, PU learning is not applied because the weights for 0 and 1 are treated as the same. To evaluate the effectiveness of the Positive Unlabeled approach, we show the effect of ρ to ARI achieved by our method in Table 2.5. It shows that, when the density is low, the best ρ tends to be high in general. This results clarify the effect of PU learning, because this setting puts more bias to positive edges and most real-world graphs are sparse. The WebKB dataset behaves differently, since it is the web graph managed by universities so there is almost no missing positive edges

[14]Even when λ is close to 0, the attributes contribute to the clustering results so the clustering performance is kept high. When lambda=0, we observe that ARI and AMI largely decrease.



(a) WebKB (b) Cora

Figure 2.4: Effect of k_2 and ρ on ARI in our method for two datasets.

2.5.7 Discussion About NAGC-U and NAGC-UH

NAGC-U is obtained from the topology cluster assignment and NAGC-UH is obtained from the attribute cluster assignment. Here we have a question: which clustering result the users should choose obtained from NAGC-U or NAGC-UH? If we know the grand truth beforehand, we can choose either of them depending on the grand truth. However, the grand truth is usually unknown in real applications. In practice, we provide the both results to the users so that they can choose more suitable one. This is a type of trial-and-error tasks during clustering analysis, such as choosing the suitable number of clusters.

We show NMI scores between the clustering results of NAGC-U and NAGC-UH in Table 2.6. NAGC-U and NAGC-UH obtain the similar clusters in five datasets which are WebKB, Wiki, Cora, Citeseer, and BlogCatalog since NMI scores are high. The difference between the clusters obtained from NAGC-U and NAGC-UH indicates that there are nodes whose cluster assignments differ depending on whether the topology or the attribute more largely effects to the clusters. On the other hand, NMI is 0.000 in Polblog. This result implies that the topology is independent from the attributes in this dataset. We also observe that in Flickr some pairs of clusters between in the topology and in the attributes overlap but others do not, since the NMI score for Flickr is relatively low.

Table 2.5: Effect of ρ on ARI achieved by our method. The density indicates the (# of partially observed positive edges)/(# of all possible edges), that is $|E|/\ n^2$.

Dataset	WebKB	Polblog	Cora	Citeseer
Density	0.18%	0.75%	0.07%	0.04%
$\rho = 0.5$	0.990	0.621	0.270	0.221
$\rho = 0.55$	**0.992**	0.625	0.296	0.216
$\rho = 0.75$	0.991	0.625	0.297	0.229
$\rho = 0.95$	0.512	**0.646**	**0.336**	0.254
$\rho = 0.995$	0.433	0.529	0.266	**0.269**

Table 2.6: NMI score for NAGC-U and NAGC-UH

Dataset	WebKB	Polblog	Wiki	Cora
NMI (U,UH)	0.834	0.000	0.735	0.838
Dataset	Citeseer	BlogCatalog	Flickr	
NMI (U,UH)	0.941	0.866	0.428	

2.6 Conclusion

We considered the clustering problem of attributed graphs. We designed an effective clustering method, NAGC, a new attributed graph clustering method by bridging the attribute space and the topology space and taking the effect of partially observed positive edges. The features of our method are two holds and both of them largely contribute to the quality of the clustering results. 1) NAGC learns a projection function between the topology space and the attribute space. The projection function consists of a rescale function and a transfer matrix that balances the effect from the attribute space to the topology space for each node independently. 2) NAGC leverages PU learning to take the effect of partially observed positive edges into the cluster assignment.

Chapter 3

GNN Transformation Framework for Improving Efficiency and Scalability

3.1 Introduction

Many GNNs have been proposed for node classification and representation learning including GCN [61], which is the most representative GNN variant, i.e., it is often used as a baseline method in other works [23, 76, 135]. Most existing GNNs adopt graph convolution that performs three tasks; 1) feature aggregation[1], 2) learnable weight multiplication, and 3) activation function application (e.g., ReLU, a non-linear function). By stacking multiple graph convolutional layers, they propagate node features over the given graph topology. However, these existing GNNs cannot be efficiently trained on large-scale graphs since the GNNs need to perform three tasks in graph convolution every time learnable weights are updated. In addition, large-scale graphs cannot be put on GPU memory for efficient matrix operations. As a result, graph convolution is not efficient and scalable for large-scale graphs.

A major approach to apply GNNs to large-scale graphs is to separate feature aggregation from graph convolution so that GNNs can precompute aggregated features [37, 82, 111]. These methods are called *precomputation-*

[1]Feature aggregation indicates aggregating node features/embeddings from neighbor nodes, which is equivalent to message (node feature) exchange between connected nodes.

based GNNs. In detail, they remove non-linearity, i.e., activation functions, from graph convolution so that feature aggregation is separated from weight learning. Thanks to the independence of feature aggregation and weight learning, precomputation-based GNNs are efficient in learning steps by precomputing feature aggregation before training learnable weights.

Though some existing works tackle the scalability problem of GNNs as discussed above, most widely studied GNNs are not scalable to large-scale graphs for the following two reasons. First, existing studies on precomputation-based GNNs [37, 82, 111] focus on introducing several specific GNN architectures that are manually designed. So, it is laborsome to apply the same precomputation idea to other GNNs. An interesting observation is that they share the common motivation: precomputation of feature aggregation is indispensable for high scalability. To our best knowledge, there are no works that study a general framework that transforms non-scalable GNNs to scalable precomputation-based GNNs. Second, existing precomputation schemes are not scalable because they need to put complete graphs (e.g., graphs with one billion edges [47]) on GPU memory. Since the size of large graphs typically exceeds the memory size of general GPU, existing works precompute feature aggregation on CPU.

To tackle the above issues, we address two research questions: **Q1**: *Can we design a general procedure that transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs while keeping their classification performance?* and **Q2**: *Can we efficiently execute the precomputation on GPU?* There are two technical challenges which must be overcome to answer our questions. First, we need to automatically transform non-scalable GNNs to precomputation-based GNNs. We should develop a common transformation procedure that can be applied to various non-scalable GNNs while preserving their expressive power. Second, we need to decompose large graphs into small groups each of which can be handled efficiently with GPU. Typically, graph decomposition suffers from an imbalance problem since node degree distributions usually follow power law distributions [86]. Hence, we should divide graphs into balanced groups and select an appropriate group size so that precomputation time is optimized.

In this chapter, we propose a framework that automatically transforms non-scalable GNNs into precomputation-based GNNs with a scalable precomputation schema. As for the first challenge, we develop a new transformation procedure, called Linear Convolution (LC) transformation, which can be applied to various non-scalable GNNs so that transformed GNNs work ef-

ficiently and scale well to large-scale graphs. Our transformation procedure removes non-linear functions from graph convolution, but incorporates non-linear functions into weight learning. This idea is derived from our hypothesis that it is not crucial to incorporate non-linearity into graph convolutional layers but into weight learning for prediction[2]. Since our transformation preserves the major functionality of graph convolution and a similar expressive power to original GNNs, the transformed GNNs can achieve competitive prediction performance to the original ones while improving their scalability. As for the second challenge, we develop a block-wise precomputation scheme which optimally decomposes large-scale graphs into small and balanced blocks each of which can fit into GPU memory. We introduce a simple decomposition approach to ensure that blocks are balanced and give minimization formulas that decide the optimal block size under limited GPU memory.

Through extensive experiments, we validate that our transformation procedure and optimized block-wise precomputation scheme are quite effective. First, we show that our LC transformation procedure transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs while keeping their node classification accuracy. Second, we show that our precomputation scheme is more efficient than that of existing precomputation-based GNNs. In summary, our transformation procedure provides simple and efficient baselines for future research on scalable GNNs by shining a spotlight on existing non-scalable methods.

The rest of this chapter is organized as follows. We describe notations and fundamental techniques for our method in Section 3.2. Section 3.3 proposes our framework. We give the purpose and results of experiments in Section 3.4. Section 3.5 describes the details of related work. Finally, we conclude this chapter in Section 3.6.

3.2 Preliminaries

An *attributed graph with class labels*[3] is a triple $G = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{C})$ where $\boldsymbol{A} \in \{0,1\}^{n \times n}$ is an adjacency matrix, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is an attribute matrix assigning

[2]The validity of this hypothesis is discussed in Section 3.3.1

[3]We re-introduce an attributed graph because a given graph includes class information in a node classification task. Note that since we consider an unsupervised task in Chapter 2, given attributed graphs in the chapter do not include class information.

Table 3.1: Notation and definitions

Variable	Explanation
n	number of nodes
d	dimension of features
y	number of classes
h	dimension of hidden layer
K	number of hidden layers
$\boldsymbol{A} \in \mathbb{R}^{n \times n}$	adjacency matrix
$\tilde{\boldsymbol{A}} \in \mathbb{R}^{N \times N}$	extended adjacency matrix
$\boldsymbol{S} \in \mathbb{R}^{n \times n}$	normalized adjacency matrix
$\boldsymbol{X} \in \mathbb{R}^{n \times d}$	feature matrix
$\boldsymbol{C} \in \mathbb{R}^{n \times y}$	class matrix
$\boldsymbol{D} \in \mathbb{R}^{n \times n}$	degree matrix
$\boldsymbol{H} \in \mathbb{R}^{n \times h}$	node embeddings
$\boldsymbol{W}_1 \in \mathbb{R}^{d \times h}, \boldsymbol{W}_{2,\ldots,K-1} \in \mathbb{R}^{h \times h}, \boldsymbol{W}_K \in \mathbb{R}^{h \times y}$	weight matrices
$\boldsymbol{Y} \in \mathbb{R}^{n \times y}$	predicted label matrix

attributes to nodes, and a class matrix $\boldsymbol{C} \in \{0,1\}^{n \times y}$ contains class information of each node, and n, d, y are the numbers of nodes, attributes, and classes, respectively. If there is an edge between nodes i and j, $\boldsymbol{A}_{ij}$ and $\boldsymbol{A}_{ji}$ are set to one. We define the degree matrix $\boldsymbol{D} = \mathrm{diag}(D_1, \ldots, D_n) \in \mathbb{R}^{n \times n}$ as a diagonal matrix, where D_i expresses the degree of node i. We also define an identity matrix $\boldsymbol{I} = \mathrm{diag}(1, \ldots, 1) \in \mathbb{R}^{n \times n}$ and an adjacency matrix extended with self-loops $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. We define node embeddings $\boldsymbol{H} \in \mathbb{R}^{n \times h}$, where h is the dimension of a hidden layer. We summarize notation and their definitions in Table 3.1.

3.2.1 Graph Convolutional Networks

Multi-layer GCN is a standard GCN model which was proposed in [61]. GCNs learn a feature representation for the feature of each node over layers. For the k-th graph convolutional layer, we denote the input node representations of all nodes by the matrix $\boldsymbol{H}^{(k-1)}$ and the output node representations by $\boldsymbol{H}^{(k)}$. The initial node representations are set to the input features, i.e.,

$\boldsymbol{H}^{(0)} = \boldsymbol{X}$. Let $\boldsymbol{S}$ denote the normalized adjacency matrix

$$\boldsymbol{S} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}. \tag{3.1}$$

This normalized adjacency matrix is commonly used as a graph filter for graph convolution. The graph filter is known as a low-pass filter that filters out noise in node features [61]. For each layer, GCN propagates the embedding of a node to its neighbors as follows:

$$\boldsymbol{H}^{(k)} = \sigma(\boldsymbol{S}\boldsymbol{H}^{(k-1)}\boldsymbol{W}_k), \tag{3.2}$$

where $\boldsymbol{W}_k$ denotes the weight matrix of the k-th layer and σ denotes a non-linear function, e.g., ReLU. In the output layer, K-layer GCN outputs a predicted label matrix $\boldsymbol{Y} \in \mathbb{R}^{n \times y}$ as:

$$\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{S}\boldsymbol{H}^{(K-1)}\boldsymbol{W}_K), \tag{3.3}$$

where $\mathrm{softmax}(\boldsymbol{P})_{ij} = \frac{\exp(\boldsymbol{P}_{ij})}{\sum_{j=1}^{y}\exp(\boldsymbol{P}_{ij})}$ for a matrix $\boldsymbol{P}$. The number of layers is typically set to $K = 2$ [61].

3.2.2 Precomputation-based GNNs

Several precomputation-based GNNs have been proposed recently [37, 82, 111]. Their fundamental and common idea is to remove non-linear functions between each layer in order to precompute feature aggregation. We explain Simplifying Graph Convolution (SGC for short) [111] which is the simplest precomputation-based GNN. Thanks to the removal, K-layer GCN can be rewritten as follows by unfolding the recursive structure:

$$\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{S}\ldots\boldsymbol{S}\boldsymbol{X}\boldsymbol{W}_1\ldots\boldsymbol{W}_K). \tag{3.4}$$

The repeated multiplication with the normalized adjacency matrix $\boldsymbol{S}$ can be simplified into a K-th power matrix $\boldsymbol{S}^K$ and the multiple weight matrices can be reparameterized into a single matrix $\boldsymbol{W} = \boldsymbol{W}_1\ldots\boldsymbol{W}_K$. The output becomes

$$\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{S}^K\boldsymbol{X}\boldsymbol{W}). \tag{3.5}$$

By separating graph feature aggregation and weight learning, SGC precomputes $\boldsymbol{S}^K\boldsymbol{X}$ before learning $\boldsymbol{W}$. The other methods also follow the same idea: separating feature aggregation and weight learning and precomputing feature aggregation.

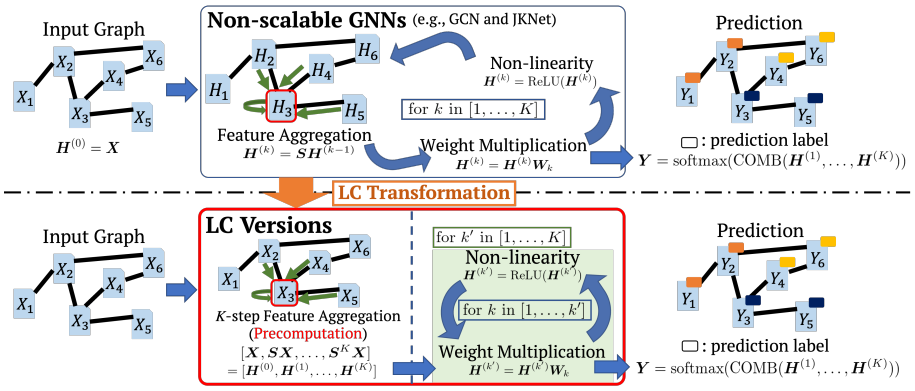


Figure 3.1: Example of LC transformation. Upper part: non-scalable GNNs operate K-layer graph convolution combining feature aggregation, weight multiplication, and activation function application (ReLU). This example corresponds to K-layer GCN if COMB outputs only $\boldsymbol{H}^K$. Lower part: LC transformation separates feature aggregation and weight learning while keeping the similar architectures with the original GNNs. LC versions avoid recomputing feature aggregation whenever learnable weights are updated at each learning step.

3.3 GNN Transformation Framework

We propose a general framework that automatically transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs and efficiently executes precomputation of feature aggregation on GPU. We first introduce a transformation procedure that automatically rewrites the formulations of non-scalable GNNs so that the transformed GNNs run efficiently and scale well to large-scale graphs (Section 3.3.1). We also describe a limitation of our transformation, namely, that it does not support GNNs that require dynamical changes of graph filters during weight learning. Our transformation procedure is applicable not only to GCN [61] but also to the state-of-the-art GNNs, such as JKNet [114], H2GCN [135] and GPRGNN [23]. Next, we introduce a block-wise precomputation scheme that efficiently computes feature aggregation for large-scale graphs (Section 3.3.2). The core idea is to decompose an adjacency matrix and feature matrix into disjoint and balanced blocks each of which can be handled on GPU. Also, we formulate and solve an optimization problem that decides the optimal size of blocks. Note

that this scheme is a general approach since it can be applied to existing precomputation-based GNNs [37, 82, 111].

3.3.1 Linear Convolution Transformation

LC transformation is the first concrete procedure that transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs, which have a similar functionality to the input GNNs. We call the output the *LC version* of the input GNN. LC transformation is motivated by the effectiveness of SGC and Multi-Layer Perceptron (MLP). SGC preserves the major benefit of graph convolution with efficient training by precomputing feature aggregation, but it degrades the accuracy due to the lack of non-linearity [23]. Beside, MLP outperforms linear regression in classification task by using non-linear functions but does not capture the structures of graphs. LC version of GNN leverages both the strengths of SGC and MLP by precomputing feature aggregation and then learning weights with non-linearity.

Figure 3.1 demonstrates an example of LC transformation by comparing it with non-scalable GNNs. Intuitively, LC transformation separates feature aggregation from graph convolution that performs 1) feature aggregation, 2) weight multiplication, and 3) activation function application (e.g., ReLU, a non-linear function). Notice that a normalized adjacency matrix $\boldsymbol{S}$ is adjacent to the feature matrix $\boldsymbol{X}$ in the formulation of LC versions (see the left part of the red box of the figure). So, we can precompute $\boldsymbol{S}^k\boldsymbol{X}$ in the same way as SGC [111]. Thanks to the separation, LC versions can avoid computing feature aggregation whenever learnable weights are updated at each learning step (see the right part of the red box of the figure). Hence, LC versions efficiently work and scale well to large-scale graphs.

Discussion

We discuss why LC versions work from two aspects, feature aggregation and weight learning. As in the discussion on the spectral analysis [111], feature aggregation acts as a low-pass filter that produces smooth features over the graph, which is the major benefit of graph convolution. In this sense, LC versions are expected to have the same functionality as the input GNNs since LC transformation preserves feature aggregation within multi-hops. As for weight learning, LC versions have a similar learning capability to their original GNNs since they have a similar model architecture of multi-layer

neural networks. As a result, LC versions can achieve a similar prediction performance to their original GNNs while scaling to large-scale graphs.

Procedure

Next, we describe the procedure of LC transformation, which removes non-linear functions from graph convolution, but incorporates non-linear functions into weight learning. We first give the definition of LC transformation below:

Definition 3.3.1 (LC transformation) *Given a non-scalable GNN algorithm, LC transformation iteratively applies a function f_{LC} to the formulation of the input GNN since non-scalable GNNs have multiple graph convolutional layers. f_{LC} commutes matrix multiplication of $\boldsymbol{S}$ and a non-linear function σ as follows:*

$$f_{LC} : g_2(\boldsymbol{S}\sigma(g_1(\boldsymbol{X}))) \underset{f_{LC}}{\longrightarrow} g_2(\sigma(\boldsymbol{S}g_1(\boldsymbol{X}))), \tag{3.6}$$

where g_1 and g_2 indicate any functions that input and output matrices. The iteration continues until the formulation does not change. LC transformation outputs a precomputation-based GNN having the transformed formulation, i.e., the LC version of the input GNN.

To intuitively explain the details, we use JKNet [114] as an example, which is a widely used GNN. The formulation of JKNet (GCN-based) is as follows:

$$\boldsymbol{H} = \mathrm{COMB}_{k=1}^{K}(\boldsymbol{S}\sigma(\boldsymbol{S}\sigma(\dots(\boldsymbol{S}\boldsymbol{X}\boldsymbol{W}_1)\dots)\boldsymbol{W}_{k-1})\boldsymbol{W}_k), \tag{3.7}$$

where COMB expresses a skip connection between different layers, such as concatenation of intermediate representations or max pooling. By applying a softmax function to feature representations $\boldsymbol{H}$, JKNet outputs a prediction result $\boldsymbol{Y}$, i.e., $\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{H})$. We apply f_{LC} to it in order to transform the formulation of an input GNN. To this end, we assign $g_1(\boldsymbol{X}) = \boldsymbol{S}\sigma(\dots(\boldsymbol{S}\boldsymbol{X}\boldsymbol{W}_1)\dots)\boldsymbol{W}_{k-1}$ and $g_2(\boldsymbol{S}\sigma(g_1(\boldsymbol{X}))) = \mathrm{COMB}_{k=1}^{K}(\boldsymbol{S}\sigma(g_1(\boldsymbol{X}))\boldsymbol{W}_k)$. By utilizing f_{LC}, g_1, and g_2, we transform Eq (3.7) as follows:

$$\boldsymbol{H} \underset{f_{LC}}{\longrightarrow} \mathrm{COMB}_{k=1}^{K}(\sigma(\boldsymbol{S}^2\sigma(\dots(\boldsymbol{S}\boldsymbol{X}\boldsymbol{W}_1)\dots)\boldsymbol{W}_{k-1})\boldsymbol{W}_k). \tag{3.8}$$

Then, we iteratively apply f_{LC} to the formulation by appropriately assigning g_1 and g_2 for each iteration. Finally, we obtain the formulation of the LC version of the input GNN, $\boldsymbol{H}^{LC}$, as follows:

$$\boldsymbol{H}^{LC} = \mathrm{COMB}_{k=1}^{K}(\sigma(\sigma(\dots(\boldsymbol{S}^k\boldsymbol{X}\boldsymbol{W}_1)\dots)\boldsymbol{W}_{k-1})\boldsymbol{W}_k). \tag{3.9}$$

Then, in the same way as the input GNN, the LC version outputs a predicted label matrix $\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{H}^{LC})$.

The LC transformation procedure is applicable not only to JKNet but also to general non-scalable GNNs including APPNP [62], MixHop [3], H2GCN [135], and GPRGNN [23].

Limitation

Precomputation-based GNNs can use multiple graph filters such as an exact 1-hop away adjacency matrix and Personalized PageRank diffusion matrix [62]. Those GNNs do not dynamically control the propagation of features during weight learning[4], since they use constant graph filters in order to precompute feature aggregation. Since our framework also leverages a precomputation scheme, it cannot support those existing GNNs [96, 106, 113] which dynamically sample edges or modify the importance of edges during weight learning. For example, Dropedge [96] randomly reduces a certain number of edges at each iteration. A possible future research direction is that we simulate random edge reduction by utilizing the deviations of feature aggregation.

3.3.2 Efficient Precomputation

Existing precomputation-based GNNs need to use CPUs to compute feature aggregations for large-scale graphs since they do not fit on GPU memory. This CPU computation has large cost and a deteriorating effect on efficiency.

To tackle this problem, we propose a simple yet efficient block-wise precomputation scheme and provide a formulation for optimal decomposition for our block-wise precomputation scheme. The core idea is to decompose the edge set of a given graph into disjoint and balanced groups, while existing approaches [131] decompose the node set into groups, i.e., row/column wise decomposition. Our scheme is inspired by edge partitioning [40, 75], which

[4]Note that this dynamic mechanism regards the adjustable propagation rule of graph convolution and is not related to the time-series of graphs.

aims to decompose a graph into groups having similar numbers of edges such that communication costs for graph operations are minimized in distributed environments. Our scheme consists of three steps. First, it decomposes an adjacency matrix and feature matrix into small disjoint blocks each of which can be put on GPU memory. Second, the scheme computes block-wise matrix operations for the disjoint blocks on GPU. Third, it aggregates the results of the block-wise matrix operations and obtains the whole matrix operation result.

Precomputation on GPU

There are two matrix operations to be precomputed, adjacency matrix normalization and feature aggregation. First, we describe the computation of adjacency matrix normalization shown by Eq. (3.1). Since an adjacency matrix is typically sparse, we utilize adjacency list $(i, j) \in \mathcal{E}$, where $\tilde{\boldsymbol{A}}_{ij} = 1$. To obtain small blocks each of which can be loaded on GPU memory, we decompose $\mathcal{E}$ into disjoint sets that include similar numbers of edges, $\mathcal{E}^{(1)} \cup \cdots \cup \mathcal{E}^{(a)}$, where a is the number of sets and $\mathcal{E}^{(p)} \cap \mathcal{E}^{(q)} = \emptyset$ if $p \neq q$. Note that the sizes of the sets $\mathcal{E}^{(1)}, \ldots, \mathcal{E}^{(a)}$ are balanced. Then, we decompose $\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}^{(1)} + \cdots + \tilde{\boldsymbol{A}}^{(a)}$, where $\tilde{\boldsymbol{A}}^{(1)}, \ldots, \tilde{\boldsymbol{A}}^{(a)} \in \mathbb{R}^{n \times n}$ and $\tilde{\boldsymbol{A}}^{(l)}_{ij} = 1$ if $(i, j) \in \mathcal{E}^{(l)}$. Then, we can rewrite Eq. (3.1) as follows:

$$\boldsymbol{S} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}} = \textstyle\sum_{l=1}^{a} \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}^{(l)} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}. \tag{3.10}$$

By appropriately selecting the number of blocks a, $\tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}}^{(l)} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ can be executed on GPU. We sum the results of the block-wise matrix computations. This summation can be efficiently computed on CPU by disjoint union of edge lists since $\mathcal{E}^{(l)}$, i.e., $\tilde{\boldsymbol{A}}^{(l)}$, is disjoint each other. Since our decomposition is agnostic on nodes, the decomposed blocks can be easily balanced while row/column(node)-wise decomposition approaches suffer from an imbalance problem. Further discussion on limitations follows below in this subsection.

Next, we introduce a block-wise computation for feature aggregation on GPU. Algorithm 2 describes the procedure of the computation. To obtain small blocks of a normalized adjacency matrix $\boldsymbol{S}$, we decompose it into $\boldsymbol{S}^{(1)}, \ldots, \boldsymbol{S}^{(b)} \in \mathbb{R}^{n \times n}$ where b is the number of blocks (line 2). Similarly to the decomposition of $\boldsymbol{A}$, each corresponding edge list is disjoint and includes similar numbers of edges. Also, in order to obtain small blocks of a feature matrix $\boldsymbol{X}$, we decompose it into $\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(c)}$, where c is the

Algorithm 2: Block-wise feature aggregation.

Input: normalized adjacency matrix $\boldsymbol{S}$, feature matrix $\boldsymbol{X}$, number of layers K
Output: aggregated feature list *SX_list*

1: $SX\_list = []$
2: $\boldsymbol{S}^{(1)}, \boldsymbol{S}^{(2)}, \ldots, \boldsymbol{S}^{(b)} = \text{split}(\boldsymbol{S})$ ▷ disjoint edge sets
3: $\boldsymbol{X}_{prev} = \boldsymbol{X}$
4: **for** $k = 1$ to K **do**
5: $\quad \boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(c)} = \text{split}(\boldsymbol{X}_{prev})$
6: $\quad$ **for** $i = 1$ to c **do**
7: $\quad\quad \boldsymbol{X}_{tmp} = [0]^{n \times \lceil d/c \rceil}$ ▷ same size to $\boldsymbol{X}^{(i)}$
8: $\quad\quad$ **for** $j = 1$ to b **do**
9: $\quad\quad\quad \boldsymbol{Z}_{tmp} = \boldsymbol{S}^{(j)}\boldsymbol{X}^{(i)}$ ▷ on GPU
10: $\quad\quad\quad \boldsymbol{X}_{tmp} = \boldsymbol{X}_{tmp} + \boldsymbol{Z}_{tmp}$ ▷ on GPU
11: $\quad\quad$ **if** $i == 1$ **then**
12: $\quad\quad\quad \boldsymbol{X}_{conc} = \boldsymbol{X}_{tmp}$ ▷ on CPU
13: $\quad\quad$ **else**
14: $\quad\quad\quad \boldsymbol{X}_{conc} = \text{concat}(\boldsymbol{X}_{conc}, \boldsymbol{X}_{tmp})$ ▷ on CPU
15: $\quad SX\_list.\text{append}(\boldsymbol{X}_{conc})$
16: $\quad \boldsymbol{X}_{prev} = \boldsymbol{X}_{conc}$

number of blocks (line 5). Since we assume that $\boldsymbol{X}$ is a dense matrix, we adopt column-wise decomposition, i.e., $\boldsymbol{X} = \text{concat}(\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(c)})$. Then, we compute matrix multiplication $\boldsymbol{S}^{(j)}\boldsymbol{X}^{(i)}$ for each pair on GPU (line 9). We aggregate $\boldsymbol{S}^{(j)}$ by summation (line 10) and aggregate $\boldsymbol{X}_{tmp}$ by concatenation (lines 11–14). $\boldsymbol{X}_{prev}$ is updated by the aggregated features $\boldsymbol{X}_{conc}$ (line 16). We repeat this aggregation K times (lines 4–16).

Optimal graph decomposition

We discuss an optimal decomposition for our block-wise precomputation scheme. We have two requirements to decompose large matrices into disjoint blocks. First, each matrix operation for disjoint blocks can be executed on GPU. Second, the number of disjoint blocks is as small as possible to reduce the number of block-wise matrix operations. To simplify the discussion, we assume that the running time of a matrix operation on GPU is the same regardless of the matrix size.

As for the block-wise adjacency matrix normalization, we minimize the number of disjoint blocks, a. We formulate the minimization as follows:

$$\min(a), \text{ subject to } \frac{\alpha_{\boldsymbol{A}} B_{\boldsymbol{A}} + \alpha_{\boldsymbol{S}} B_{\boldsymbol{S}}}{a} + \alpha_{\boldsymbol{D}} B_{\boldsymbol{D}} \leq B_{\text{GPU}}, \tag{3.11}$$

where $\alpha_{\boldsymbol{A}}, \alpha_{\boldsymbol{S}}$, and $\alpha_{\boldsymbol{D}}$ indicate coefficients for executing matrix operations regarding $\boldsymbol{A}, \boldsymbol{S}$, and $\boldsymbol{D}$, respectively, and $B_{\boldsymbol{A}}, B_{\boldsymbol{S}}, B_{\boldsymbol{D}}$, and B_{GPU} indicate the volume of an adjacency matrix, the volume of a normalized adjacency matrix, the volume of a degree matrix, and the available volume of a GPU, respectively. As for block-wise feature aggregation, we minimize the number of pairs of disjoint blocks, bc. We formulate the minimization as follows:

$$\min_{b,c}(bc), \text{ subject to } \frac{\beta_{\boldsymbol{S}} B_{\boldsymbol{S}}}{b} + \frac{\beta_{\boldsymbol{X}} B_{\boldsymbol{X}}}{c} \leq B_{\text{GPU}}, \tag{3.12}$$

where $\beta_{\boldsymbol{S}}$ and $\beta_{\boldsymbol{X}}$ indicate coefficients for executing matrix operations regarding $\boldsymbol{S}$ and $\boldsymbol{X}$, respectively, and $B_{\boldsymbol{X}}$ indicates the volume of a feature matrix. Note that $\alpha_{\boldsymbol{A}}, \alpha_{\boldsymbol{S}}, \alpha_{\boldsymbol{D}}, \beta_{\boldsymbol{S}}$, and $\beta_{\boldsymbol{X}}$ depend on execution environments[5].

Next, we discuss optimization regarding Eq. (3.11) and (3.12). As for Eq. (3.11), it is trivial to find the minimum number of blocks a since there are no other parameters. As for Eq. (3.12), an exhaustive search is applicable since the number of combinations of b and c (natural numbers) is not large. Consequently, these optimization problems can be easily solved.

Limitation

Our precomputation scheme focuses on feature aggregation on a whole graph. This indicates that our scheme is not suitable for node-wise operations since it may decompose the edge set of the same node into different groups. However, accelerating feature aggregation on a whole graph is still crucial since many GNNs [37, 61, 82, 111] adopt it.

3.4 Experiments

We design our experiments to answer the following questions; **Q1**: Can our LC transformation improve the efficiency and scalability of GNNs? **Q2**: Can

[5]In real environments, users can measure $\alpha_{\boldsymbol{A}}, \alpha_{\boldsymbol{S}}, \alpha_{\boldsymbol{D}}, \beta_{\boldsymbol{S}}$, and $\beta_{\boldsymbol{X}}$ by monitoring the memory usage on small graphs, even if users do not know the details of their own environments.

Table 3.1: Summary of datasets.

Dataset	Nodes	Edges	Features	Classes
Flickr	89,250	899,756	500	7
Reddit	232,965	11,606,919	602	41
arxiv	169,343	1,166,243	128	40
papers100M	111,059,956	1,615,685,872	128	172

our block-wise precomputation scheme accelerate precomputation?[6]

Dataset. We use four commonly used datasets, `Flickr` [127], `Reddit` [42], `ogbn-arxiv` (arxiv for short), and `ogbn-papers100M` (papers100M for short) [47]. Table 3.1 provides the summary of the datasets. The sizes of the datasets range from 9K nodes to 110M.

In the Flickr dataset, nodes represent images uploaded to Flickr. If two images share common properties such as same geographic location, same gallery, comments by the same users, there is an edge between the nodes. Node features represent the 500-dimensional bag-of-words associated with the image (node). As for the class labels of nodes, the authors of [127] scan over 81 tags of each image and manually merged them to 7 classes. In the Reddit dataset, nodes represent posts. If the same user left comments on two posts, then there is an edge between the two posts. Node features are the embedding of the contents of the posts. The labels of nodes indicate communities which the nodes belong to. In the ogbn-arxiv dataset, nodes represent ARXIV papers and edges indicate that one paper cites another one. Node features represent 128-dimensional feature vectors obtained by averaging the embeddings of words in titles and abstracts. The class labels of nodes indicate subject areas of ARXIV CS papers[7]. In the ogbn-papers100M (papers100M) dataset, its graph structure and node features are constructed in the same way as ogbn-arxiv. Among its nodes, approximately 1.5 million nodes are labeled with one of ARXIV's subject areas. As in [126], Flickr and Reddit are under the inductive setting. ogbn-arxiv and ogbn-papers100M are under the transductive setting. As for the inductive setting, we remove the nodes in the test set during a model train phase and then conduct inference on a whole graph during a test phase. On the other hand, as for the transductive

[6]Though the questions **Q1** and **Q2** look slightly different from the research questions discussed in Section 3.1, we just rewrite the questions more concretely.

[7]`https://arxiv.org/archive/cs`

setting, models can see all nodes in datasets during a model train phase, i.e., the nodes in the test set are regarded as unlabeled nodes.

Baseline. We compare three types of existing methods as baselines; non-scalable GNNs, precomputation-based GNNs, and sampling-based GNNs which are scalable but inefficient (we discuss the details in Section 3.5). As for non-scalable GNNs, we use GCN[8] [61], JKNet[9] [114], and GPRGNN[10] [23]. As for precomputation-based GNNs, we use SGC[11] [111] and FSGNN[12] [82]. As for sampling-based GNNs, we use ShaDow-GNN[13] [126]. FSGNN and ShaDow-GNN are the state-of-the-art precomputing-based and sampling-based GNNs, respectively. We note that we use our block-wise precomputation to the precomputation-based GNNs instead of using its original CPU computation for a fair comparison.

Setup. We tune hyperparameters on each dataset by Optuna [6] and use Adam optimizer [59]. We adopt mini-batch training for precomputation-based GNNs, sampling-based GNNs, and LC-versions to deal with large-scale graphs. As for ShaDow-GNN, we use the best hyperparameter sets provided by the authors and adopt GAT [106] as a backbone model since ShaDow-GAT achieves the best accuracy in most cases reported in the paper. We measure training time on a NVIDIA Tesla V100S GPU (32GB) and Intel(R) Xeon(R) Gold 5220R CPUs (378GB).

3.4.1 Effectiveness of LC Transformation (Q1)

Table 3.2 shows the test accuracy of LC versions and the baselines. LC versions (GCN_LC, JKNet_LC, and GPRGNN_LC) achieve comparable test accuracy with their original GNNs (GCN, JKNet, and GPRGNN) for all datasets. Next, Table 3.3 shows the training time of LC versions and the baselines. The LC versions run faster than their original GNNs. Note that LC versions tend to stop earlier than non-scalable GNNs since LC versions train their models more times due to mini-batch training. For example, in Flickr data LC versions more efficiently train than non-scalable GNNs even

[8] `https://github.com/tkipf/pygcn`

[9] Since official codes of JKNet from the authors are not provided, we simply implement JKNet based on the implementation of GCN.

[10] `https://github.com/jianhao2016/GPRGNN`

[11] `https://github.com/Tiiiger/SGC`

[12] `https://github.com/sunilkmaurya/FSGNN`

[13] `https://github.com/facebookresearch/shaDow_GNN`

Table 3.2: Comparison on test accuracy. We report the average values (standard deviation) over 5 runs.

	Flickr	Reddit	arxiv
GCN	0.525(0.003)	0.945(0.000)	0.702(0.005)
JKNet	0.526(0.004)	0.941(0.006)	0.712(0.001)
GPRGNN	0.494(0.006)	0.918(0.012)	0.694(0.006)
SGC	0.494(0.037)	0.948(0.001)	0.692(0.004)
FSGNN	0.513(0.001)	0.964(0.001)	0.722(0.003)
ShaDow-GAT	0.531(0.003)	0.947(0.003)	0.716(0.004)
GCN_LC	0.515(0.003)	0.947(0.001)	0.710(0.001)
JKNet_LC	0.517(0.004)	0.951(0.000)	0.710(0.003)
GPRGNN_LC	0.513(0.001)	0.961(0.000)	0.720(0.004)

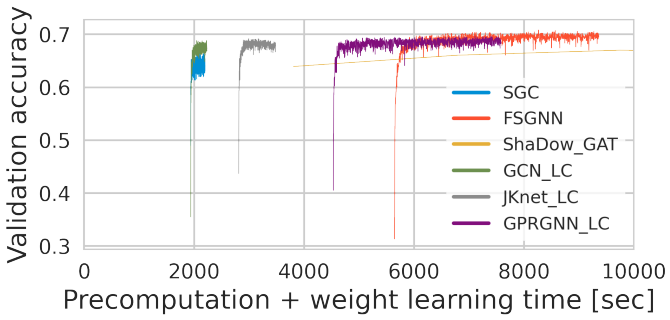


Figure 3.2: Validation accuracy over training time (precomputation and weight learning time) on papers100M. Plots indicate epochs. LC versions (GCN_LC, JKNet_LC, and GPRGNN_LC) are faster than FSGNN and ShaDow-GAT while achieving competitive accuracy.

if they have similar training time per epoch. These results indicate that our framework transforms non-scalable GNNs into efficient precomputation-based GNNs with the comparable classification accuracy to the original GNNs.

Comparison on a large-scale graph. Table 3.4 shows the performance comparison on papers100M having more than 100 million nodes and one billion edges. Non-scalable GNNs (GCN, JKNet, and GPRGNN) cannot work on papers100M since the whole graph cannot be put on GPU mem-

Table 3.3: Comparison on training time (per epoch/total). Note that total training time includes precomputation time for SGC, FSGNN, ShaDow-GAT, GCN_LC, JKNet_LC, and GPRGNN_LC. We report the average values over 5 runs.

	Flickr	Reddit	arxiv
GCN	64.62[ms] / 129.24[s]	654.70[ms] / 1309.40[s]	210.81[ms] / 421.63[s]
JKNet	170.43[ms] / 253.25[s]	1428.51[ms] / 2552.45[s]	529.05[ms] / 1058.10[s]
GPRGNN	272.86[ms] / 539.48[s]	1456.01[ms] / 2806.62[s]	523.08[ms] / 961.76[s]
SGC	51.18[ms] / 30.31[s]	141.68[ms] / 285.43[s]	50.27[ms] / 42.23[s]
FSGNN	346.97[ms] / 133.63[s]	1066.66[ms] / 1793.91[s]	284.73[ms] / 382.67[s]
ShaDow-GAT	120.85e3[ms] / 3634.65[s]	376.42e3[ms] / 11321.09[s]	163.67e3[ms] / 4913.29[s]
GCN_LC	56.75[ms] / 49.85[s]	165.73[ms] / 212.16[s]	62.59[ms] / 120.60[s]
JKNet_LC	144.78[ms] / 78.24[s]	430.41[ms] / 865.71[s]	138.52[ms] / 277.63[s]
GPRGNN_LC	287.54[ms] / 164.88[s]	818.13[ms] / 1645.49[s]	219.66 [ms] / 204.56[s]

Table 3.4: Results on papers100M. We show test/validation accuracy (standard deviation) and training time (per epoch / total). Total training time includes precomputation time. OOM indicates that the execution is out of memory.

	Test accuracy	Val accuracy	Time (epoch / total)
GCN	OOM	OOM	OOM
JKNet	OOM	OOM	OOM
GPRGNN	OOM	OOM	OOM
SGC	0.623(0.007)	0.667(0.002)	425.15[ms] / 2211.23[s]
FSGNN	0.665(0.003)	0.706(0.001)	3550.82[ms] / 8612.48[s]
ShaDow-GAT	0.666(0.003)	0.703(0.001)	2948.50e3[ms] / 92264.76[s]
GCN_LC	0.647(0.006)	0.688(0.002)	611.90[ms] / 2477.55[s]
JKNet_LC	0.641(0.003)	0.689(0.004)	1488.80[ms] / 3396.69[s]
GPRGNN_LC	0.658(0.002)	0.696(0.001)	2749.27[ms] / 7410.47[s]

ory. GPRGNN_LC achieves comparable accuracy (approximate one percent difference) with FSGNN, which is the state-of-the-art precomputation-based GNN while GPRGNN_LC runs over 10% faster than FSGNN in terms of both training time per epoch and total time. This indicates that our LC transformation can automatically provide precomputation-based GNN algorithms that achieve similar accuracy to the state-of-the-art GNN (and also run faster) without manually designing new specific algorithms. Though ShaDow-GAT achieves the highest accuracy, it requires more than 10× to-

tal training time than other models. This is because it needs to operate graph convolutions on many enclosing subgraphs extracted from the whole graph. SGC obtains lower accuracy than GCN_LC. This result validates that non-linearity contributes to weight learning for better classification.

In order to analyze the results on papers100M in details, we show the validation accuracy at each epoch over total training time in Figure 3.2. Note that total training time consists of precomputation and weight learning time. We observe that GCN_LC, JKNet_LC, and GPRGNN_LC are plotted in the upper left corner of the figure. This observation indicates that they require less total training time than FSGNN and ShaDow-GAT. The LC versions achieve competitive performance with them. Through these experiments, we demonstrate that LC versions are efficient and scalable for large-scale graphs.

3.4.2 Precomputation Efficiency (Q2)

To validate the efficiency of our block-wise precomputation, we compare it with naive CPU computation adopted by existing works [47, 82]. We use a large-scale graph, papers100M, which requires a 67GB normalized adjacency matrix and a 57GB feature matrix. For adjacency matrix normalization, we set the number of disjoint blocks of an adjacency matrix to $a = 3$, which satisfies Eq. (3.11). Also, for feature aggregation we set numbers of disjoint blocks of a normalized adjacency matrix and feature matrix to $b = 10$ and $c = 16$, respectively, which satisfy Eq. (3.12).

Figure 3.3 shows the precomputation time for normalization and feature aggregation on CPU and GPU. The result demonstrates that our block-wise precomputation is 20× faster than CPU computation for normalization. Also, the result indicates that our precomputation is up to twice faster than CPU computation for feature aggregation. Hence, we conclude that our precomputation is more efficient than CPU computation on a single machine.

3.5 Related Work

Relationship between non-scalable GNNs and LC versions. We discuss the background of non-scalable GNNs and their LC versions. Graph convolution is motivated by the 1-dim Weisfeiler-Lehman (WL-1) algorithm [110] which is used to test graph isomorphism; two graphs are called isomorphic if they are topologically identical. WL-1 iteratively aggregates the labels

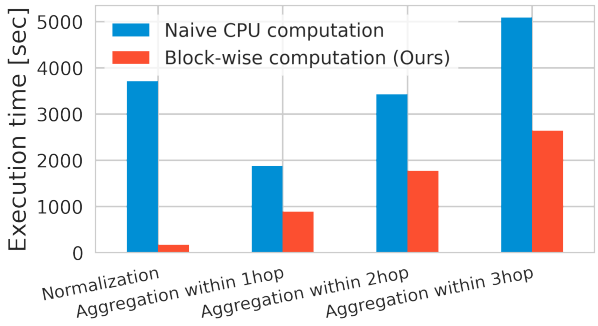


Figure 3.3: Precomputation time comparison between a naive CPU computation and our block-wise computation.

of nodes and their neighbors, and hashes the aggregated labels into unique labels. The algorithm decides whether two graphs are isomorphic or not by using the labels of nodes at some iteration. Non-scalable GNNs such as GCN [61] replace the hash function of WL-1 with a graph convolutional layer which consists of feature aggregation, weight multiplication, and non-linear function application. As for LC versions, they replace the hash function of WL-1 with feature aggregation. These observations indicate that WL-1 is analogous to feature aggregation of LC versions, similarly to graph convolution of non-scalable GNNs.

Sampling-based GNNs. Sampling-based GNNs [21, 22, 42, 126, 127] avoid keeping a whole graph on GPU by computing node representations from enclosing subgraphs of the input graph. The major drawback of the sampling-based GNNs is that they need costly training time since they need to operate graph convolutions on many enclosing subgraphs extracted from the input graph.

GNNs dynamically modifying the importance of edges. As we discussed in Section 3.3.1, our transformation cannot support GNNs which dynamically control the propagation of features during weight learning. An example of such GNNs is GAT [106], which learns attention parameters controlling the importance of edges for each iteration. Another example is GIN [113] learns a parameter controlling a weight between self features and features from neighbors. One possible direction is that we first determine the parameters by training on a subset of an input graph, then fix them in order to precompute feature aggregation.

Distributed matrix operations. Matrix operations can be parallelized for distributed computing [11, 17]. For example, the authors of [43] proposed Mars which is an approach for hiding the programming complexity of MapReduce on GPU. Also, MR-Graph [94] is a customizable and unified framework for GPU-based MapReduce. It allows its users to implement their applications more flexibly. As for distributed graph neural network training, DistDGL [131] has proposed mini-batch training on graphs, which scales beyond a single machine. It suffers from an imbalance problem since it uses a typical graph clustering algorithm METIS [56] to partition large-scale graphs into subgraphs, while our scheme can partition an edge set into balanced subsets. For further scale up of graphs, it would be important to combine distributed computing and our block-wise precomputation for graphs.

3.6 Conclusion

We presented a framework that automatically transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs. There are two major characteristics of our framework: 1) it supports a novel transformation procedure that transforms non-scalable GNNs to efficient and scalable precomputation-based GNNs having a similar functionality to the original GNNs, 2) the precomputation of the transformed GNNs can be efficiently executed by our block-wise precomputation scheme that decomposes large-scale graphs into disjoint and balanced blocks each of which can be handled on GPU memory. Through our experiments, we demonstrated that the transformed GNNs run more efficiently than their original GNNs and can be scaled to graphs with millions of nodes and billions of edges. Due to the strong performance of LC versions, we argue that LC versions will be beneficial as baseline comparisons for future research on scalable GNNs.

Chapter 4

Comprehensive Evaluation of Graph Machine Learning Methods with Flexible Synthetic Graph Generator

4.1 Introduction

Towards the practical use cases of graph machine learning methods, researchers and developers need to deeply understand the pros and cons of graph machine learning methods from various aspects, e.g., when methods work well or not. However, it is still challenging to comprehensively evaluate such graph machine learning methods because a variety of real-world graphs is limited, leading to the lack of fine-grained analysis.

4.1.1 Limitations of Existing Evaluation of GNNs

Several studies [30, 38] addressed benchmarking the performance of GNNs. However, the comprehensive evaluation of GNNs has been challenging since most models are assessed on well-known but limited benchmark datasets such as Cora, Citeseer, and PubMed [121]. Although recent studies [47, 73] provide a collection of real-world datasets to mitigate the shortage of datasets, these datasets are still insufficient to understand the pros and cons of various model architectures.

To examine GNNs on graphs with different characteristics, a study [30] uses a commonly used graph generator, SBM [2]. However, the quality and variety of synthetic graphs generated by SBM are limited since it cannot generate realistic graphs and does not support generating node attributes. Due to the limitation, existing benchmarking studies of GNNs using real-world and synthetic graphs lack fine-grained analysis, e.g., the evaluation on graphs by changing one or a few target characteristic(s), while keeping other characteristics unchanged. To the best of our knowledge, no studies conducted comprehensive evaluations of GNNs on various synthetic graphs with controlled characteristics regarding classes despite the great interest in developing and evaluating GNNs.

Hence, in this chapter, we first address developing a new synthetic graph generator that can generate realistic graphs with user specified characteristics and then conduct an empirical study of node classification by using the new graph generator.

4.1.2 Requirements of New Graph Generators

New synthetic graph generators should satisfy two requirements: the quality and efficiency of graph generation. To ensure the quality of generated graphs, generators need to support several phenomena that real-world graphs exhibit. Also, to efficiently generate large-scale graphs, graph generation steps should be efficient and scalable.

Phenomena in attributed graphs

The major topological characteristics that web graphs and social networks have are small-world property and power law node degree distribution. Regarding the interplay between classes, attributes, and topology, we focus on two widely known phenomena in attributed graphs: ***core/border*** and ***network homophily/heterophily***.

Core/border. There are two kinds of nodes in a class, core and border [33]. The core nodes in a class are nodes with similar attribute values to the average attribute values of the nodes in the class. The border nodes in a class are nodes with attribute values mixed from nodes in different classes. In other words, core and border nodes strongly and weakly belong to their classes, respectively. To generalize these phenomena, we assume that each

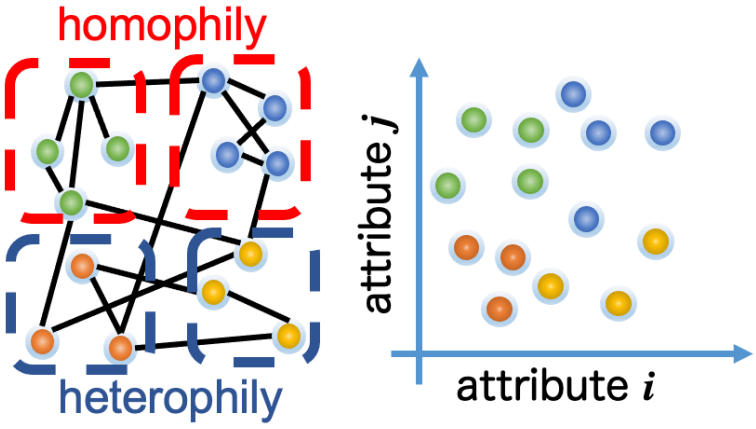


Figure 4.1: Colors indicate the classes which nodes belong to. The nodes in each class have similar attributes. The nodes in a class with the homophily property (surrounded by a red circle) tend to connect internally. The nodes in a class with the heterophily property (surrounded by a blue circle) tend to connect externally.

node is correlated with multiple classes with certain degrees[1]

Homophily/heterophily. Homophily and heterophily are phenomena to express the relationships between classes and the topology. First, homophily is a well-known phenomenon of real-world graphs where nodes in the same class are more likely to connect to each other. Classes with the homophily property are called communities, i.e., sets of nodes that are densely connected internally [83]. Second, heterophily is the inverse notion of homophily: nodes with dissimilar attributes are more likely to connect to each other. From the viewpoint of classes, nodes in a class with heterophily property are more likely to connect to other classes than the class. Figure 4.1 shows an intuitive example of classes with homophily/heterophily properties. Recent graph machine learning methods [134, 135] aim to capture both homophily/heterophily phenomena.

In summary, to capture the class structure, i.e., the relationships between classes, attributes, and topology, graph generators should support the core/border and homophily/heterophily phenomena.

[1]A class label represents the class to which a node is most correlated.

Efficiency of graph generation

Since analyzing large-scale graphs is one of the hottest topics, graph generators should be efficient and scalable to evaluate methods for large-scale graphs. However, it is not trivial to satisfy the constraints of the graph generation problem since they are interdependent and the problem of satisfying even the single constraint on node degrees is NP-complete [13]. Moreover, the high-quality and efficiency of graph generation, are in a trade-off relationship. Hence, the generator should be designed to balance the two aspects.

4.1.3 Existing Generators

From the above discussion, graph generators should support class structure and be efficient/scalable. However, most existing techniques do not explicitly generate class labels and control the class structure of graphs. gMark [12, 13], pgMark [14], and TrillionG [92] support class labels but are not designed to control the class structure. LFR [67] and DC-SBM [54] are designed to support the relationship between the class labels and topology for evaluating the performance of community detection [123] and label propagation methods [66]. They cannot simulate the core/border phenomena because they do not control the connection proportions to classes from individual nodes (i.e., all nodes in a class have the same connection proportions). Some other methods for generating attributed graphs with class labels have been proposed, such as ANC [68] and DANCer [16]. These attributed graph generators cannot control the homophily/heterophily phenomena in each class since they can set the same ratio of intra- and inter-edges to all classes. Moreover, ANC and DANCer cannot control the variety of connection proportions between classes but each pair of classes in real-world graphs may have different connection proportions. As for the efficiency of graph generation, graph generators supporting the relationship between the class labels do not scale well due to their heavy edge generation steps that require over linear time to the number of generated edges.

Graph generators based on deep learning have also been proposed recently, such as VGAE [60], GraphVAE [103], and NetGAN [19]. They aim to reproduce synthetic graphs from given input graphs, but they do not support generating graphs with user-controlled class structure.

In summary, there are no graph generators that support flexibly controlling the proportions of connections between nodes and classes, leading to the

lack of comprehensive evaluation of node classification.

4.1.4 Contributions

To address the above issue regarding the evaluation of graph machine learning methods with various graphs, in this chapter we propose an evaluation framework with synthetic graphs, which allows users to evaluate their methods from various aspects. The contributions of this evaluation framework are two-fold.

New synthetic graph generator

We propose a new graph generator, GenCAT, for synthetic graph generation satisfying the aforementioned requirements. GenCAT allows us to flexibly control the class structure of a generated attributed graph by capturing core/border and homophily/heterophily phenomena. We introduce ***node-class membership proportion*** and ***node-class connection proportion*** to capture the core/border and the homophily/heterophily phenomena, respectively. First, the node-class membership proportion of a node represents how likely the node belongs to classes. A core node has a value close to 1 for its class and a border node has balanced values for multiple classes. Second, the node-class connection proportion of a node represents how likely the node connects to classes. The node-class membership and connection proportions of a node should be similar if the node's class has the homophily property and be the opposite if the node's class has the heterophily property.

To efficiently generate graphs, we heuristically assign priorities to the constraints of graph characteristics to be satisfied and take an effective edge generation approach that utilizes inverse transform sampling [27]. Thanks to this approach, GenCAT can generate graphs in linear time to the number of edges.

We summarize the properties of graph generators in Table 4.1. GenCAT supports all of the desired properties, whereas existing methods lack one or more of them. Since GenCAT is a general generator that can support various settings, it can also simulate the existing generators LFR and DC-SBM by giving appropriate parameters (the details are described in Section 4.3.5) without any modifications of GenCAT itself.

The three main characteristics of GenCAT are as follows:

Table 4.1: Properties of graph generators: ✓ and × indicate that the generators satisfy and do not satisfy the characteristic, respectively. The topological reproducibility (the second right-most column) indicates whether generators can reproduce input graphs. The complexity column indicates the time complexity including training steps. $O(m+)$ represents the complexity which is $O(m)$ when the number of nodes is fixed. *: the time complexity of ANC and DANCer also depends on the cost of a clustering method, e.g., the cost of KMedoids is $O(n^2y)$. p represents the number of threads, f the dimension of the latent factors and B the batch size. See Table 4.3 for other parameters.

Generator	Node-class membership proportion	Class labels (class size)	Node attribute	Topological reproducibility	Complexity
LFR [67]	✓(class-level only)	✓(power law only)	×	×	$O(m+)$
DC-SBM [54]	✓(class-level only)	✓(input list)	×	×	$O(n^2)$
ANC [68]	✓(class-level only)	✓(clustering method-driven)	✓	×	$O(m)^*$
DANCer [16]	✓(class-level only)	✓(clustering method-driven)	✓	×	$O(m)^*$
pgMark [12, 13, 14]	×	✓(schema-driven)	✓	×	$O(m)$
TrillionG [92]	×	✓(schema-driven)	×	×	$O((m \log n)/p)$
VGAE [60]	×	×	×	✓	$O(fn^2)$
GraphVAE [103]	×	×	×	✓	$O(n^4)$
NetGAN [19]	×	×	×	✓	$O(B)$
GenCAT	✓	✓(power law, normal, and input list)	✓	✓	$O(myr + dyn)$

- GenCAT generates graphs with user-specified node degrees and the connection proportions between nodes and classes.
- The attribute values generated by GenCAT follow user-specified distributions and users can flexibly control the correlation between the attributes and classes.
- GenCAT scales linearly to the number of generated edges and can generate graphs with billion edges.

GenCAT is the first method having all three of these desirable characteristics. For community use and further study, our complete code base is available as open source.[2]

Empirical study of node classification

The comprehensive evaluation of GNNs has not been conducted due to limited real-world benchmark datasets. To understand the pros and cons of GNNs, we empirically study the performance of GNNs through extensive experiments on various graphs by synthetically changing one or a few target characteristic(s) of graphs. In terms of the four major characteristics of attributed graphs with class labels, 1) class size distributions, 2) the relationship between classes and topology, 3) the relationship between classes and attributes, and 4) graph sizes, we attempt to answer the following questions: **Q1.** *To what extent do class size distributions affect the performance of GNNs?* **Q2.** *How effectively do GNNs work on graphs with various edge connection proportions between classes?* **Q3.** *To what extent do attribute values contribute to the performance of GNNs?* **Q4.** *How effectively do GNNs work on graphs with various sizes?*

We summarize key takeaways of our empirical study, which we hope could benefit the community focusing on developing new GNN algorithms:

- GNNs including the state-of-the-art methods suffer from a class imbalance problem that typically deteriorates the performance of multi-class classification. Interestingly, the simplest algorithm SGC [111] outperforms recent complicated GNN algorithms in a class imbalanced setting because the complicated algorithms tend to over-fit to major classes.

[2] `https://github.com/seijimaekawa/GenCAT`

- GNNs generalizing to graphs with few edges within each class (we call heterophilic graphs) provide marginal classification performance gains in a heterophily setting from a graph-agnostic classifier MLP. This indicates that such GNNs almost ignore the topology information in the setting.

We hope the use of our framework will greatly relieve the burden of comparing existing baseline GNNs and developing new algorithms. Our codebase[3] is available under the MIT License.

Organization. The rest of this chapter is organized as follows. We describe the problem statement and the challenges of a graph generation problem in Section 4.2. We propose GenCAT in Section 4.3. Section 4.4 gives a detailed experimental analysis of the quality of generated graphs and the efficiency of graph generation. Then, Section 4.5 shows our empirical studies of node classification with GNNs by using GenCAT. We also position our generator and empirical studies in Section 4.6. Finally, we conclude this chapter in Section 4.7

4.2 Problem statement

In this section, we first describe notation and define the graph/class features of attributed graphs. Then, we define our problem and describe challenges to solve our problem.

Notation

As we defined in Section 3.2, an attributed graph with class labels is $G = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{C})$. Also, we define Ω_l as the class for label l, i.e., the set of nodes labeled with l.

4.2.1 Graph Features

We highlight two features that real-world graphs typically have: topology statistics and attribute statistics, which are desirable in synthetically generated graphs.

[3] `https://github.com/seijimaekawa/empirical-study-of-GNNs`

Topology statistics. Real graphs have well-known topological statistics [70, 86]: for example, node degrees in social graphs often follow a power law distribution. For this reason, graph generators should support various types of distributions of node degrees.

Attribute statistics. Attributes in real datasets typically follow underlying distributions. For example, binary categories follow Bernoulli distribution, such as word appearance and questions with two possible answers. Also, it is well known that many numerical attributes follow a normal distribution, such as biological data and data including measurement errors. So, graph generators should support various types of the distribution of attribute values such as Bernoulli and normal distributions, which support typical attributes.

4.2.2 Class features

As we mentioned in Section 4.1, graph generators should support the node-class membership/connection proportions capturing the phenomena of core/border and homophily/heterophily. To generate the latent factors for expressing the node-class membership/connection proportions, we identify three basic statistics of class features as input parameters: class preference mean, class preference deviation, and class size distribution. Also, to generate the latent factor for expressing the relationship between the attributes and classes, we identify a statistic of class features for the attributes: attribute-class correlation.

Class preference mean. To simulate the homophily/heterophily phenomena, we introduce ***class preference mean***, $\boldsymbol{M} \in \mathbb{R}^{k \times k}$. An element of class preference mean, $\boldsymbol{M}_{l_1 l_2}$ expresses the average of connection proportions from the nodes in class l_1 to the nodes in class l_2. We formulate class preference mean between class l_1 and class l_2 as follows:

$$\boldsymbol{M}_{l_1 l_2} = \frac{1}{|\Omega_{l_1}|} \sum_{i \in \Omega_{l_1}} \Big(\sum_{j \in \Omega_{l_2}} \boldsymbol{A}_{ij} / \sum_{j=1}^{n} \boldsymbol{A}_{ij} \Big). \tag{4.1}$$

Class preference mean is a more general notion than the simple binary representation of homophily/heterophily. For example, if $\boldsymbol{M}_{ll} = 0.7$ and $y = 3$, class l has a stronger homophily property because the nodes in class l are more likely to connect to each other than the nodes in other classes. Note that the diagonal elements express the proportions of the connections inside of each class.

Class preference deviation. We also introduce ***class preference deviation***, $\boldsymbol{D} \in \mathbb{R}^{y \times y}$, in order to simulate the core/border phenomena. Class preference deviation indicates the variety of node-class membership proportions between classes. That is, it expresses the extent to which nodes in a class belong to multiple classes. Class preference deviation is a more general notion than the simple binary representation of core/border. An element of class preference deviation, $\boldsymbol{D}_{l_1 l_2}$ indicates the standard deviation of connection proportions from nodes in class l_1 to nodes in class l_2. We formulate class preference deviation between class l_1 and class l_2 as follows:

$$\boldsymbol{D}_{l_1 l_2} = \sqrt{\frac{1}{|\Omega_{l_1}|} \sum_{i \in \Omega_{l_1}} (\sum_{j \in \Omega_{l_2}} \boldsymbol{A}_{ij} / \sum_{j=1}^{n} \boldsymbol{A}_{ij} - \boldsymbol{M}_{l_1 l_2})^2}. \tag{4.2}$$

That is, class preference mean and class preference deviation express the average and deviation of the connection proportions between classes, respectively.

Class size distribution. The class preference mean and class preference deviation capture detailed characteristics of classes, however, they lack information of class sizes. We introduce ***class size distribution***, which complements them. In many real-world graphs, such as social networks, the distribution of class sizes is usually well-approximated by power law [24, 89].

Attribute-class correlation. Since the relationship between the attributes and classes is typically various, we assume that each attribute is correlated to classes with certain degrees. To capture the correlation, we introduce ***attribute-class correlation***, $\boldsymbol{H} \in \mathbb{R}^{d \times y}$, which represents the strength of the correlation between the attributes and classes. We formulate attribute-class correlation between attribute δ and class l as the average of values of attribute δ of nodes in class l:

$$\boldsymbol{H}_{\delta l} = \frac{1}{|\Omega_l|} \sum_{i \in \Omega_l} (\boldsymbol{X}_{i\delta}). \tag{4.3}$$

Since we assume that the topology and the attributes share the same class structure, we utilize node-class membership proportions and the attribute-class correlation in order to capture the correlations between nodes and attributes.

4.2.3 Problem Definition and Challenges

We can now define the problem that we solve in this chapter. We assume two practical usage scenarios as follows. In the first scenario, the user inputs statistics of graphs to be generated so as to flexibly control the characteristics of generated graphs. In the second scenario, the user inputs graphs with class labels so as to generate graphs similar to the given input graphs.

Problem Definition

We give two definitions for these two usage scenarios. Given either 1) statistics of graphs; graph features (node degree and attribute distributions) and class features (class preference mean, class preference deviation, class size distribution, and attribute-class correlation) [4] or 2) topological information (node degree, class preference mean, class preference deviation and class size distribution) extracted from adjacency matrix $\boldsymbol{A}'$ and class labels $\boldsymbol{C}'$ of a given graph, attribute distributions, and attribute-class correlation, we efficiently generate $G = (\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{C})$ whose statistics are similar to the inputs.

To address this graph generation problem, we define the loss L that indicates the difference between the user-specified statistics and the statistics of a generated graph. We formulate the loss as follows:

$$L = L^{\text{topo}} + L^{\text{attr}} \tag{4.4}$$

where L^{topo} indicates the loss of the topological part and L^{attr} indicates the loss of the attribute part. We design the losses, L^{topo} and L^{attr}, in order to clarify the relationship between the graph generation problem and the constraints that generated graphs should satisfy graph features and class features. First, we formulate the loss L^{topo} as follows:

$$L^{\text{topo}} = L^{\text{topo}}_{\text{graph feature}} + L^{\text{topo}}_{\text{class feature}} \tag{4.5}$$

where $L^{\text{topo}}_{\text{graph feature}}$ indicates the topological loss between the given graph features and the graph features in the generated graph and $L^{\text{topo}}_{\text{class feature}}$ indicates

[4] To make the input easier, we can accept the diagonal elements of class preference mean instead of class preference mean and class preference deviation. In this case, the deviation is randomly generated since we generate class preference mean from dirichlet distribution.

a topological loss between the given class features and the class features in the generated graph. Second, we formulate the loss L^{attr} as follows:

$$L^{\text{attr}} = L^{\text{attr}}_{\text{graph feature}} + L^{\text{attr}}_{\text{class feature}} \tag{4.6}$$

where $L^{\text{attr}}_{\text{class feature}}$ indicates the attribute loss between the given graph features and the graph features in the generated graph and $L^{\text{attr}}_{\text{class feature}}$ indicates an attribute loss between the given class features and the class features in the generated graph.

Challenges

To solve our problem, we address two major challenges. The first challenge is that GenCAT must generate edges that satisfy multiple topological constraints[5] of the graph features and the class features.

By adopting latent factors that express the node-class membership/connection proportions, $L^{\text{topo}}_{\text{class feature}}$ is expressed with two parts: 1) the loss between a generated graph and latent factors and 2) the loss between latent factors and class features. We formulate $L^{\text{topo}}_{\text{class feature}}$ as follows:

$$\begin{aligned} L^{\text{topo}}_{\text{class feature}} = & \underbrace{L_{\text{edge precision}}}_{\text{between generated graph and latent factors}} \\ & + \underbrace{L_{\text{mean}} + L_{\text{deviation}} + L_{\text{class size}}}_{\text{between latent factors and class features}} \end{aligned} \tag{4.7}$$

where $L_{\text{edge precision}}$ is a loss that expresses the precision of the generated edges according to the probabilities of the edge existence calculated by the latent factors, L_{mean} is a loss between user-specified class preference mean and the class preference mean estimated from the latent factors, $L_{\text{deviation}}$ is a loss between user-specified class preference deviation and the class preference deviation estimated from the latent factors, and $L_{\text{class size}}$ is a loss between the expected class size distribution and the class size distribution in the generated graph.

The second challenge is efficiently generating large-scale graphs. There are two problems; 1) deciding whether there exists a graph satisfying given node degrees is an NP-complete problem [13], 2) since GenCAT assumes

[5]The attribute part has fewer constraints such as the distribution of attribute values, so we focus on the topology part here.

that each node has its node-class membership/connection proportion, it is required to estimate the probabilities of the edge existence of all node pairs, resulting in the large cost of $O(yn^2)$. To overcome the first problem, we design an efficient algorithm that heuristically assigns degrees to nodes and generates edges that satisfy the node degrees so as to avoid the large cost of satisfaction problem of graph generation. As for the second problem, we take an efficient edge generation approach that utilizes an approximation in order to achieve a linear time algorithm to the number of edges.

4.3 GenCAT: attributed graph generator for controlling class structure

In this section, we explain the design of GenCAT. First of all, we introduce latent factors and design the loss for class features $L^{\text{topo}}_{\text{class feature}}$ consisting of $L_{\text{edge precision}}$, L_{mean}, $L_{\text{deviation}}$, and $L_{\text{class size}}$, shown in Eq. (4.7) by using the latent factors. Also, we design the loss with graph feature, $L^{\text{topo}}_{\text{graph feature}}$ and provide an overview of graph generation by GenCAT. Then, we design the losses with class and graph features regarding attributes so that an adjacency matrix and an attribute matrix share the user-controlled class structure in Section 4.3.1. As we described in Section 4.2, we provide two scenarios of graph generation. In the first scenario, users input graph features (node degree distribution and attribute value distribution) and class features (class preference mean, class preference deviation, class size distribution, and attribute-class correlation). In the second scenario, GenCAT extracts the features from an input graph. We explain the first scenario in Section 4.3.2 and then the second scenario in Section 4.3.3. Next, we analyze the time and space complexities of GenCAT in Section 4.3.4. Finally, we show how GenCAT simulates existing generators in Section 4.3.5. Table 4.2 lists the main symbols and their definitions for the following descriptions. GenCAT supports various input parameters as shown in Table 4.3. The most basic parameters are 1) the number of nodes, edges, attributes, and classes to generate, and 2) the distribution parameters for class sizes[6] and attribute values.

[6]As for the exponents, we choose typical values of real networks: $1 \leq \phi_C \leq 2$, where ϕ_C is the parameter for class size [67].

Table 4.2: Definition of main symbols.

Variable	Explanation
$\boldsymbol{A} \in \{0,1\}^{n \times n}$	adjacency matrix
$\boldsymbol{X} \in \mathbb{R}^{n \times d}$	attribute matrix
$\boldsymbol{C} \in \{1,\ldots,y\}^{n}$	class label
$\Omega_l \in \{1,\ldots,n\}^{*}$	set of nodes labeled with class l
$\boldsymbol{U} \in \mathbb{R}^{n \times y}$	node-class membership proportions
$\boldsymbol{U}' \in \mathbb{R}^{n \times y}$	node-class connection proportions
$\boldsymbol{V} \in \mathbb{R}^{d \times y}$	attribute-class proportions
$\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^{n}$	expected and actual node degree proportions
$\boldsymbol{\rho} \in \mathbb{R}_{+}^{y}$	class size distribution
$Z \subset \{1,\ldots,y\}^{*}$	candidate set for edge generation

4.3.1 Generating Model

The basic idea of GenCAT is to capture class structure by using intermediate data structures, called *latent factors*, and then to generate graphs from the latent factors (See Figure 4.2).

Latent factors

GenCAT generates output graphs with class labels by using three latent factors: node-class membership proportions $\boldsymbol{U} \in \mathbb{R}^{n \times y}$, node-class connection proportions $\boldsymbol{U}' \in \mathbb{R}^{n \times y}$, and attribute-class proportions $\boldsymbol{V} \in \mathbb{R}^{d \times y}$, that are core components for capturing class features in the real-world graphs.

GenCAT calculates adjacency matrix $\boldsymbol{A}$ according to the probabilities of edge existence between nodes by multiplying $\boldsymbol{U}$ and $\boldsymbol{U}'^{\top}$ and attribute matrix $\boldsymbol{X}$ by multiplying $\boldsymbol{U}$ and $\boldsymbol{V}^{\top}$. By sharing $\boldsymbol{U}$ in the generations of $\boldsymbol{A}$ and $\boldsymbol{X}$, GenCAT can inject the common characteristics of classes into both the adjacency and the attribute matrices. We describe the detailed designs of $\boldsymbol{U}$ and $\boldsymbol{U}'$ in Section 4.3.1 and $\boldsymbol{V}$ in Section 4.3.1.

We formally define three latent factors as follows:

Definition 4.3.1 (Node-class membership proportions $\boldsymbol{U}$) *$\boldsymbol{U}$ regards a class assignment projection from nodes to classes. $\boldsymbol{U}$ is a matrix whose size is $n \times y$ and each element is in [0,1]. An element at i-th row and j-th*

Table 4.3: Description of the graph generator parameters.

Input	Description
$n, m, d, y \in \mathbb{N}$	number of nodes, edges, attributes, classes
$\boldsymbol{M} \in \mathbb{R}^{y \times y}$	class preference mean
$\boldsymbol{D} \in \mathbb{R}^{y \times y}$	class preference deviation
$\boldsymbol{H} \in \mathbb{R}^{d \times y}$	attribute-class correlation
$\phi_C \in \mathbb{R}_+$	parameter for class size distribution
$\omega \in \mathbb{R}$	deviation of normal distribution for attributes
$r \in \mathbb{N}$	number of iterations for edge generation

column indicates that node i likely belongs to class j if the element is close to one.

Definition 4.3.2 (Node-class connection proportions $\boldsymbol{U}'$) *$\boldsymbol{U}'$ regards an edge connectivity projection from nodes to classes. $\boldsymbol{U}'$ is a matrix whose size is $n \times y$ and each element is [0,1]. An element at i-th row and j-th column indicates that node i likely has edges with nodes in class j if the element is close to one.*

Definition 4.3.3 (Attribute-class proportions $\boldsymbol{V}$) *$\boldsymbol{V}$ regards an attribute projection from attributes to classes. $\boldsymbol{V}$ is a matrix whose size is $d \times y$ and each element is [0,1]. An element at i-th row and j-th column indicates that attribute i is likely correlated with class j if the element is close to one.*

Loss from class feature regarding topology

To incorporate the class features into the edge generation of GenCAT, we design the loss, $L^{\text{topo}}_{\text{class feature}}$ (Eq. (4.7)), which is expressed with 1) the loss between generated graph and latent factors and 2) the loss between latent factors and class features as follows.

Between generated graph and latent factors. We design GenCAT to generate edges by using latent factors, $\boldsymbol{U}$ and $\boldsymbol{U}'$. According to the definition of $\boldsymbol{U}$ and $\boldsymbol{U}'$, we can calculate the edge probability between nodes by multiplying $\boldsymbol{U}$ and $\boldsymbol{U}'^{\top}$, which expresses a composition of two projections, class assignment from nodes to classes ($\boldsymbol{U}$) and edge connectivity from classes to

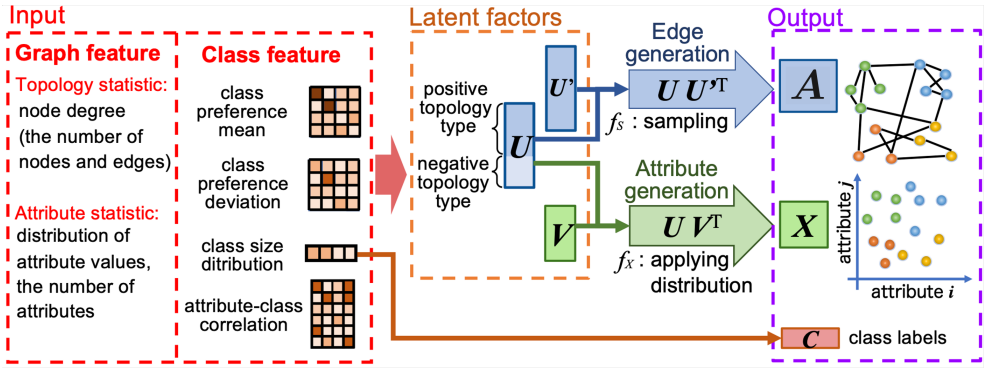


Figure 4.2: Illustration of our method. GenCAT first generates intermediate data structures $\boldsymbol{U}, \boldsymbol{U}'$, and $\boldsymbol{V}$ and then generates the adjacency matrix $\boldsymbol{A}$ and the attribute matrix $\boldsymbol{X}$ by using $\boldsymbol{U}, \boldsymbol{U}'$, and $\boldsymbol{V}$. The class labels $\boldsymbol{C}$ are generated based on class size distribution.

nodes ($\boldsymbol{U}'^{\top}$). By using the edge probabilities, we formulate the loss that expresses the precision of the generated edges as below:

$$L_{\text{edge precision}} = \|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{U}'^{\top}\|_F. \tag{4.8}$$

Recall the definition of homophily/heterophily in Section 4.1: homophily is a phenomenon where the nodes with similar attributes are more likely to connect to each other and heterophily is the inverse notion of homophily. By following them, we can design $\boldsymbol{U}$ and $\boldsymbol{U}'$ to have the same proportions for nodes (rows) with homophily property and to have the reverse proportions for nodes with heterophily property, respectively. To give precise definitions of classes with the homophily/heterophily properties, we introduce types of class. We assign *positive topology type* to a class with the homophily property, and *negative topology type* to a class with the heterophily property from input parameters as follows. Positive topology type is assigned to class l if the diagonal elements of the class preference mean are larger than a random connection proportion ($\boldsymbol{M}_{ll} \geq 1/y$), otherwise negative topology type is assigned. From the above discussion, we formulate $\boldsymbol{U}'$ of which nodes (rows) in positive topology type classes have the same values of $\boldsymbol{U}$ and nodes in negative topology type classes have the reversed values of $\boldsymbol{U}$ as below:

$$\boldsymbol{U}'_{i.} = \begin{cases} \boldsymbol{U}_{i.} & (i \in \Omega_l \text{ and } \boldsymbol{M}_{ll} \geq 1/y) \\ f_{\text{rev}}(\boldsymbol{U}_{i.}) & (i \in \Omega_l \text{ and } \boldsymbol{M}_{ll} < 1/y). \end{cases} \tag{4.9}$$

Also, the reverse function f_{rev} is formulated as follows:

$$f_{\text{rev}}(\boldsymbol{U}_{i.}) = \begin{cases} 1 - \boldsymbol{U}_{ih} & (h = l) \\ (1 - \boldsymbol{U}_{ih}) \frac{\boldsymbol{U}_{il}}{\sum_{j \neq l}^{y} (1 - \boldsymbol{U}_{ij})} & (h \neq l) \end{cases} \tag{4.10}$$

where node i is a node whose class is typed with negative topology and class l is a class which node i belongs to. Also, by regularizing the values other than in the class each node belongs to, the sum of each row of $\boldsymbol{U}'$ equals to one.

Notice that the edge generation by $\boldsymbol{U}\boldsymbol{U}'^{\top}$ is a more generalized form of $\boldsymbol{U}\boldsymbol{U}^{\top}$ used by SymNMF [63, 64], a well known technique for graph clustering. SymNMF is a special case when all classes have homophily property, so node-class membership proportions $\boldsymbol{U}$ are identical with node-class connection proportions $\boldsymbol{U}'$.

Between latent factors and class features. Next, we design loss formulas between latent factors and class features so that the latent factors precisely capture the class features, which are class preference mean, class preference deviation, and class size distribution. First, to reduce the loss between user-specified class preference mean and the class preference mean estimated by latent factors which is calculated by $\boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.}$ as an approximation, we formulate the loss as follows:

$$L_{\text{mean}} = \sum_{l}^{y} \| \boldsymbol{M}_{l.} - \underbrace{\frac{1}{|\Omega_l|} \sum_{i \in \Omega_l} \boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.}}_{\text{estimated connection proportions}} \|_F \tag{4.11}$$

where $*$ denotes the element-wise product. The reason that we adopt the approximation to calculate the estimated class preference mean is that the cost to obtain the exact proportion of edges between classes is the large cost of $O(kn^2)$.[7] Next, to reduce the loss between user-specified class preference deviation and the estimated deviation, we also formulate the loss as follows:

$$L_{\text{deviation}} = \sum_{l}^{y} \| \boldsymbol{D}_{l.} - \underbrace{\sqrt{\frac{1}{|\Omega_l|} \sum_{i \in \Omega_l} (\boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.} - \overline{\boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.}})^2}}_{\text{estimated deviation}} \|_F \tag{4.12}$$

[7] The cost comes from $\boldsymbol{U}\boldsymbol{U}'^{\top}$ which calculates all possible combinations of nodes.

where $\overline{\boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.}}$ denotes the average of $\boldsymbol{U}_{i.} * \boldsymbol{U}'_{i.}$ where $i \in \Omega_l$. We adopt an approximation due to the same reason as Eq. (4.11). Finally, we formulate the loss between the expected class sizes and the class sizes in a generated graph. The loss, $L_{\text{class size}}$, is described as:

$$L_{\text{class size}} = \sum_{l=1}^{y} (\boldsymbol{\rho}[l] - |\Omega_l|/n)^2 \tag{4.13}$$

where $\boldsymbol{\rho}$ denotes the class size distribution specified by users and $|\Omega_l|/n$ represents the class size proportion of class l in a generated graph.

Loss from graph feature regarding topology

The graph feature is expressed with the topology statistic and the attribute statistic and we focus on the topology part here. Because graph generators should support user-specified node degrees, the quality of a generated graph is computed by the difference between user-specified node degrees and the node degrees of a generated graph as follows:

$$L^{\text{topo}}_{\text{graph feature}} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{\boldsymbol{\theta}_i - \boldsymbol{\theta}'_i}{\boldsymbol{\theta}_i} \right| \tag{4.14}$$

where $\boldsymbol{\theta}$ is the expected node degree and $\boldsymbol{\theta}'$ is the actual node degrees. We employ *Mean Absolute Percentage Error* (MAPE) for the loss. The reason that we utilize MAPE is that it can treat high degree nodes and low degree nodes equally. Other measures can be used, such as mean squared error.

Edge generation

Adjacency matrix $\boldsymbol{A}$ is generated by $\boldsymbol{U}\boldsymbol{U}'^{\top}$ (See Eq. (4.8)). Thanks to the latent factors, $\boldsymbol{U}$ and $\boldsymbol{U}'$, GenCAT can take into account user-specified class features in the edge generation. As we mentioned in Section 4.2.3, the adjacency matrix generation has two problems; 1) deciding whether there exists a graph having given node degrees is an NP-complete problem, 2) the computational cost for the probability of edge existence, $\boldsymbol{U}\boldsymbol{U}'^{\top}$, is $O(yn^2)$. Moreover, these problems are correlated by the dependency between node degrees (Eq. (4.14)) and edges generated based on the node-class membership/connection proportions (Eq. (4.8)) because the degree of node i represents the number of edges associated with node i. Hence, GenCAT adopts

an efficient algorithm that heuristically assigns degrees to nodes and generates edges that satisfy the node degrees so as to avoid the large cost of the satisfaction problem of graph generation. Also, to overcome the large cost to compute the edge probabilities, we incorporate an inverse transform sampling [27] into our heuristic approach.

Approach to generate edges. The ideas of the approach are two-fold: 1) we generate edges in order starting from high degree nodes and 2) we accelerate the calculation of the probability of edge existence by utilizing an inverse transform sampling. The purpose of the first idea is that we avoid leaving high degree nodes to the later phase of edge generation so that high degree nodes can have a sufficient number of candidates to connect. If high degree nodes are left to the later phase of generation, most edges which should be associated with nodes may not be generated. Hence, we start the edge generation in the order starting from high degree nodes so that they have many candidates to connect with at the earlier phase of the edge generation. As for the second idea, we interpret the probability of the edge existence between node i and node j as computed by $\boldsymbol{U}_{i.} * \boldsymbol{U}'_{j.}$ in two selection steps, *Class selection from source node* step based on $\boldsymbol{U}$ and *Target node selection from class* step based on $\boldsymbol{U'}^{\top}$[8]. Thanks to the interpretation that transforms the calculation into the two probabilistic selection steps, we can incorporate inverse transform sampling into the approach. First, we describe the two selection steps in detail, and then we explain how to utilize the sampling method.

In *Class selection from source node* step, we choose classes $Z \subset \{1, \ldots, y\}^{\boldsymbol{\theta}_i}$ from each node i's node-class membership proportions $\boldsymbol{U}_{i.}$. Then, in *Target node selection from class* step, a node is selected from the node-class connection proportions $\boldsymbol{U}'^{\top}_{l}$ for each class $l \in Z$. An edge between the selected node and node i is generated[9]. This approach still suffers from the high computational cost to execute *Target node selection from class* for each source node, which requires $O(yn^2)$.

Inverse transform sampling. To accelerate the approach, we incorporate an inverse transform sampling for monotone densities into the approach. It can considerably improve the speed of edge generation since it realizes that the time complexity of *Target node selection from class* step is $O(1)$. First, we

[8]We use source and target even if we assume undirected graphs

[9]The edge probability between node i and node j is calculated by $\sum_{l=1}^{y} \boldsymbol{U}_{il}\boldsymbol{U}_{jl}$ which corresponds to $(\boldsymbol{U}\boldsymbol{U'}^{\top})_{ij}$.

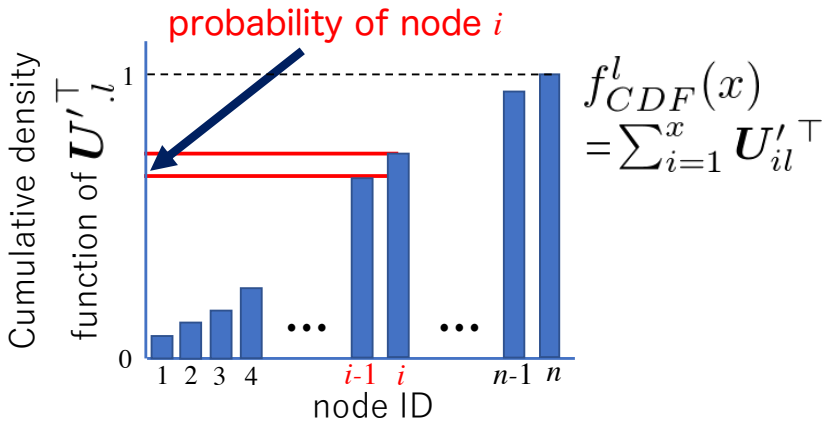


Figure 4.3: The concept of inverse transform sampling. Bars represent the cumulative density function of $\boldsymbol{U'}_{.l}^{\top}$ where l indicates a class.

calculate the cumulative distribution function (CDF) of $\boldsymbol{U'}^{\top}$ for each class. Figure 4.3 depicts the cumulative density function of $\boldsymbol{U'}$ by regularizing $\boldsymbol{U'}$ for a certain column. The horizontal axis indicates nodes, the vertical axis indicates the CDF of a column (class) of $\boldsymbol{U'}$, and the function f_{CDF} represents the CDF of the probability that nodes are selected from the class. The width between node $i-1$ and node i indicates the probability of selecting node i, which is expressed with $f_{CDF}(i) - f_{CDF}(i-1)$. We generate the value of inverse CDF, $f_{CDF}^{-1}(x)$, which is a node ID of node i if $f_{CDF}(i-1) \leq x < f_{CDF}(i)$. This means that the inverse CDF returns a node ID based on the probability by choosing a random number from zero to one. For the quick access, we generate a list by which we can obtain a node ID based on its selecting probability, in a similar way as the inverse CDF, by dividing the range of random numbers into small steps. That is, a node ID of node i is obtained by utilizing the list if the list receives the value between $f_{CDF}(i-1)$ and $f_{CDF}(i)$. Using this list, *Target node selection from class* is executed in $O(1)$. Note that the complexity to generate the list is $O(yn)$ because we compute the CDF for each class.

Loss regarding attribute

As we mentioned in Section 4.2, GenCAT should support both 1) the class feature that indicates the attribute-class correlation and 2) the graph feature for attributes, which is the distribution of attribute values.

First, we incorporate the node-class membership proportions $\boldsymbol{U}$ into attribute generation so that GenCAT generates graphs with the class structure shared by the topology and the attributes. To share the class structure with the topology, an attribute matrix $\boldsymbol{X}$ is generated based on $\boldsymbol{U}\boldsymbol{V}^\top$, which expresses a composition of class assignment projection from nodes to classes ($\boldsymbol{U}$) and attribute projection from classes to attributes ($\boldsymbol{V}^\top$). In order to reduce the loss between user-specified attribute-class correlation and the estimated attribute-class correlation, we formulate the loss as follows:

$$L^{\text{attr}}_{\text{class feature}} = \sum_{l=1}^{y} \| \boldsymbol{H}_{.l}^\top - \underbrace{\frac{1}{|\Omega_l|} \sum_{i \in \Omega_l} (\boldsymbol{U}\boldsymbol{V}^\top)_{i.}}_{\text{estimated attribute-class correlation}} \|_F. \tag{4.15}$$

By designing attribute-class proportions $\boldsymbol{V}$ to reduce $L^{\text{attr}}_{\text{class feature}}$, generated attributes satisfy user-specified attribute-class correlation. Since node-class membership proportions are commonly used in edge generation and attribute generation, the generated attributes share the class structure with the topology.

Second, we design the loss $L^{\text{attr}}_{\text{graph feature}}$ so that the distributions of generated attribute values are similar to the user-specified distributions. We adopt the *Earth-Mover* (EM) distance[10] that is a widely used measurement of the distance between two distributions [10]. By using the EM distance, we formulate $L^{\text{attr}}_{\text{graph feature}}$ as follows:

$$L^{\text{attr}}_{\text{graph feature}} = \sum_{\delta}^{d} \inf_{\gamma \in \prod(\boldsymbol{X}_{.\delta}, p(\boldsymbol{X}_{.\delta}))} \mathbb{E}_{(\alpha,\beta)\sim\gamma}[|\alpha - \beta|] \tag{4.16}$$

where $\prod(\boldsymbol{X}_{i\delta}, p(\boldsymbol{X}_{i\delta}))$ denotes the set of all joint distributions $\gamma(\alpha, \beta)$ whose marginals are $\boldsymbol{X}_{i\delta}$ and $p(\boldsymbol{X}_{i\delta})$, respectively, and $p(\cdot)$ indicates user-specified distributions for the attributes. This loss indicates the difference between the distributions of generated attribute values and user-specified distributions $p(\cdot)$ (Bernoulli or normal distribution). For instance, if users specify a normal distribution for attributes, we formulate $p(\cdot)$ as follows:

$$p(\boldsymbol{X}_{i\delta}) = \mathcal{N}(\frac{\sum_{i=1}^{n} \boldsymbol{X}_{i\delta}}{n}, w^2) \tag{4.17}$$

where ω indicates the user-specified deviation of a normal distribution.

[10]Earth Mover distance is also known as Wasserstein-1

Attribute generation

In order to reduce $L^{\text{attr}}_{\text{class feature}}$ and $L^{\text{attr}}_{\text{graph feature}}$, first 1) we obtain base attribute vectors for nodes by computing the product of $\boldsymbol{U}, \boldsymbol{V}^\top$ so that two nodes in the same class (reflecting the effect of the topology and attributes) should share similar attribute values, and then 2) we apply user-specified distribution to the base attribute vectors so that the attribute values should follow the distribution.

Therefore, GenCAT can generate an adjacency matrix $\boldsymbol{A}$ and an attribute matrix $\boldsymbol{X}$ with user-controlled class structure. Also, GenCAT outputs class labels $\boldsymbol{C}$ of nodes, which are obtained based on class size distribution.

4.3.2 Algorithm

We next explain the attributed graph generation algorithms of GenCAT in detail. Algorithm 3 describes the whole procedure of graph generation. It consists of three phases, latent factor generation phase, adjusting proportion phase, and graph generation phase. First, in latent factor generation phase, we generate and initialize latent factors by graph/class features specified by users. Second, in adjusting proportion phase, we adjust the latent factors $\boldsymbol{U}$ and $\boldsymbol{U}'$ to minimize the loss L_{mean} (Eq. (4.11)) and $\boldsymbol{V}$ to minimize the loss $L^{\text{attr}}_{\text{class feature}}$ (Eq. (4.15)) for graph generation. Finally, in graph generation phase, we generate an adjacency matrix $\boldsymbol{A}$ and an attribute matrix $\boldsymbol{X}$ from the latent factors.

Latent factor generation

In the latent factors generation phase (lines 1–11), we first generate a class size distribution from a power law distribution controlled by input parameter ϕ_C (line 2)[11]. Next, we generate a class label $\boldsymbol{C}$ for each node based on the class size distribution and initialize $\boldsymbol{U}$ by a normal distribution whose average is $\boldsymbol{M}$ and deviation is $\boldsymbol{D}$ (lines 3–5). To support both positive and negative topology types, we initialize node-class connection proportions, $\boldsymbol{U}'$, by $\boldsymbol{U}$ and then reverse the node-class membership proportions of nodes which are in classes typed with negative topology (lines 6–10). Note that this reverse realizes that each node has the large value in the node-class

[11] GenCAT allows users to adopt a normal distribution and to directly input class size distribution $\boldsymbol{\rho}$.

Algorithm 3: Graph generation

input : $n, m, d, y, \boldsymbol{M}, \boldsymbol{D}, \boldsymbol{H}, \phi_C, \omega, r, f_S, f_X$
output: adjacency matrix $\boldsymbol{A}$, attribute matrix $\boldsymbol{X}$, class label $\boldsymbol{C}$

Latent factors generation phase
$\boldsymbol{\rho} = power\ law(y, \phi_C)$
for $i = 1$ *to* n **do**
  $\boldsymbol{C}_i = Rand_{int}(\text{range}=(1, y), \text{weight}=\boldsymbol{\rho})$
  $\boldsymbol{U}_{i.} = normal(\boldsymbol{M}_{\boldsymbol{C}_i.}, \boldsymbol{D}_{\boldsymbol{C}_i.})$
$\boldsymbol{U}' = \boldsymbol{U}$
for $l = 1$ *to* y **do**
  if $\boldsymbol{M}_{ll} < \frac{1}{y}$ **then**
    for $i \in \Omega_l$ **do**
      $\boldsymbol{U}_{i.} = f_{\text{rev}}(\boldsymbol{U}_{i.})$ (Eq. (4.10)) ▷ negative topology type
$\boldsymbol{V} = \boldsymbol{H}$
Adjusting proportion phase
for $l = 1$ *to* y **do**
  if $\boldsymbol{M}_{ll} \geq \frac{1}{y}$ **then**
    $T_{min} = \underset{T}{\text{argmin}}(\|\boldsymbol{M}_{l.} - \frac{1}{|\Omega_l|}\sum_{i\in\Omega_l} f_{\text{AP}}(\boldsymbol{U}_{i.}, T) * f_{\text{AP}}(\boldsymbol{U}_{i.}, T)\|_F)$
      ▷ minimize L_{mean} (Eq. (4.11)) by using f_{AP} (Eq. (4.18))
    for $i \in \Omega_l$ **do**
      $\boldsymbol{U}_{i.} = f_{\text{AP}}(\boldsymbol{U}_{i.}, T_{min})$
      $\boldsymbol{U}'_{i.} = \boldsymbol{U}_{i.}$
  else
    $T_{min} = \underset{T}{\text{argmin}}(\|\boldsymbol{M}_{l.} - \frac{1}{|\Omega_l|}\sum_{i\in\Omega_l} f_{\text{AP}}(\boldsymbol{U}_{i.}, T) * f_{\text{rev}}(f_{\text{AP}}(\boldsymbol{U}_{i.}, T))\|_F)$
      ▷ minimize L_{mean} by using f_{rev} (Eq. 4.9) and f_{AP}
    for $i \in \Omega_l$ **do**
      $\boldsymbol{U}_{i.} = f_{\text{AP}}(\boldsymbol{U}_{i.}, T_{min})$
      $\boldsymbol{U}'_{i.} = f_{\text{rev}}(\boldsymbol{U}_{i.})$
for $\delta = 1$ *to* d **do**
  $T_{min} = \underset{T}{\text{argmin}}(\|\boldsymbol{H}_{\delta.} - \frac{1}{|\Omega_l|}\sum_{i\in\Omega_l} \boldsymbol{U}_{i.} * f_{\text{AP}}(\boldsymbol{V}_{\delta.}, T)\|_F)$
    ▷ minimize $L^{\text{attr}}_{\text{class feature}}$ (Eq. (4.15)) by using f_{AP}
  $\boldsymbol{V}_{\delta.} = f_{\text{AP}}(\boldsymbol{V}_{\delta.}, T_{min})$
Graph generation phase
$\boldsymbol{A} = f_S(\boldsymbol{U}, \boldsymbol{U}', n, m, y, r)$ ▷ f_S = Edge_generation
$\boldsymbol{X} = f_X(\boldsymbol{U}, \boldsymbol{V}, \omega)$
return $\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{C}$

membership proportions for a class that the node belongs to according to Eq. (4.10). Then, we initialize attribute-class proportions, $\boldsymbol{V}$, by $\boldsymbol{H}$ so that the attribute-class proportions reflect user-specified correlation between the attributes and classes (line 11).

Adjusting proportion phase

The goal of this phase is to rescale the node-class membership proportions and attribute-class proportions to reduce the losses imposed by the inputs of class preference mean and attribute-class correlation, respectively. The reason that we need to adjust these proportions is that the initialization of the proportions is not designed to minimize the losses (Eq. (4.11) and (4.15)) since the initialization aims to capture the tendency of user-specified statistics.

First, we adjust the node-class membership proportions $\boldsymbol{U}$ for each class to minimize the loss L_{mean} shown in Eq. (4.11) (lines 13–23). The procedure adjusting $\boldsymbol{U}$ depends on whether a class is typed with positive topology (lines 14–18) or negative topology (lines 19–23). As for the positive topology type, we adopt a grid parameter search for T from 0 to 1 in 0.05 step, in order to minimize L_{mean} shown in Eq. (4.11) (line 15). To rescale the node-class membership proportions with probability values for the minimization, we utilize the same idea in [112] by adopting rescale function f_{AP} below:

$$f_{\text{AP}}(\boldsymbol{U}_{i\cdot}, T) = \boldsymbol{U}_{i\cdot}^{\frac{1}{T}} / \sum_{j}^{y} (\boldsymbol{U}_{ij}^{\frac{1}{T}}) \tag{4.18}$$

where i is a node ID and T is a parameter which controls the degree of rescale. Then, for each node i in the class l, we update $\boldsymbol{U}_{i\cdot}$ by using f_{AP} with T_{min} that is the output of the grid search (line 17) and update $\boldsymbol{U}'_{i\cdot}$ to the same value as $\boldsymbol{U}_{i\cdot}$ (line 18). As for the negative topology type, we use f_{rev} (Eq. (4.9)) to compute $\boldsymbol{U}'$ from $\boldsymbol{U}$ and minimize L_{mean} by utilizing a grid search similarly as positive topology type (line 20). Then, for each node i in the class l, we update $\boldsymbol{U}_{i\cdot}$ by using f_{AP} with T_{min} (line 22) and update $\boldsymbol{U}'_{i\cdot}$ by using f_{rev} since the class l is typed with negative topology (line 23). Note that we adjust $\boldsymbol{U}_{i\cdot} * \boldsymbol{U}'_{i\cdot}$ with $\boldsymbol{M}$ (See Eq. (4.11)) as an approximation since the cost to obtain the exact proportion of edges between classes is $O(yn^2)$[12].

[12] $O(yn^2)$ stems from the matrix multiplication $\boldsymbol{U}\boldsymbol{U}'^{\top}$ to consider all pairs of nodes.

Algorithm 4: Edge_generation($\boldsymbol{U}, \boldsymbol{U}', n, m, y, r$)

input : $\boldsymbol{U}, \boldsymbol{U}', n, m, y, r$
output: $\boldsymbol{A}$

1 $\phi_d^{min} = \underset{\phi_d}{\operatorname{argmin}} |m - \sum_{i=1}^{n} (\text{powerlaw}(n, \phi_d))_i / 2|$
2 $\boldsymbol{\theta} = \text{sort}(\text{powerlaw}(n, \phi_d^{min}), \text{descending order})$
3 $\boldsymbol{\theta}' = [0]^n$
4 **for** $i = 1$ *to* n **do**
5 $counter = 0$
6 **while** $counter < r$ *and* $\boldsymbol{\theta}'_i < \boldsymbol{\theta}_i$ **do**
7 **### Class selection from source node ###**
8 $Z = \{\}$
9 **for** $iter = 1$ *to* $\boldsymbol{\theta}_i - \boldsymbol{\theta}'_i$ **do**
10 $Z = Z \bigcup Rand_{int}(\text{range}= (1, y), \text{weight}= \boldsymbol{U}_{i.})$
11 **### Target node selection from class ###**
12 **for** $l \in Z$ **do**
13 $j = Rand_{int}(\text{range}= (1, n), \text{weight}= \boldsymbol{U'}_{.l}^{\top})$
14 **if** $\boldsymbol{A}_{ij} == 0$ *and* $\boldsymbol{\theta}'_i < \boldsymbol{\theta}_i$ *and* $\boldsymbol{\theta}'_j < \boldsymbol{\theta}_j$ **then**
15 $\boldsymbol{A}_{ij} = 1$
16 $\boldsymbol{A}_{ji} = \boldsymbol{A}_{ij}$ ▷ undirected graph
17 $(\boldsymbol{\theta}'_i, \boldsymbol{\theta}'_j) = (\boldsymbol{\theta}'_i + 1, \boldsymbol{\theta}'_j + 1)$ ▷ increment degrees
18 $counter = counter + 1$
19 **return** $\boldsymbol{A}$

Second, we adjust the attribute-class proportions $\boldsymbol{V}$ for each attribute to minimize the loss $L^{\text{attr}}_{\text{class feature}}$ shown in Eq. (4.15) (lines 25–26). We can obtain adjusted $\boldsymbol{V}$ by using a grid search similarly as $\boldsymbol{U}$.

Graph generation phase

In the graph generation phase, we generate the adjacency matrix $\boldsymbol{A}$ (line 28) and attribute matrix $\boldsymbol{X}$ (line 29) by using the adjusted latent factors.

Edge generation. Algorithm 4 describes how adjacency matrix $\boldsymbol{A}$ is generated. First, we adopt a grid parameter search for a power law parameter ϕ_d^{min} from 1 to 3 in 0.01 step, so as to reduce the loss between the number

of edges and the sum of the expected node degree proportions[13] (line 1). Then, $\boldsymbol{\theta}$ is generated by using a power law distribution with ϕ_d^{min}, and we sort it in descending order so that we start edge generation from high degree nodes as we mentioned in Section 4.3.1 (line 2). Let $\boldsymbol{\theta}'$ be the actual node degrees during the edge generation. It is initialized as zeros (line 3) and its entries $\boldsymbol{\theta}'_i$ and $\boldsymbol{\theta}'_j$ are incremented when a new edge (i, j) is generated (line 17). For each node i, edges are iteratively generated until $\boldsymbol{\theta}'_i$ gets close to the expected node degree proportion $\boldsymbol{\theta}_i$ by *Class selection from source node* step (lines 8–10) and *Target node selection from class* step (lines 12–17). Thanks to these steps, GenCAT generates edges according to the stochastic process specified by $\boldsymbol{U}$ and $\boldsymbol{U'}^\top$. In *Class selection from source node* step, classes are chosen from the node-class membership proportions of source node i, $\boldsymbol{U}_{i.}$ (line 10). *Target node selection from class* step is iterated for the classes selected in the previous step (line 12). We select target node j from the node-class connection proportions $\boldsymbol{U'}^\top$ (line 13).

In order to accelerate *Target node selection from class* step, we generate a list *to_node_prob* by which we can obtain a node ID based on the probability of selecting the node, in a similar way as the inverse CDF of $\boldsymbol{U'}^\top$. Setting a step size w, which balances memory size and the accuracy of the selection, *to_node_prob* is formulated below:

$$to\_node\_prob[c] = f_{CDF}^{-1}(w \cdot c) \tag{4.19}$$

where $c \in \mathbb{N}$ and $w * c \leq 1$. The length of *to_node_prob* is $1/w$. We set c by the following equation[14]:

$$c = \lceil Rand(range(0, 1))/w \rceil \tag{4.20}$$

We set $w = 1/(100n)$ so as to vary with n since there is a trade-off between memory size[15] and the accuracy of the selection.

If neither the actual node degrees $\boldsymbol{\theta}'_i$ or $\boldsymbol{\theta}'_j$ reach the expected node degrees $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$, respectively[16], we generate an edge between them and then

[13]Users can input an arbitrary node degree distribution. In our algorithm, we show a case of using a power law distribution that node degrees in real-world graphs often follow.

[14]If node IDs are numbered from 0 to $n-1$, we just replace the ceiling function to the floor function.

[15]*to_node_prob* needs the space complexity, $O(yn)$, which is one of the largest elements (The details are described in Section 4.3.4).

[16]If the actual node degrees reach the expected node degrees, that means the node has enough number of edges.

increment their node degrees (lines 14–17). Note there is a possibility that the edge generation loop (lines 6–18) does not stop because there is such a case that the last node of the loop cannot have its expected degree. To avoid this, we exit the loop at the user-specified r iterations (line 6)[17]. It is our future work to guarantee the theoretical quality bounds of the generated graphs.

Attribute generation. Attribute matrix $\boldsymbol{X}$ is generated by $f_X(\boldsymbol{U}, \boldsymbol{V}, \omega)$ (line 29 in Algorithm 3). As we described in Section 4.3.1, we obtain base attribute vectors by multiplying $\boldsymbol{U}\boldsymbol{V}^\top$ so that nodes in the same class should share similar attribute values, and then we apply user-specified distribution to the base attribute vectors so that the attribute values should follow the distribution in order to reduce the loss $L^{\text{attr}}_{\text{graph feature}}$ shown in Eq. (4.16). As for the application of user-specified distributions, GenCAT supports two types of distributions, normal distribution and Bernoulli distribution. First, if users specify a normal distribution for attributes, we calculate $\boldsymbol{X}_{i\delta} = (\boldsymbol{U}\boldsymbol{V}^\top)_{i\delta} + \mathcal{N}(0, \omega^2)$. We normalize all attribute values to [0,1] without the loss of generality. Second, if users specify the Bernoulli distribution for attributes, we calculate $\boldsymbol{X}_{i\delta} = \mathcal{B}((\boldsymbol{U}\boldsymbol{V}^\top)_{i\delta})$, where $\mathcal{B}(x)$ is a function which returns 1 with probability x or 0 with probability $1 - x$. It is our future work to support other distributions for the attributes such as power-law and categorical distributions.

4.3.3 Parameter Extraction from Given Graph Dataset

Recall the second scenario that we described in Section 4.2: we extract parameters from input graphs with class labels and reproduce graphs similar to the input graphs. GenCAT uses a parameter extracting function that obtains topology statistic and class features of given graphs; node degrees, class preference mean, class preference deviation, and class sizes. Finally, we construct graphs in the same way as the first scenario.

Thanks to the parameter extracting function, GenCAT easily generates graphs similar to those that users input. Additionally, it enables generating graphs in arbitrary size with similar class features of the given graphs by permitting users to change the numbers of nodes and edges.

[17]Although we adjust the sum of all the node degrees in $\boldsymbol{\theta}$ to be the number of edges m, some candidate edges may not be generated when the node degrees of the adjacent nodes exceeds the expected ones, so the actual number of the generated edges tends to be smaller than the expected number of edges, m.

4.3.4 Complexity

We discuss the time/space complexities of GenCAT. As is typical in network analytics, we focus on sparse graphs [86] as real-world graphs are often sparse. On the sparse condition, the mean of the node degree, θ_{Avg}, can be treated as a constant. As the result, $m \propto n$ holds. Hence, m is considered to be a much smaller value than n^2.

Time complexity.

We analyze the time complexity of the latent factor generation, the edge generation, and the attribute generation, respectively. First, the complexity for generating the node-class membership/connection proportions and the attribute-class proportions is $O(yn + dy)$ based on their matrix sizes. Second, adjusting proportion phase consists of two parts, adjusting the node-class membership/connection proportions and adjusting the attribute-class proportions. The former phase requires $O(yn)$ to adjust them. The latter phase requires $O(dyn)$ to calculate $\boldsymbol{U}\boldsymbol{V}^\top$. In practical cases, we can rewrite $\frac{1}{|\Omega_l|}\sum_{i\in\Omega_l}\boldsymbol{U}\boldsymbol{V}^\top$ as $\boldsymbol{P}\boldsymbol{V}^\top$, where $\boldsymbol{P} \in \mathbb{R}^{y\times y}$ and $\boldsymbol{P}_{l\cdot} = \frac{1}{|\Omega_l|}\sum_{i\in\Omega_l}\boldsymbol{U}_{i\cdot}$. The computational cost of this phase is $O(yn + dy^2)$ owing to this transformation. Note that y is typically much smaller than n.

The edge generation consists of the *Class selection from source node* step and the *Target node selection from class* step. In the former step, classes are chosen based on the node-class membership proportions $\boldsymbol{U}$ for each node. The cost is $O(yn\theta_{Avg})$ since we select as many classes as the remained node degrees. In the latter step, we select a node for each class selected in the former step, based on the transpose of the topology proportions $\boldsymbol{U}'$. We generate list *to_node_prob* to accelerate this selection. The cost of this step is $O(1)$ since this step only includes the operation of a random value generation and list access. These two steps require $O(myr)$ since $m = n\theta_{Avg}$ and r is a constant number of iterations for the edge generation. Finally, in the attribute generation, the matrix multiplication $\boldsymbol{U}\boldsymbol{V}^\top$ requires $O(dyn)$ and the application of user-specified distribution requires $O(dn)$. Hence, the complexity of the attribute generation is $O(dyn)$. Therefore, the total time complexity is $O(myr + dyn)$.

Space complexity.

The largest concern is the adjacency matrix $\boldsymbol{A}$ since the size of $\boldsymbol{A}$ could be as large as n^2. Hence, we use sparse representation for $\boldsymbol{A}$, such as an adjacency list, with size $O(m)$. The sizes of the latent factors $\boldsymbol{U}, \boldsymbol{U}'$, and $\boldsymbol{V}$ are $O(yn), O(yn)$, and $O(dy)$, respectively. To utilize inverse transform sampling, we construct lists whose size is $O(yn)$. The size of an attribute matrix $\boldsymbol{X}$ is $O(dn)$. Therefore, the space complexity is $O(m + yn + dn)$ since y is smaller than n.

4.3.5 Simulating Existing Generators

We show that GenCAT is a general generator of existing methods, LFR and DC-SBM. First, LFR specifies the edge fraction of intra- and inter-edges by a mixing parameter. So, GenCAT can simulate LFR by adding three conditions that 1) nodes in a class have the same node-class membership proportions: $\boldsymbol{U}_{i\cdot} = \boldsymbol{U}_{j\cdot}$ if $\boldsymbol{C}_i = \boldsymbol{C}_j$, 2) the connection proportions between each class and other classes are the same: $\boldsymbol{U}_{ih_1} = \boldsymbol{U}_{ih_2}$ if $h_1, h_2 \neq \boldsymbol{C}_i$, and 3) all classes have the same edge fraction of intra- and inter-edges: $\boldsymbol{U}_{i\boldsymbol{C}_i} = \boldsymbol{U}_{j\boldsymbol{C}_j}$ for all pairs of nodes (i, j). To realize the first condition, we can set all elements of $\boldsymbol{D}$ to zero so that every node in a class has the same node-class membership proportion. As for the second condition, a given class preference mean should satisfy $\boldsymbol{M}_{lh_1} = \boldsymbol{M}_{lh_2}$ if $h_1, h_2 \neq l$ for each class l. For the third condition, a given class preference mean should also satisfy $\boldsymbol{M}_{ll} = \mu$ where $1 \leq l \leq y$ and μ is a mixing parameter of LFR.

Next, DC-SBM specifies the connection proportions between classes. So, the edge probabilities of GenCAT and DC-SBM correspond by adding a condition that nodes in a class have the same node-class membership proportions: $\boldsymbol{U}_{i\cdot} = \boldsymbol{U}_{j\cdot}$ if $\boldsymbol{C}_i = \boldsymbol{C}_j$. To realize the condition, we can set all elements of $\boldsymbol{D}$ to zero in addition to the first condition of LFR. As for graph features, GenCAT can utilize the same distributions as the existing generators. Hence, GenCAT can simulate the existing generators in terms of both graph/class features.

4.4 Validation of Effectiveness and Efficiency of GenCAT

We next describe an experimental study of GenCAT. Our goal in the experiments is to answer the following questions:

Q1 Does GenCAT support users to flexibly control graph/class features regarding **topology**? (Sec. 4.4.1)

Q2 Does GenCAT support users to flexibly control graph/class features regarding **attributes**? (Sec. 4.4.2)

Q3 How well does GenCAT scale? (Sec. 4.4.3)

Q4 How precisely does GenCAT reproduce real-world graphs? (Sec. 4.4.4)

Q5 How efficient and effective are the proposed techniques on edge generation? (Sec. 4.4.5)

In Q1 and Q2, we validate that GenCAT can output attributed graphs following graph/class features specified by given parameters. In Q3 and Q4, we evaluate the performance of scalability and reproducibility of GenCAT compared with existing graph generators. In Q5, we demonstrate the efficiency and effectiveness of the proposed techniques by conducting an ablation study. We use LFR[18] and DC-SBM[19] as the main competitors because LFR and DC-SBM generate graphs with class labels and controlled class-level connection proportions, which are closest to the capabilities of GenCAT in terms of generated topology structure. Since schema-driven (e.g., pgMark and TrillionG) and clustering method-driven (e.g., ANC) approaches generate quite different graphs, it is hard to fairly compare them with GenCAT, so we do not compare them with GenCAT. For evaluating scalability, we compare GenCAT with LFR and DC-SBM. For evaluating reproducibility, we

[18] `https://networkx.github.io/documentation/networkx-2.1/reference/algorithms/generated/networkx.algorithms.community.community_generators.LFR_benchmark_graph.html`

[19] Since the source code of SBM is available, we extend SBM to DC-SBM that incorporates node degrees into its edge generation algorithm. Note that we make minor changes to the code of SBM for the extension, but the cost of time and space for SBM is the same as for DC-SBM. `https://networkx.org/documentation/stable/reference/generated/networkx.generators.community.stochastic_block_model.html`

Table 4.4: MAPE between the expected node degree θ and the actual node degree θ'.

m	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Error	1.03e–3	7.37e–4	5.76e–4	5.25e–4	5.00e–4

compare GenCAT with LFR, DC-SBM, VGAE[20], and NetGAN[21]. VGAE is a baseline method of network embedding approaches. NetGAN is the-state-of-the-art deep learning-based graph generator. We do not compare with GraphVAE as it can only be used on very small graphs.

GenCAT is implemented in Python3. The experiments are operated on Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz with 1TB memory. All experiments are operated on a single thread and a single core.

4.4.1 Evaluation of Graph/Class Features Regarding Topology (Q1)

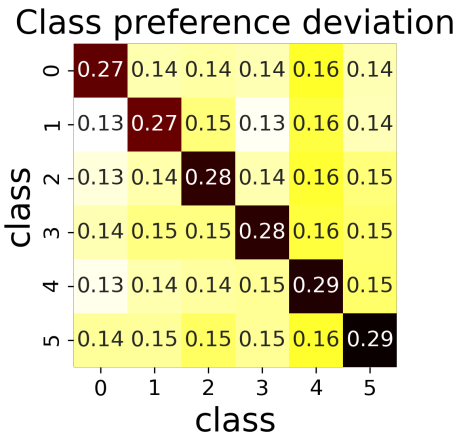
We here validate that GenCAT generates graphs following the users' given graph/class features regarding the topology.

Graph feature regarding topology

We show that GenCAT can generate graphs that almost follow given node degrees by calculating the loss of graph features, the difference between the expected node degree $\boldsymbol{\theta}$ and the actual node degrees $\boldsymbol{\theta}'$ by Eq. (4.14). We vary the number of edges m within the range of $\{2^{16}, 2^{17}, 2^{18}, 2^{19}, 2^{20}\}$. We set the parameters y, r, and n as $5, 50$, and $m/16$, respectively, all the diagonal elements of $\boldsymbol{M}$ to 0.4 and the other elements to 0.15. We generate five graphs and report the average of their results. Table 4.4 shows the MAPE of node degrees. This result shows that the graphs generated by GenCAT almost follows given node degrees.

[20] `https://github.com/zfjsail/gae-pytorch`
[21] `https://github.com/danielzuegner/netgan`

(a) Each cell represents the class preference mean of the generated graph, $\boldsymbol{M}^{gen}$. The diagonal elements of $\boldsymbol{M}$ are set to 0.6 and other elements are set to 0.08.

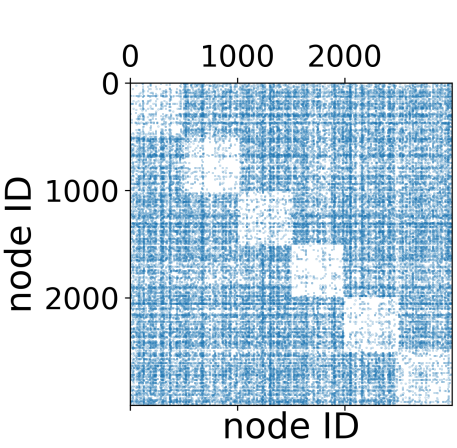


(b) Each cell represents the class preference deviation of the generated graph, $\boldsymbol{D}^{gen}$. The diagonal elements of $\boldsymbol{D}$ are set to $[0.2, 0.2, 0.25, 0.25, 0.3, 0.3]$ and other elements are set to 0.05.

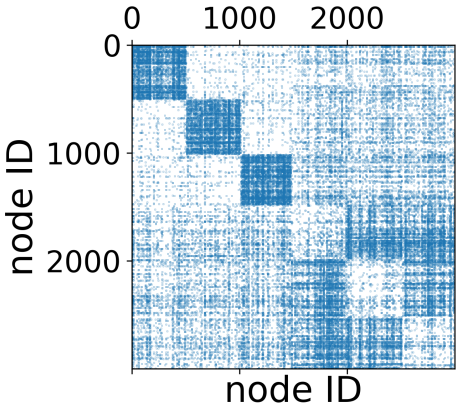
Figure 4.4: Visualization of class preference mean and class preference deviation. The parameters are set as follows: $n = 2^{16}$, $m = 2^{20}$, and $y = 6$.

Class feature regarding topology

We demonstrate that GenCAT flexibly controls class features in generated graphs. Figure 4.4 shows the heatmaps of the class preference mean and the class preference deviation of the generated graph. In this experiment, We set all the diagonal elements of $\boldsymbol{M}$ to 0.6. Also, we set the diagonal elements of $\boldsymbol{D}$ to $[0.2, 0.2, 0.25, 0.25, 0.3, 0.3]$, respectively, and the other elements to 0.05. Let $\boldsymbol{M}^{gen}$ and $\boldsymbol{D}^{gen}$ be a class preference mean and a class preference deviation of the generated graph. First, we demonstrate that GenCAT can control the class-level connection proportions. Figure 4.4a shows that the diagonal elements of $\boldsymbol{M}^{gen}$ are almost 0.6. The reason that the diagonal elements of $\boldsymbol{M}^{gen}$ do not completely match 0.6 is that the edge generation step is based on the probabilistic procedures. This figure validates that GenCAT supports classes typed with positive topology. Figure 4.4b shows that GenCAT can control the deviation of the connection proportions between nodes and classes. There is a tendency that $\boldsymbol{D}^{gen}_{00}$ and $\boldsymbol{D}^{gen}_{11}$ are smaller than $\boldsymbol{D}^{gen}_{44}$



(a) Negative topology classes. All diagonal elements of $\boldsymbol{M}$ are set to 0.05.



(b) Coexistence of positive and negative topology classes. The diagonal elements of $\boldsymbol{M}$ of class $0, 1, 2$ are set to 0.9 and the ones of class $3, 4, 5$ are set to 0.05.

Figure 4.5: Visualization of adjacency matrices. The parameters are set as follows: $n = 3000$, $m = 30000$, and $y = 6$.

and $\boldsymbol{D}^{gen}_{55}$. The reason that the diagonal elements of $\boldsymbol{D}^{gen}$ do not completely match the values we set is that GenCAT prioritizes other constraints higher than estimating class preference deviation in the edge generation, such as the node degrees and the class preference mean. This result indicates that GenCAT supports the deviation of the connection proportions, albeit at a lower priority. In summary, the experiments validate that GenCAT supports the connection proportions between nodes and classes by using the class preference mean and the class preference deviation. We compare GenCAT with other generators in Section 4.4.4.

Next, we also validate that GenCAT supports various types of class labels with negative and mixed topology types in graphs, respectively. Figure 4.5 shows the adjacency matrices of the generated graphs. Blue dots indicate that there exist edges between nodes whose identifiers are the node IDs on vertical and horizontal axes. In this experiment, we set the number of classes to six, and we can observe the classes as blocks in the diagonal part of the figure. Figure 4.5a additionally shows the case that all classes are negative types. We set all diagonal elements of $\boldsymbol{M}$ to 0.05, lower than the average. The nodes clearly tend to connect with nodes from other classes. We demonstrate

Table 4.5: Comparison of the distributions of the generated attributes and the matrix multiplication $\boldsymbol{U}\boldsymbol{V}^\top$ by using the EM distance. $D_{\mathrm{EM}}(\boldsymbol{X}, p(\boldsymbol{X}))$ denotes the sum of the EM distance between $\boldsymbol{X}$ and $p(\boldsymbol{X})$ for all attributes, as shown in Eq. (4.16). $\boldsymbol{H}^{(1)}$, $\boldsymbol{H}^{(2)}$, and $\boldsymbol{H}^{(3)}$ are the variations of given attribute-class correlations.

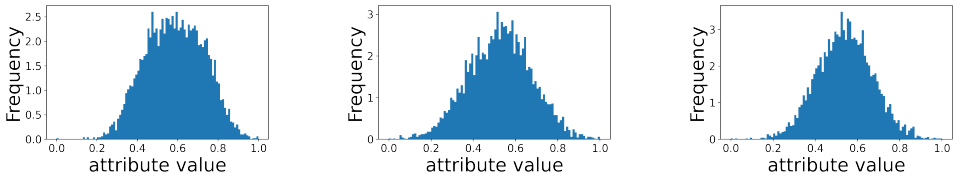
EM distance	$\boldsymbol{H}^{(1)}$	$\boldsymbol{H}^{(2)}$	$\boldsymbol{H}^{(3)}$
$D_{\mathrm{EM}}(\boldsymbol{X}, p(\boldsymbol{X}))$	0.049	0.089	0.108
$D_{\mathrm{EM}}(\boldsymbol{U}\boldsymbol{V}^\top, p(\boldsymbol{X}))$	0.473	0.492	0.481

a more complicated case that both positive and negative topology types in a single graph, in Figure 4.5b. Nodes from class $0, 1, 2$ are densely connected inside since the types of their classes are positive topology. Nodes from class $3, 4, 5$ tend to connect with nodes from other classes. In summary, we confirm that GenCAT flexibly controls the connection proportions between nodes and classes.

4.4.2 Evaluation of Graph/Class Features Regarding Attributes (Q2)

We demonstrate that GenCAT can generate node attributes following the users' given graph/class features regarding the attributes. In this experiment, we set the parameters d, y, and n as $2, 4$, and 5000, respectively[22], all the diagonal elements of $\boldsymbol{M}$ to 0.7, the other elements to 0.1, the diagonal elements of $\boldsymbol{D}$ to $[0.2, 0.2, 0.25, 0.25]$, respectively, and the other elements of $\boldsymbol{D}$ to 0.1. We set a normal distribution as the distribution of the attribute and its deviation ω to 0.2. We vary attribute-class correlations $\boldsymbol{H}$ in three patterns of separation degrees between classes: $\boldsymbol{H}^{(1)} = [[0.5, 0.0, 0.0, 0.5], [0.0, 0.5, 0.0, 0.5]]$, $\boldsymbol{H}^{(2)} = [[0.4, 0.1, 0.1, 0.4], [0.1, 0.4, 0.1, 0.4]]$, and $\boldsymbol{H}^{(3)} = [[0.3, 0.2, 0.2, 0.3], [0.2, 0.3, 0.2, 0.3]]$.

[22]To make visualization easier to understand, we set the number of attributes d to 2. Note that GenCAT can efficiently generate more attributes since its time and space complexities are linear to d (see Section 4.3.4)

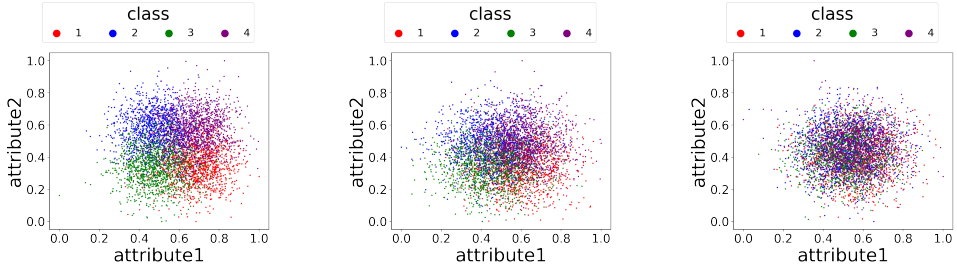


(a) $\boldsymbol{H}_{1.}^{(1)} = [0.5, 0.0, 0.0, 0.5]$. (b) $\boldsymbol{H}_{1.}^{(2)} = [0.4, 0.1, 0.1, 0.4]$. (c) $\boldsymbol{H}_{1.}^{(3)} = [0.3, 0.2, 0.2, 0.3]$.

Figure 4.6: Histograms of the generated attributes (attribute1 in Figure 4.7) with three variations of attribute-class correlations, $\boldsymbol{H}^{(1)}$, $\boldsymbol{H}^{(2)}$, and $\boldsymbol{H}^{(3)}$. All attributes of the three variations follow user-specified distributions which are normal distributions even when given attribute-class correlations are different from each other.

Graph feature regarding attribute

First, in order to validate that the attributes of generated graphs with various attribute-class correlations follow user-specified distributions (i.e., the graph feature regarding attributes), Figure 4.6 shows the histograms of a single attribute $\boldsymbol{H}_{1.}$ for each variation of attribute-class correlations. We observe that all distributions of the attribute values in the generated graphs with $\boldsymbol{H}^{(1)}$, $\boldsymbol{H}^{(2)}$, and $\boldsymbol{H}^{(3)}$ follow normal distributions, which are shown in Figure 4.6a, 4.6b, and 4.6c, respectively. Hence, we conclude that GenCAT can generate attribute values according to the given distribution even when users input various attribute-class correlations. To quantitatively evaluate the effectiveness of the application of distributions, we investigate the distance between the distributions of generated attributes $\boldsymbol{X}$ and user-specified distributions $p(\boldsymbol{X})$ by using the EM distance (Eq. (4.16)). We also calculate the distance between user-specified distributions and the distributions of the matrix multiplication $\boldsymbol{U}\boldsymbol{V}^\top$ (i.e., we directly use the matrix multiplication) for comparison. Table 4.5 shows the results of the EM distances for $\boldsymbol{H}^{(1)}$, $\boldsymbol{H}^{(2)}$, and $\boldsymbol{H}^{(3)}$. Since the distances between $\boldsymbol{X}$ and $p(\boldsymbol{X})$ are smaller than the distance between $\boldsymbol{U}\boldsymbol{V}^\top$ and $p(\boldsymbol{X})$, we confirm the effectiveness of the application of user-specified distributions in the attribute generation.



(a) $\boldsymbol{H}^{(1)} = [[0.5, 0.0, 0.0, 0.5], [0.0, 0.5, 0.0, 0.5]]$. (b) $\boldsymbol{H}^{(2)} = [[0.4, 0.1, 0.1, 0.4], [0.1, 0.4, 0.1, 0.4]]$. (c) $\boldsymbol{H}^{(3)} = [[0.3, 0.2, 0.2, 0.3], [0.2, 0.3, 0.2, 0.3]]$.

Figure 4.7: Distributions of the generated attributes with three variations of attribute-class correlations. We depict the 2-D plots of the values of two attributes. The colors indicate classes which nodes belong to.

Class feature regarding attribute

Next, to show that the generated attributes support the class features specified by users, Figure 4.7 depicts 2-D plots of the values of two attributes in the generated graphs with $\boldsymbol{H}^{(1)}$, $\boldsymbol{H}^{(2)}$, and $\boldsymbol{H}^{(3)}$. Intuitively, a range of values of $\boldsymbol{H}$ indicates the differences of the attribute values for classes. Since $\boldsymbol{H}^{(1)}$ has a wide range of values (i.e., $[0.0, 0.5]$) the attributes in different classes tend to have dissimilar values in Figure 4.7a. Then, Figure 4.7b and 4.7c show that the attributes of the nodes in classes are more mixed when the values of the attribute-class correlations are more similar between classes. Through this experiment, we observe that GenCAT can flexibly control the class structure in the generated attributes by user-specified attribute-class correlations. In summary of this subsubsection, we show that the attributes in graphs generated by GenCAT closely follow the users' desired graph/class features.

4.4.3 Scalability (Q3)

To investigate the scalability of GenCAT, we demonstrate the runtime and memory consumption with varying the number of edges. We vary the number of edges m within the range of $\{2^{21}, 2^{22}, 2^{23}, 2^{24}, 2^{25}, 2^{26}, 2^{27}, 2^{28}, 2^{29}, 2^{30}\}$. We set the parameters y, r, and n to 5, 50, and $m/32$, respectively, all diagonal elements of $\boldsymbol{M}$ to 0.6, the other elements of $\boldsymbol{M}$ to 0.1, all diagonal elements

Table 4.6: Execution time and memory consumption: TO and OOM indicate that the execution is not finished in 36 hours and is out of memory, respectively.

	m	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
Time [sec]	GenCAT	$1.90e2$	$3.77e2$	$7.50e2$	$1.56e3$	$3.16e3$
	LFR	$2.17e2$	$7.66e2$	$4.19e3$	$1.50e4$	$6.05e4$
	DC-SBM	$9.60e3$	OOM	OOM	OOM	OOM
Memory [MiB]	GenCAT	$9.24e2$	$1.78e3$	$3.51e3$	$7.33e3$	$1.48e4$
	LFR	$1.40e3$	$2.25e3$	$3.99e3$	$7.45e3$	$1.39e4$
	DC-SBM	$6.37e5$	OOM	OOM	OOM	OOM
	m	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}
Time [sec]	GenCAT	$6.53e3$	$1.34e4$	$2.84e4$	$5.92e4$	$1.23e5$
	LFR	TO	TO	TO	TO	TO
	DC-SBM	OOM	OOM	OOM	OOM	OOM
Memory [MiB]	GenCAT	$2.95e4$	$5.92e4$	$1.19e5$	$2.37e5$	$4.92e5$
	LFR	TO	TO	TO	TO	TO
	DC-SBM	OOM	OOM	OOM	OOM	OOM

of $\boldsymbol{D}$ to 0.2, and the other elements of $\boldsymbol{D}$ to 0.1. We compare GenCAT with LFR and DC-SBM, which are the state-of-the-art generators closest in functionality to GenCAT.[23]

We first evaluate runtime of GenCAT, LFR, and DC-SBM for topology structure generation; recall that LFR and DC-SBM do not support attribute generation. In Table 4.6, we show the runtime to generate edges. This table indicates that GenCAT scales linearly to the number of edges and can generate graphs with billion edges in a reasonable time (i.e., the generation for 2^{30} edges finishes in 35 hours). Comparing GenCAT with LFR and DC-SBM, GenCAT significantly outperforms them. This is because GenCAT efficiently generate edges due to the inverse transform sampling, but LFR often fails to generate edges as the graph size increases, and DC-SBM's complexity is

[23]We do not compare with TrillionG that is the state-of-the-art method in terms of scalability. The graphs generated by TrillionG are significantly different from those generated by GenCAT since TrillionG is a schema-driven approach without attributes and without flexible control of the class structure.

$O(n^2)$. From the results, we confirm that GenCAT efficiently and scalably generates graphs with class labels.

Next, we analyze memory consumption. Table 4.6 shows the memory usage to generate graphs. It shows that GenCAT scales linearly to the number of edges. GenCAT and LFR have similar memory consumption. We observe that DC-SBM operates with memory usage proportional to the square of the number of nodes.

In summary, these experiments validate that both time and space complexities of GenCAT are linear to the number of edges, it significantly outperforms the state-of-the-art, and it generates edges with the controlled structure of large sizes not possible to practically generate with the state-of-the-art.

4.4.4 Reproduction of Real-world Graphs (Q4)

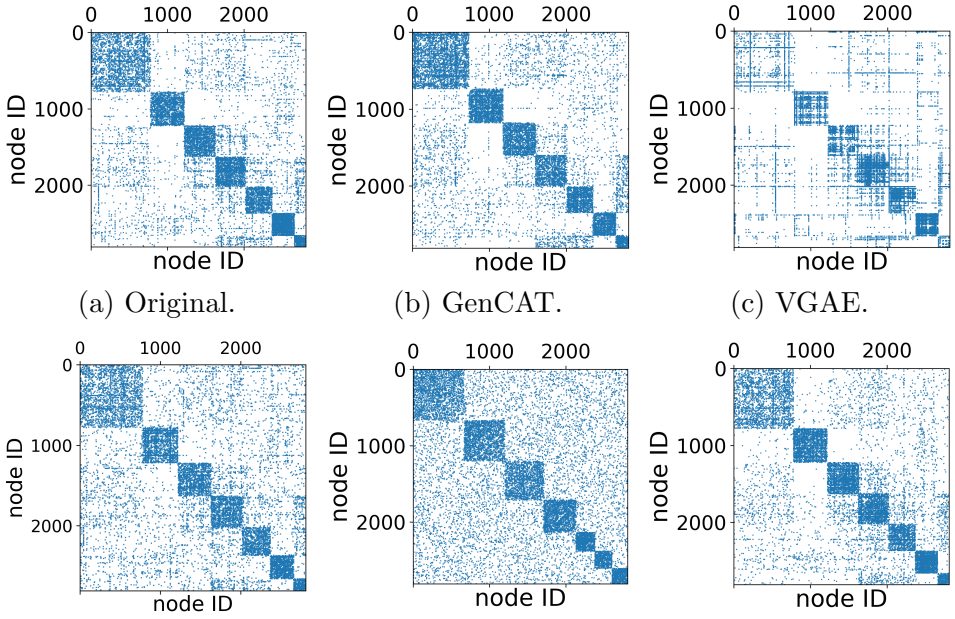
To evaluate reproducibility of real graphs with GenCAT, we show how precisely GenCAT reproduces the real-world graph by comparing with existing methods. We use GenCAT, VGAE, NetGAN, LFR, and DC-SBM[24] to reproduce the CORA dataset [85], which is a typical graph dataset often used in studies of community structure, e.g., [19, 60]. In this graph, the numbers of nodes, edges, and classes are 2810, 7981, and 7, respectively.

First, we clarify and explain the strong points and drawbacks of GenCAT and existing methods by visualizing the adjacency matrices of CORA dataset and generated graphs. Second, we quantitatively evaluate the class structure in reproduced graphs by the class features introduced in Section 4.2.2. Third, we also evaluate the class structure by community-related statistics which are commonly used [19]. Finally, we demonstrate that GenCAT can generate various size graphs with the same class structure as a given graph.

Visualization

Figure 4.8 shows the adjacency matrices of CORA dataset and generated graphs. For a fair comparison, we randomly order the nodes in each class. We here note that since VGAE and NetGAN do not explicitly generate the class labels and the nodes in generated graphs bijectively correspond to the

[24]LFR and DC-SBM do not have a function to reproduce given graphs. So, for LFR we manually set parameters appropriately to reproduce given graphs. As for DC-SBM, we compute node degrees by the same way as GenCAT and input them as parameters for DC-SBM.



(a) Original. (b) GenCAT. (c) VGAE.

(d) NetGAN. (e) LFR. (f) DC-SBM.

Figure 4.8: Visualization of adjacency matrices of CORA dataset and generated graphs.

nodes in original graphs, we assign nodes of the generated graph to the class labels of the corresponding nodes of the original dataset. From Figure 4.8, we observe that GenCAT most precisely reproduces CORA among the five methods. This is because GenCAT can capture the connection proportions between nodes and classes. VGAE does not precisely reconstruct the graph structure since its main purpose is learning node embeddings. NetGAN does not explicitly consider the class structures in its learning steps, so it fails to reproduce the detailed parts. In LFR, inter-edges are uniformly distributed because LFR randomly generates them. In DC-SBM, inter-edges of each pair of classes are uniformly distributed since DC-SBM assumes that the nodes in the same class have the same connection proportions.

Evaluation on class features

To evaluate the accuracy quantitatively, we measure the mean square errors (MSE) of the class preference mean and class preference deviation between

Table 4.7: Evaluation for class preference mean and class preference deviation between the original graph and generated graphs. Mean squared error (MSE) is used for the evaluation measure.

Mean squared error	GenCAT	VGAE	NetGAN	LFR	DC-SBM
Class preference mean	8.71e–4	2.05e–2	2.93e–3	7.05e–3	1.66e–3
Class preference deviation	9.26e–4	4.12e–3	1.86e–3	5.36e–3	1.42e–3

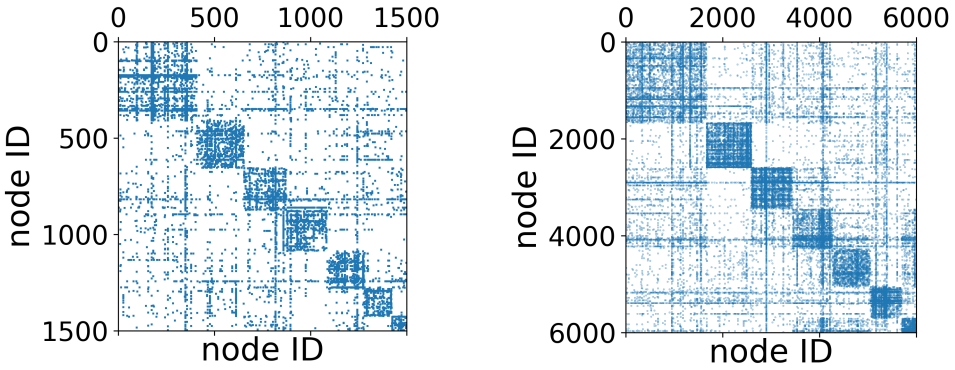
Table 4.8: Statistics of Cora and the graphs generated by GenCAT and the baselines, averaged over five trials.

	Original	GenCAT	VGAE	NetGAN	LFR	DC-SBM
Intra-community density	1.97e-3	1.89e-3	2.04e-3	1.41e-3	1.36e-3	1.88e-3
Inter-community density	5.07e-4	4.92e-4	4.87e-4	6.0e-4	7.03e-4	5.01e-4
Size of LCC	2810.0	2810.0	1686.4	2804.2	2810.0	2499.0
# connected components	1.0	1.0	1123.4	3.4	1.0	306.2
Characteristic path length	5.27	4.36	3.99	5.19	4.74	4.14
Average rank	-	1.8	3.8	3.0	2.8	3.2

the original graph and generated graphs. Table 4.7 shows the MSE of all the methods. GenCAT achieves the best performance, and this result indicates that GenCAT can precisely reproduce the class structures in a given real graph.

Evaluation on community-related statistics

We evaluate the quality of generated graphs by statistics related to communities, in order to show that GenCAT can generate realistic communities in a graph. The statistics include intra-community density, inter-community density, the size of largest connected component (called LCC for short), the number of connected components, and characteristic path length (average number of steps along the shortest paths for all node pairs), which are used in [19]. We observe that graphs generated by GenCAT are similar to the original graph in terms of all community-related statistics. The bottom row



(a) $n = 1500, m = 5000$. (b) $n = 6000, m = 20000$.

Figure 4.9: Visualization of adjacency matrices of generated graphs with various sizes.

in Table 4.8 shows the average rank of each method over all statistics and demonstrates that GenCAT ranks the highest. This result means that GenCAT reproduces graphs with similar statistics to the original one. Though VGAE and DC-SBM precisely capture intra- and inter-community densities, they generate highly disconnected graphs. NetGAN achieves the closest score for characteristic path length because it learns random walks on a given graph. However, NetGAN cannot accurately capture intra- and inter-community density because it does not explicitly learn communities in a given graph. LFR fails to accurately reproduce intra- and inter-community density from the original graph since it assumes that all communities have the same density. In summary, we validate that GenCAT can generate realistic communities in a graph with regard to various statistics used commonly.

Changing graph size

Next, we demonstrate GenCAT generates various size graphs which have the same class structures as a given graph. Figure 4.9 shows two adjacency matrices of graphs generated by GenCAT: the parameters of the former are $n = 1500$ and $m = 5000$, and those of the latter are $n = 6000$ and $m = 20000$. Figure 4.9a shows the reproduced graph has the similar class structure to

Table 4.9: Ablation study on edge generation. We show execution time, MSE between user specified class preference mean and the class preference mean of a generated graph, and MSE between user specified class preference deviation and the class preference deviation of a generated graph. "w/o ITS" and "w/o AP" indicate the variants removing inverse transform sampling and adjusting proportion, respectively. We report the scores averaged over five trials.

	m	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Time[sec]	GenCAT	6.34	12.30	23.78	47.71	95.29
	w/o ITS	30.70	110.10	421.84	1683.24	6714.91
	w/o AP	6.01	11.52	22.15	44.87	89.87
Class preference mean	GenCAT	1.25e-3	0.42e-3	0.20e-3	0.14e-3	0.09e-3
	w/o ITS	1.19e-3	0.49e-3	0.24e-3	0.16e-3	0.09e-3
	w/o AP	14.96e-3	13.57e-3	13.32e-3	12.96e-3	12.93e-3
Class preference deviation	GenCAT	2.71e-3	2.21e-3	2.09e-3	2.14e-3	2.02e-3
	w/o ITS	3.05e-3	2.35e-3	2.30e-3	1.99e-3	2.11e-3
	w/o AP	5.22e-3	4.89e-3	4.89e-3	4.85e-3	4.82e-3

CORA dataset shown in Figure 4.8a[25]. Figure 4.9b shows that the larger graph also has a similar class structure to the original dataset.

These experiments validate that GenCAT can generate the class structures of real-world graphs and enable users to choose arbitrary sizes of generated graphs while maintaining the accuracy of these structures.

4.4.5 Ablation Study

To validate the efficiency of inverse transform sampling (ITS) and the effectiveness of adjusting proportion (AP), We evaluate the effect on edge generation of ITS and AP. We vary the number of edges m within the range of $\{2^{16}, 2^{17}, 2^{18}, 2^{19}, 2^{20}\}$. We set the parameters y, r, and n as 6, 50, and $m/16$, respectively, all diagonal elements of $\boldsymbol{M}$ to 0.6, and the other elements to 0.08. Also, we set the diagonal elements of $\boldsymbol{D}$ to $[0.2, 0.2, 0.25, 0.25, 0.3, 0.3]$, respectively, and the other elements to 0.05.

[25]We note that the adjacency matrix looks more sparse than the original dataset due to the size of the figure.

Table 4.9 shows the execution time and losses between user specified class preference mean/deviation and the class preference mean/deviation of a generated graph. We use MSE in order to measure the losses between those class preference means/deviations. In this table, "w/o ITS" and "w/o AP" indicate the variants removing ITS and AP, respectively. First, the table shows that GenCAT and w/o AP scale linearly to the number of edges. On the other hand, w/o ITS requires $O(n^2)$ because it calculates the matrix multiplication $\boldsymbol{U}\boldsymbol{U}'^{\top}$ to obtain edge probability. Hence, we validate that ITS accelerates the edge generation of GenCAT. Second, in Table 4.9 we observe that GenCAT and w/o ITS can generate graphs that satisfy the constraints of class preference mean/deviation better than w/o AP. From this observation, we validate that AP largely reduces the losses between user specified class preference mean/deviation and the class preference mean/deviation of a generated graph. Since the losses of GenCAT and w/o ITS are comparable, this result indicates that ITS does not decrease the quality of generated graphs.

In summary, we conclude that ITS improves the efficiency of GenCAT while keeping the quality of edge generation in terms of class preference mean/deviation. Also we validate that AP improves the quality of edge generation.

4.4.6 Summary of This Section

We experimentally validated four major aspects of GenCAT: 1) edges in generated graphs follow user-specified graph features (i.e., node degrees) and user-specified class features (i.e., the proportions of connections between nodes and classes), 2) GenCAT can generate attributes that follow user-specified attribute distributions and the controlled class structure, 3) GenCAT scales linearly to the number of edges, and we show that GenCAT generates graphs with billion edges, 4) GenCAT more precisely reproduces the class structure in real-world graphs than existing methods. GenCAT is the first graph generation method having all four of these practical features. Through our experiments, we demonstrated that GenCAT can successfully generate massive attributed graphs with sophisticated user-controlled class structures.

Since GenCAT enables us to generate various graphs that are suitable to evaluate GNNs, we conduct comprehensive evaluations of GNNs for node classification with various graphs generated by GenCAT in the next section.

4.5 An Empirical Study of Node Classification with GNNs

Thanks to our graph generator GenCAT proposed in Section 4.3, we can generate various graphs by synthetically changing one or a few target characteristic(s) of graphs for fine-grained analysis. In this section, we generate a variety of graphs to evaluate GNNs, in terms of four major characteristics of attributed graphs with class labels: 1) class size distributions, 2) the relationship between classes and topology, 3) the relationship between classes and attributes, and 4) graph sizes. To be concrete, we attempt to answer the following questions: **Q1.** *To what extent do class size distributions affect the performance of GNNs?* **Q2.** *How effectively do GNNs work on graphs with various edge connection proportions between classes?* **Q3.** *To what extent do attribute values contribute to the performance of GNNs?* **Q4.** *How effectively do GNNs work on graphs with various sizes?*

First, we describe the setups of our empirical studies. Second, we show experiments regarding node classification performance and discuss the results in detail. Finally, we measure the training time per epoch of GNNs on synthetic graphs with various sizes to investigate their efficiency.

4.5.1 Experimental Setup

We use 16 GNNs including the state-of-the-art methods such as GPRGNN [23] and shaDow-GNN [126] as we discussed in Section 3[26]. We also execute a graph-agnostic classifier (i.e., it ignores the topology structure), multi-layer perceptron (MLP), in order to evaluate how largely the topology structure contributes to classification performance. We use random class splits (60%/20%/20% of nodes for train/validation/test) whose ratio is provided by [23]. We perform a grid search to tune the hyperparameters of GNNs to generated graphs for each setting. The hyperparameter search space and their best parameter set for each setting are reported in our codebase for the experimental reproducibility. To reduce the randomness, we generate three graphs for each setting of graph generation and execute GNNs with three restarts for each generated graph. We report average scores and standard deviations as error bars.

[26]We drop several existing methods such as GIN [113] and SIGN [37] due to the space limitation. However, we do not fail to use the state-of-the-art methods.

In our experiments, we extract parameters of the graph generation from the Cora dataset, i.e., we obtain class and graph features from Cora. Note that Cora has 2708 nodes, 5278 edges, 1433 attributes, and 7 classes. Then, we configure one or a few parameters for each setting[27]. We use f1-macro to evaluate classification quality, which can better reflect the performance on minority classes than accuracy. We measure training time on a NVIDIA Tesla V100S GPU (32GB) and Intel(R) Xeon(R) Gold 5220R CPU×2 (378GB).

Hyperparameter Search Space

We select hyperparameters for search space of GNNs according to their papers or codebases. We show hyperparameter search space for each GNN and MLP as follows.

MLP. The hyperparameter search space of MLP is listed as follows:

- Weight decay: $[0, 5e\text{-}6, 5e\text{-}5, 5e\text{-}4]$
- Learning rate: $[0.002, 0.01, 0.05]$
- Early stopping: $[40, 100]$
- Hidden layer: $[64]$
- Dropout: $[0.5]$

GCN.

- Weight decay: $[0, 5e\text{-}6, 5e\text{-}5, 5e\text{-}4]$
- Learning rate: $[0.002, 0.01, 0.05]$
- Early stopping: $[40, 100]$
- Hidden layer: $[16, 32, 64]$
- Dropout: $[0.5]$

Monet.

- Weight decay: $[5e\text{-}5, 5e\text{-}4, 1e\text{-}4]$

[27]Parameters not mentioned are set to those extracted from Cora.

- Learning rate: [1e-4, 0.002, 0.001, 0.01]
- Early stopping: [100]
- Hidden layer: [64]
- Aggregation: [mean, max]
- Dropout: [0.5]

ChebNet.

- Weight decay: [0, 5e-6, 5e-5, 5e-4]
- Learning rate: [0.002, 0.01, 0.05]
- Early stopping: [40, 100]
- Hidden layer: [16, 32, 64]
- Dropout: [0.5]

GAT.

- Weight decay: [0, 5e-6, 5e-5, 5e-4]
- Learning rate: [0.002, 0.01, 0.05]
- Early stopping: [40, 100]
- Hidden layer: [16, 32, 64]
- Dropout: [0.5]
- Heads: [8]
- Output heads: [1, 4, 8]

GraphSAGE.

- Weight decay: [0, 5e-6, 5e-5, 5e-4]
- Learning rate: [0.002, 0.1, 0.7]
- Epochs: [500]

- Early stopping: $[40, 100]$
- Hidden layer: $[64, 128]$
- Layers: [2]
- Dropout: [0.5]

SGC.

- Weight decay: $[0, 5e\text{-}6, 5e\text{-}5, 5e\text{-}4]$
- Learning rate: [0.2]
- Epochs: [100]
- Early stopping: [40]

JKNet.

- Skip connection: [cat, max, lstm]

As for the hyperparameters of base models, GCN, GAT, and GraphSAGE, we follow the best hyperparameters of base models.

GraphSAINT (-GAT and -GraphSAGE).

- Weight decay: [0]
- Learning rate: $[0.002, 0.01, 0.01]$
- Epochs: [300]
- Early stopping: [40]
- Hidden layer: $[128, 256]$
- Dropout: $[0, 0.1, 0.2]$
- Heads: [8]
- Output heads: [1]
- Walk length: $[2, 4]$

- Sample coverage: [50]
- Root: [1500, 2000]

shaDow-GNN (-GAT and -GraphSAGE).

- Weight decay: [0]
- Learning rate: [0.002, 0.01, 0.01]
- Epochs: [200]
- Early stopping: [40]
- Hidden layer: [128, 256]
- Dropout: [0, 0.1, 0.2]
- Heads: [8]
- Output heads: [1]
- Depth: [2]
- Number of neighbors: [5, 10, 20]
- Batch size: [64, 128]

FSGNN.

- Weight decay: [0, 5e-6, 5e-5, 5e-4]
- Learning rate: [0.002, 0.01, 0.05]
- Early stopping: [40, 100]
- Weight decay for attention: [0.001, 0.01, 0.1]
- Dropout: [0.5]
- Hidden: [64]
- Number of layers: [3, 8]

GPRGNN.

- Weight decay: $[0, 5e\text{-}6, 5e\text{-}5, 5e\text{-}4]$
- Learning rate: $[0.002, 0.01, 0.05]$
- Early stopping: $[40, 100]$
- Dropout: $[0.5]$
- Parameter initializing attention: $[0.1, 0.5, 0.9]$
- Number of propagation layer: $[10]$
- Number of layer of MLP: $[3]$

LINKX.

- Weight decay: $[0.001]$
- Learning rate: $[0.002, 0.01, 0.05]$
- Hidden layer: $[64, 128]$
- Dropout: $[0, 0.5]$
- Number of edge layers: $[1, 2]$
- Number of node layers: $[1, 2]$
- Number of prediction layer: $[2, 3, 4]$

Best Parameters

For the reproduction of Figures 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15, we report the best set of hyperparameters for each experiment. Since we use over 20 synthetic graphs and 16 GNN models, we provide the best parameter sets in our codebase. Please see README in the repository, to find the best parameter sets.

Table 4.10: URLs of baseline codes.

Baseline	URLs
SGC	`https://github.com/Tiiiger/SGC`
H2GCN	`https://github.com/GemsLab/H2GCN`
FSGNN	`https://github.com/sunilkmaurya/FSGNN`
GPRGNN	`https://github.com/jianhao2016/GPRGNN`

Source Codes of Baseline GNNs

Table 4.10 summarizes the URLs to download the baseline codes. As for GCN, MoNet, ChebNet, GAT, JKNet, GraphSAGE, GraphSAINT, ShadowGNN, and LINKX, PyTorch Geometric[28] provides their model architectures, so we implemented the GNN algorithms based on PyTorch Geometric. As for H2GCN, the authors implemented it in TensorFlow, so we reimplemented it in PyTorch for fair comparisons.

4.5.2 Classification Quality on Synthetic Graphs with Various Characteristics

Various Class Size Distributions

Most studies on GNNs for node classification do not consider class size distributions in their evaluations. In fact, existing studies use only accuracy to evaluate the classification quality since they do not consider class size distributions. However, class size distributions are typically imbalanced [44] in practical use cases. Instead of using accuracy, we use f1-macro which is suitable to equally treat major and minor classes. The trade-off between fairness and accuracy has been actively discussed [4, 25]. We also show experimental results regarding accuracy in the supplementary materials.

Detailed setup. To investigate the extent to which class imbalance affects the classification performance, we conduct experiments on synthetic graphs

[28] `https://github.com/pyg-team/pytorch_geometric`

with configured class size distributions $\boldsymbol{\rho}^{\text{conf}}$ as follows:

$$\boldsymbol{\rho}_l^{\text{conf}} = \begin{cases} \alpha & (l = 1) \\ \alpha * (1 - \sum_{l'=1}^{l-1} \boldsymbol{\rho}_{l'}^{\text{conf}}) & (1 < l < y) \\ 1 - \sum_{l'=1}^{l-1} \boldsymbol{\rho}_{l'}^{\text{conf}} & (l = y) \end{cases}, \tag{4.21}$$

where α indicates the size of the largest class. We set $\alpha \in [0.4, 0.5, 0.6, 0.7]$ to investigate how GNNs perform on graphs with imbalanced classes, which most existing studies ignore. Note that since the Cora dataset has seven classes, the classes in generated graphs are imbalanced when $\alpha = 0.4, 0.5, 0.6,$ and 0.7. To compare these imbalanced settings with a balanced setting, we also conduct experiments with graphs having completely balanced classes. For the intuitive explanation, we give the visualization showing how generated graphs look like in the supplementary material.

Observations. Figure 4.10 shows the node classification performance, i.e., f1-macro, with various class sizes. No methods consistently outperform other methods across the balanced and imbalanced settings. In the balanced setting, the very recent method GPRGNN achieves the best f1-macro due to its high expressive capability. On the other hand, interestingly the simplest algorithm SGC outperforms other complicated GNN algorithms in the imbalanced settings, $\alpha = 0.4, 0.5, 0.6,$ and 0.7. A few studies [95, 109, 130] have addressed a class imbalance problem with GNNs. However, they focus on typical homophilic graphs and ignore more complicated settings such as the combinations of imbalanced classes, heterophily property, and large-scale graphs. For example, widely used heterophilic datasets Texas, Wisconsin, and Cornell have imbalanced classes, i.e., their largest classes contain $55\%, 55\%, 47\%$ of nodes, respectively, while they all have five classes.

Various Edge Connection Proportions between Classes

To clarify in detail how effectively GNNs perform on graphs with various edge connection proportions between classes, we conduct experiments with fine-grained patterns of synthetic graphs in terms of edge connection proportions.

Detailed setup. To generate graphs with various edge connection proportions, we configure the class preference means $\boldsymbol{M}^{\text{conf}}$ as follows:

$$\boldsymbol{M}_{l_1 l_2}^{\text{conf}} = \begin{cases} \max(\boldsymbol{M}_{l_1 l_2}^{\text{Cora}} - 0.1 * \beta, 0) & (l_1 = l_2) \\ \boldsymbol{M}_{l_1 l_2}^{\text{Cora}} + 0.1 * \beta/(k-1) & (l_1 \neq l_2) \end{cases}, \tag{4.22}$$

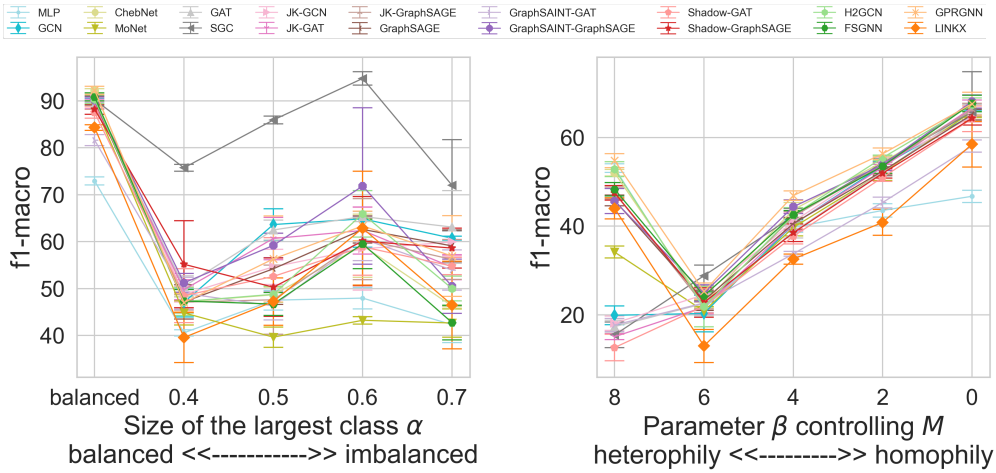


Figure 4.10: Classification performance on graphs with various class sizes.

Figure 4.11: Classification performance on graphs with various edge connection proportions.

where β is a parameter controlling the homophily/heterophily property in a graph and $\boldsymbol{M}^{\text{Cora}}$ indicates the class preference mean of Cora. Note that since the average diagonal elements of $\boldsymbol{M}^{\text{Cora}}$ is 0.81, synthetic graphs without modifying the configuration generated from the Cora dataset are also homophilic graphs. If β is small, classes have many intra-edges, i.e., homophily property. In contrast, if β is large, classes have few edges internally, i.e., heterophily property. For intuitive explanation, we show how generated graphs look like when varying β in the supplementary material. We generate synthetic graphs for each $\beta \in [0, 2, 4, 6, 8]$.

Observations. Figure 4.11 shows the node classification performance with various edge connection proportions, i.e., class preference means. First, all models work well in the homophily setting (see the rightmost points). On the other hand, in the heterophily setting (see the leftmost points), GNNs generalizing to heterophilic graphs perform well such as H2GCN, FSGNN, and GPRGNN, while typical GNNs such as GCN and GAT fail to perform well. However, they obtain the marginal improvement from MLP in the heterophily setting despite their complicated designs. This indicates that GNNs generalizing heterophilic graphs hardly utilize the topology information and almost ignore the information in this setting even if graphs have the strong heterophily property. Note that the average diagonal elements

of $\boldsymbol{M}^{\text{conf}}$ is 0.03 when $\beta = 8$, i.e., there are only few edges inside a class. Since GraphSAGE adopts ego- and neighbor-embedding separation, it obtains relatively high f1-macro scores in the heterophily setting. The very recent method LINKX cannot achieve the state-of-the-art f1-macro scores. This is because we just follow the hyperparameter search space specified in [72] (for details, see the hyperparameter search space described in the supplementary material). Broader search space may increase the classification performance. Finally, all models obtain low f1-macro scores when edges are almost randomly generated between classes (see plots locating around $\beta = 6$ in the horizontal axis). This is because the values of class preference means are similar to uniform values in this case[29].

Various Attribute Values

We aim to clarify the extent to which attribute values affect the performance of GNNs by using fine-grained patterns of synthetic graphs in terms of attribute values.

Detailed setup. We mix the attribute-class correlation $\boldsymbol{H}^{\text{Cora}}$ of Cora into a uniform distribution with a certain degree γ, to generate graphs with various attribute-class correlations, i.e., from biased attributes for classes to random attributes. The mixing calculation formula to configure the attribute-class correlations $\boldsymbol{H}^{\text{conf}}$ is as follows: $\boldsymbol{H}^{\text{conf}} = (\boldsymbol{H}^{\text{Cora}} + \gamma c)/(1 + \gamma)$, where c is the average value of $\boldsymbol{H}^{\text{Cora}}$, i.e., $c = \sum_{i=1}^{d} \sum_{j=1}^{k} \boldsymbol{H}_{ij}^{\text{Cora}}/(d * k)$. If $\gamma = 0$, $\boldsymbol{H}^{\text{conf}}$ corresponds to the attribute-class correlation of the original. If γ is large, H^{conf} is close to uniform, i.e., H^{conf} is less informative to predict the class labels of nodes. We generate graphs with configured attribute-class correlations for $\gamma \in [16, 4, 1, 0]$. To compare the above setting with random attributes, we also conduct experiments with graphs having uniform attribute-class correlation, i.e., every element of $\boldsymbol{H}^{\text{conf}}$ is c.

Observations. Figure 4.12 shows the node classification performance with various attribute values. We observe that most models obtain low f1-macro scores when attribute values are close to random, i.e., γ is large. The reason that SGC works well in some cases is that it does not over-fit large classes due to the simplicity of its model. Remember that Cora has weakly imbalanced classes as we discussed in Section 4.5.2. However, SGC is not stable

[29]Note that attributes also are almost random in this case since the class structure is shared by the edge and attribute generation steps of GenCAT.

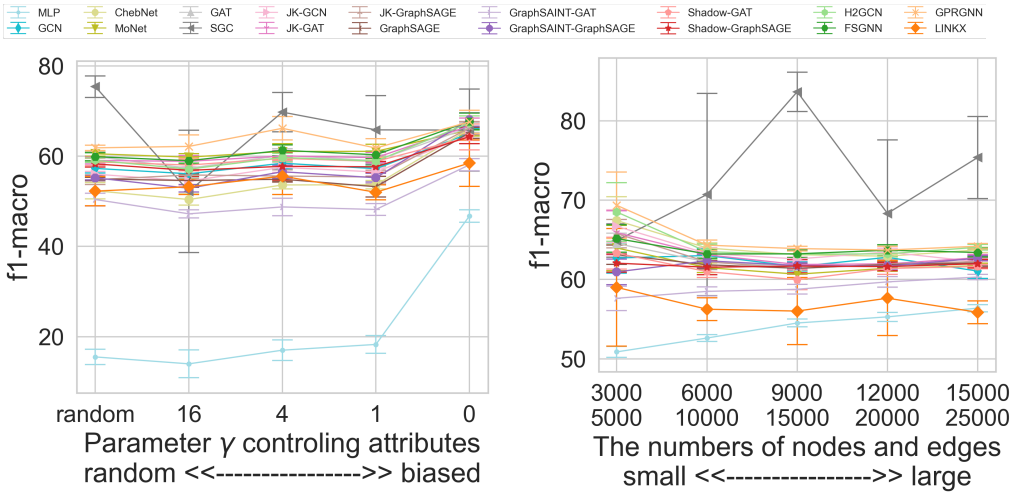


Figure 4.12: Classification performance on graphs with various attribute values.

Figure 4.13: Classification performance on graphs with various graph sizes.

across various settings. Another observation is that the very recent method GPRGNN consistently performs well compared with other GNNs. Finally, MLP achieves the larger performance gain than most GNNs between the leftmost and rightmost points (random and biased attributes) in Figure 4.12. This indicates some overlap between the contributions of the topology and attributes to node classification.

Various graph sizes

To clarify how graph sizes affect the performance of GNNs, we conduct experiments with graphs having various sizes and the same characteristics other than sizes.

Detailed setup. To generate graphs with various sizes, we set pairs of the numbers of nodes and edges (n, m) to $[(3000, 5000), (6000, 10000), (9000, 15000), (12000, 20000), (15000, 25000)]$.

Observations. Figure 4.13 shows the classification performance with various graph sizes. We observe that several GNNs obtain lower f1-macro scores when graphs are larger. GNNs may tend to over-fit larger classes and give less priority to smaller classes when larger graphs can provide more patterns of

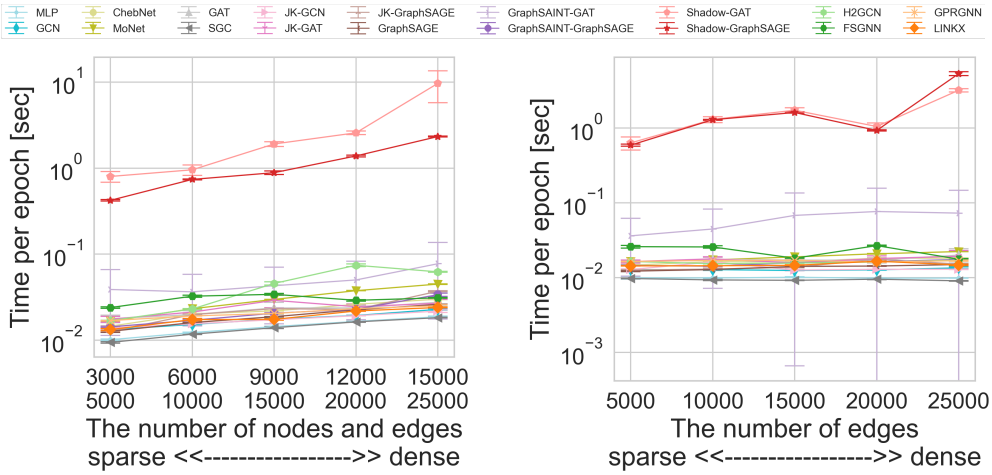


Figure 4.14: Training time per epoch of GNNs on graphs with various sizes.

Figure 4.15: Training time per epoch of GNNs on graphs with various numbers of edges.

their subgraphs for model training, because the loss functions of most GNNs are designed to increase their accuracy. Actually, when we use accuracy as a metric, most GNNs achieve better scores as graphs grow (we show the results regarding accuracy in the supplementary materials). As the same reason in Section 4.5.2, SGC works well in some cases but is not stable across various graph sizes. MLP increases its classification performance as graph sizes are large, since it appropriately fits to node attributes by using more train data from larger graphs.

4.5.3 Training Efficiency on Synthetic Graphs with Various Graph Sizes

Various numbers of nodes and edges

To investigate how graph sizes, i.e., the numbers of nodes and edges, affect training efficiency, we measure the total execution time of GNNs for model training with synthetic graphs with various sizes. Note that we use the same synthetic graphs in Section 4.5.2. We tune the hyperparameters of GNNs which maximize f1-macro scores by using a grid search.

Figure 4.14 shows the training time per epoch of GNNs on graphs with

various sizes. We observe that all methods tend to need longer training time per epoch as graphs grow, since the sizes of matrices storing the topology and attribute information increase. Shadow-GAT and Shadow-GraphSAGE require longer execution time than other GNNs because they execute a number of convolutional operations on sampled subgraphs.

Various numbers of edges and fixed number of nodes

To investigate how the edge densities of graphs affect training efficiency, we measure the execution time of GNNs for model training. We set the number of edges to $[5000, 10000, 15000, 20000, 25000]$ and the number of nodes to that of nodes in Cora.

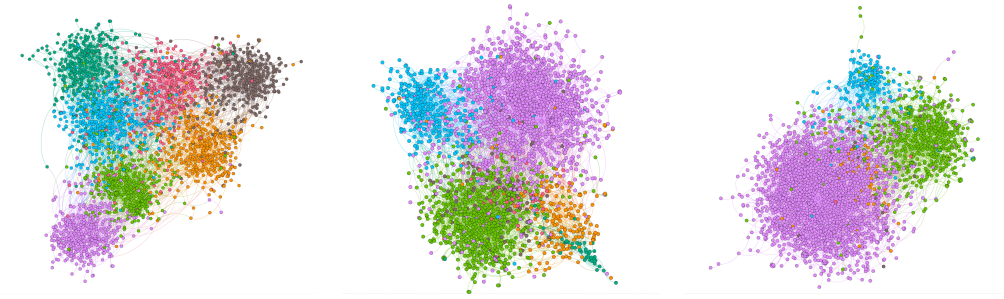
Figure 4.15 shows the training time per epoch of GNNs on graphs with various edge densities. Most GNNs require similar training time per epoch across sparse and dense settings since the size of the adjacency matrix is the same. They perform efficiently as long as a whole graph can be stored in a GPU memory[30]. Sampling-based GNNs, GraphSAGE, GraphSAINT, and Shadow-GNN, tend to need longer training time per epoch as the edge density becomes higher, since nodes in more dense graphs have more neighbors. The reason that the training time per epoch of Shadow-GAT and Shadow-GraphSAGE decreases at graphs with 20000 edges is that smaller numbers of sampled neighbors are selected in hyperparameter tuning to maximize their f1-macro scores. Also, the reason that the training time per epoch of FSGNN decreases at graphs with 15000 and 25000 edges is that fewer hops for feature aggregation are selected than those of other settings.

4.5.4 Visualization of Generated Graphs

To intuitively show how generated graphs with configured characteristics look like, we visualize several generated graphs used in our empirical studies. We first show graphs with various class size distributions in Figure 4.16. In Figure 4.16a, all classes share the same size. When we set $\alpha = 0.5$, the largest class (see the purple class in Figure 4.16b) includes 50% of nodes in a graph. Setting $\alpha = 0.7$ is a more extreme case. Figure 4.16c shows a graph where the large classes (the purple, green, and blue classes) include most nodes in

[30]We briefly discuss the application of GNNs to large-scale graphs that exceed the size of a GPU memory in Section 4.5.7

a graph and a few nodes belong to other classes (the red, brown, and orange classes).



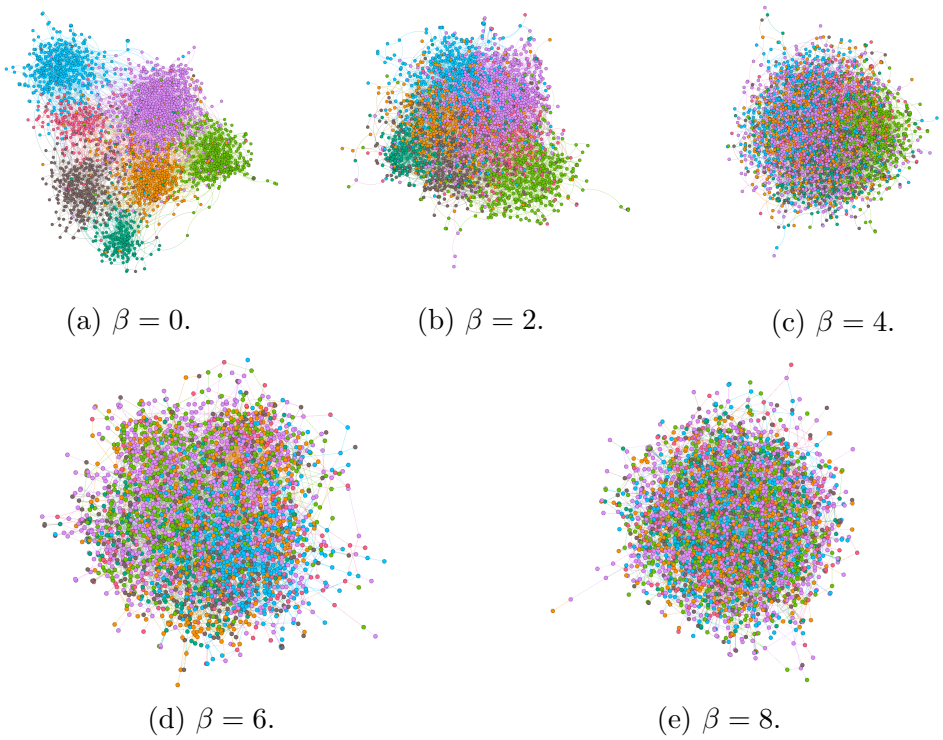
(a) Balanced. (b) $\alpha = 0.5$. (c) $\alpha = 0.7$.

Figure 4.16: Visualization showing graphs with various class size distributions. Colors indicate the class labels of nodes.

Next, we show graphs with various class preference means in Figure 4.17, which are used in Section 4.5.2. Figure 4.17a shows a graph with a strong homophily property, i.e., nodes in each class are densely connected. When we set $\beta = 2$, each class has more edges connecting to other classes than $\beta = 0$ while nodes in the same class are still plotted close together in Figure 4.17b. In Figure 4.17c, the class separation becomes more ambiguous, compared with smaller β. Graphs with larger β than 4 look like random graphs (see Figures 4.17d and 4.17e), Also, Figure 4.18 depicts heatmaps representing class preference means to show the exact values used in our experiments. The class preference means in Figures 4.18a, 4.18b, 4.18c, 4.18d, and 4.18e are used to generate graphs in Figures 4.17a, 4.17b, 4.17c, 4.17d, and 4.17e, respectively. The larger diagonal elements of class preference means indicate more edges within a class.

Finally, we visualize the attributes of generated graphs in Figure 4.19. In our experiments, we generate attributes that are binary for each dimension since the Cora dataset has a binary value for each attribute dimension. However, high-dimensional datasets with binary attributes are not suitable to visualize. To address this, we utilize t-SNE[31] to reduce the attribute dimension to two-dimensional real numbers. Figure 4.19a shows attributes

[31] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`



(a) $\beta = 0$. (b) $\beta = 2$. (c) $\beta = 4$.

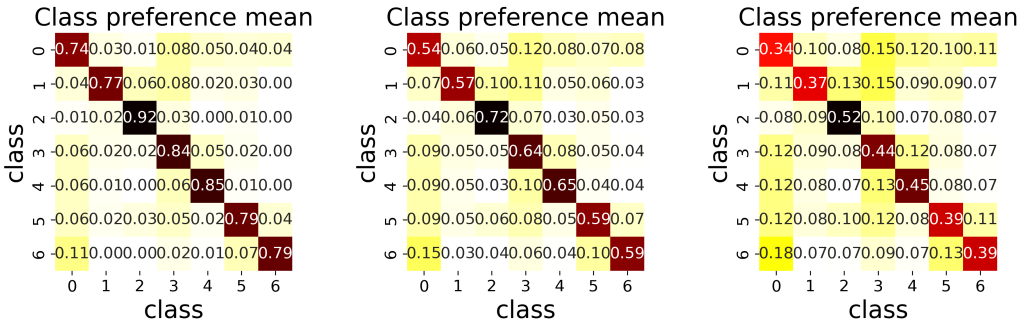
(d) $\beta = 6$. (e) $\beta = 8$.

Figure 4.17: Visualization showing graphs with various class preference means. Colors indicate the class labels of ndoes.

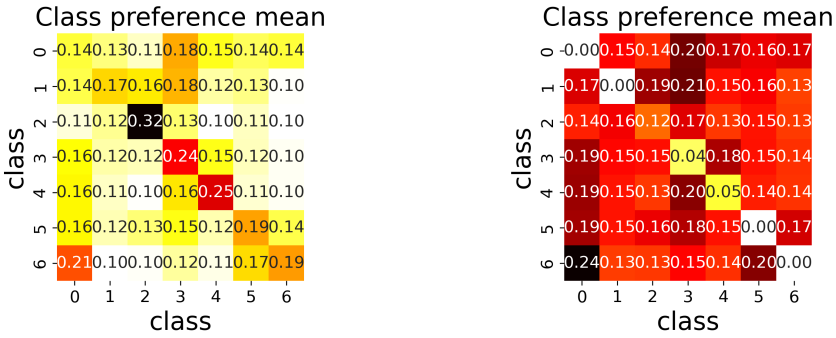
generated by using the attribute-class correlation extracted from the Cora dataset, i.e., $\gamma = 0$. Nodes in the same class (nodes with the same color) are plotted close together, e.g., many nodes colored with blue are plotted in the right bottom part. On the other hand, Figure 4.19b depicts generated attributes when setting $\gamma = 4$. The attributes are almost randomly plotted because the configured attribute-class correlation is mixed with the average value of Cora's attribute-class correlation in this setting, i.e., the configured attribute-class correlation has a weaker bias for some particular classes.

4.5.5 Accuracy Analysis on Node Classification

In Section 4.5.2, we utilize f1-macro which can better reflect the performance on minority classes. We also use accuracy to evaluate classification quality since it is commonly used in existing studies developing GNNs. We use



(a) $\beta = 0$. (b) $\beta = 2$. (c) $\beta = 4$.



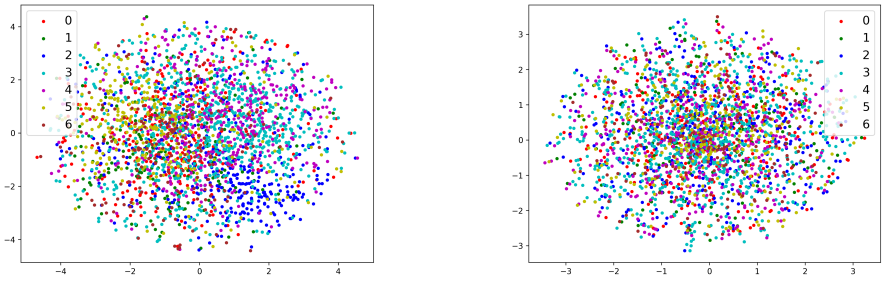
(d) $\beta = 6$. (e) $\beta = 8$.

Figure 4.18: Class preference means used in our experiments. Each cell represents the class preference mean of the generated graph.

the same hyperparameter search space to Section 4.5.2 and report the best parameter set for each setting in our codebase.

Various Class Size Distributions

Figure 4.20 shows accuracy on graphs with various class size distributions, which are the same setting to Section 4.5.2. We observe that all models obtain better accuracy as the size of the largest class increases. This result is the opposite of that of f1-macro. This suggests the risk of evaluating accuracy alone when class size distributions are not balanced. Another observation is that a very recent method GPRGNN achieves the highest accuracy across various class size distributions due to its model expressive capabil-



(a) $\gamma = 0$. (b) $\gamma = 4$.

Figure 4.19: Two-dimensional projection of attributes. Each dot indicates a node and colors indicate the class labels of ndoes.

ity. The reason that accuracy scores are not stable across various class size distributions is that accuracy highly depends on the characteristics of large classes, i.e., classes to which many nodes belong. Actually, we observe that the largest class has many edges within the class (i.e., the class is easy to classify) when $\alpha = 0.4, 0.6$. So, the accuracy scores in the settings are high. Note that this trend happened by chance because we shuffled class IDs before generating graphs.

Various Edge Connection Proportions

Figure 4.21 shows accuracy on graphs with various edge connection proportions between classes, which is the same setting to Section 4.5.2. All GNNs perform well in the homophily setting, i.e., $\beta = 0$. On the other hand, MLP and GNNs considering the heterophily perform well when $\beta = 8$. We observe that GPRGNN achieves the highest accuracy for all patterns of β since it can utilize the information from high-order neighbors by its deep propagation layer.

Various Attribute Values

Figure 4.22 shows accuracy on graphs with various attribute values, which is the same setting to Section 4.5.2. We observe that all models obtain lower accuracy as attribute values are closer to random and have a weaker bias for

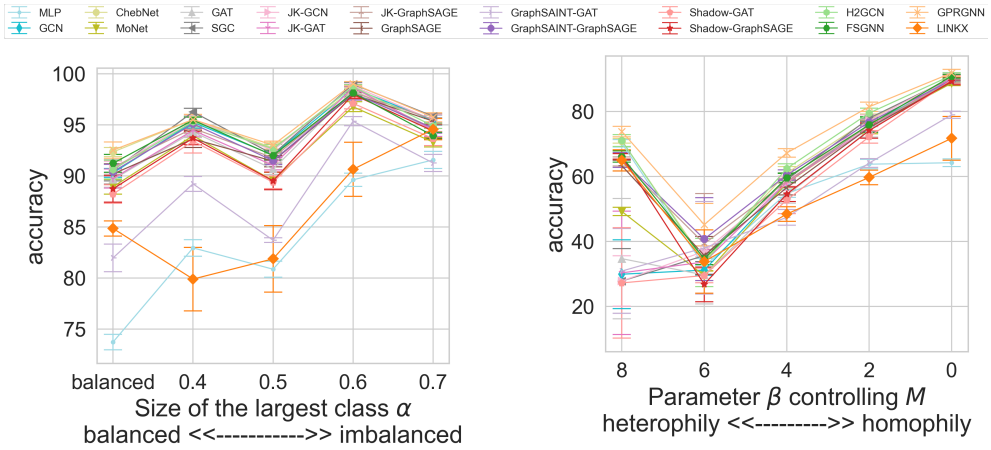


Figure 4.20: Accuracy on graphs with various class size distributions.

Figure 4.21: Accuracy on graphs with various edge connection proportions.

some classes, i.e., γ is larger. Also, GPRGNN achieves the highest accuracy across various settings.

Various Graph Sizes

Figure 4.23 shows accuracy on graphs with various graph sizes, which is the same setting to Section 4.5.2. We observe that most GNNs tend to obtain higher accuracy as graphs grow. This is because the models can fit to given graphs better by using more patterns of subgraphs from larger graphs. Since generated graphs have weakly imbalanced classes in this setting, interestingly we observe a different behavior between scores of f1-macro (in Figure 4.13) and accuracy (in Figure 4.23), i.e., accuracy slightly increases as graphs grow while f1-macro tends to decrease.

Through these experiments with accuracy, we clarify that careful selection of evaluation metrics is important to fairly compare GNNs since their classification performance depends on the metrics. Practitioners need to choose a GNN algorithm that is suitable for their objectives. From the viewpoint of research, though a few researches [109, 130] tackled a class imbalance problem on graphs, various and complicated settings, e.g., the combinations of class imbalance, heterophily property, and large-scale graphs, are still challenging. These complicated settings need to be addressed towards real applications.

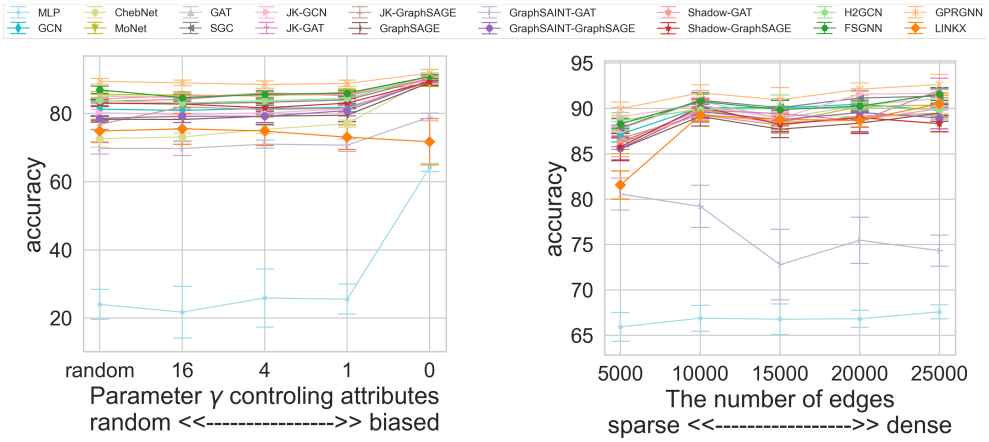


Figure 4.22: Accuracy on graphs with various attribute values.

Figure 4.23: Accuracy on graphs with various graph sizes.

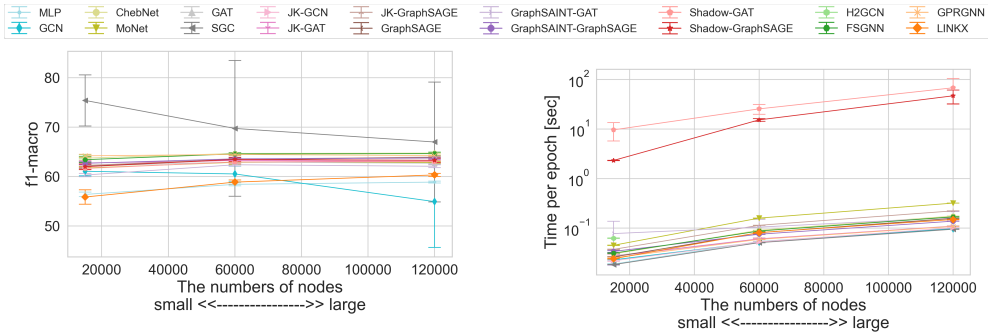
4.5.6 Experiments for Large Datasets

We explored small datasets in our empirical studies. So, we also provide experimental results using larger datasets in Figure 4.24. We set pairs of the numbers of nodes and edges (n, m) to $[(60000, 100000), (120000, 200000)]$. Note that a graph with 120000 nodes is a size of the same magnitude as ogbn-arxiv (169343 nodes). Due to time limitations, we use the hyperparameter tuned to graphs with 15000 nodes and 25000 edges and also plot the results of the graphs in the figure for comparison. Figure 4.24a shows f1-macro scores and Figure 4.24b shows training time per epoch.

Observations. First, H2GCN suffers from out-of-memory on large graphs since it computes a high-order adjacency matrix, i.e., an exact 2-hop away matrix, which requires the power of the adjacency matrix. Then, the results of other most GNNs on large graphs have similar tendencies to those on small graphs.

4.5.7 Summary, Open Questions, and Limitations

Thanks to a variety of synthetic graphs with controlled characteristics enabled by GenCAT, our empirical study reveals several interesting findings of GNNs that may be helpful for developing future algorithms, while limited benchmark datasets cannot provide such fine-grained analysis. We also



(a) Classification performance (f1-macro).

(b) Training time per epoch.

Figure 4.24: Experiments on large graphs.

provide an open-sourced PyTorch-based library to foster future research on GNNs.

Open Questions

Through our empirical studies, we have identified several open questions to developing new GNNs for node classification.

Class imbalance. We demonstrated that a class imbalance problem deteriorates the classification performance of most GNNs. This is because their loss functions are not designed to maximize classification performance in a class imbalance setting. Though a few studies [95, 109, 130] have addressed a class imbalance problem on a node classification task with GNNs, their main focus is homophilic and small-scale graphs. Hence, it is still an open question how to develop GNNs that work well in various and complicated settings such as the combinations of class imbalance, heterophily property, and large-scale graphs.

Heterophily setting. Recent GNNs [23, 82, 135] have been proposed to support homophilic and heterophilic graphs. However, the performance of MLP on the strong heterophily setting is comparable to those of such GNNs (see the leftmost points in Figure 4.11). This indicates that they almost ignore the topology information from heterophilic graphs and there is room for performance improvement in the heterophily setting. Hence, it is an open question how to develop GNNs that can capture the class structure

from heterophilic graphs while achieving the state-of-the-art performance on homophilic graphs.

We hope our experimental results motivate researchers to develop new GNN algorithms that address the above open questions, since they are crucial to apply GNN algorithms to real applications.

Limitations

In this empirical study, we focus on only a single task, node classification, though it is one of the hottest tasks of GNNs. By focusing on node classification, we can provide deep analysis that clarifies the pros/cons of GNNs. Also, this study does not support edge directions, edge labels, and time-series. Several recent studies [48, 77, 125] focus on developing GNNs that work on heterogeneous graphs in which objects of different types interact with each other in various ways. It would be interesting to conduct experiments with various heterogeneous graph datasets and benchmark such GNNs.

4.6 Related Work

4.6.1 Synthetic Graph Generator

There is a rich literature on graph generation (e.g., [9, 13, 20, 39, 70]). In this section, we first review five types of graph generators; traditional generators, generators for graphs with community structure, generators for graphs with community structure and node attributes, generators for large-scale graphs, and neural network-based graph generators. As described in Table 4.1, we can see the advantages of GenCAT relative to the state of the art. Then, we discuss the relationship between graph generators and a null model [1] that has an unbiasedly random structure.

Existing graph generator

Traditional graph generator. While many traditional graph generators have been proposed such as Erdős-Rényi [32], Barabasi-Albert [8], Chung-Lu [5], and BTER [100], they cannot control the class structure in generated graphs. As for Erdős-Rényi, users can specify only edge density for a whole graph. Barabasi-Albert assumes that degree distributions follow power law

distributions and implements a preferential attachment process so that generated graphs have power law node degree distributions. Chung-Lu aims to recreate a given node degree sequence. BTER controls degree distributions and cluster coefficient in generated graphs. In summary, these graph generators explicitly control edge density or node degree distributions but ignore the class structure in generated graphs.

Generator for graphs with community structure. This type of graph generators takes into account not only the topological characteristic of a complex graph (e.g., power law node degree) but also topological structures within communities. The LFR-benchmark [67] is designed to evaluate community mining algorithms. This assumes that the distributions of node degrees and community sizes follow power law distributions. The LFR-benchmark is extended to generate synthetic graphs with overlapping communities (i.e., nodes belong to multiple communities) [99] and hierarchical communities [120]. DC-SBM [45, 54] supports controlling the proportions of connections between classes. However, since this type of graph generators does not support node attributes, it cannot capture relationships between attributes and topological structures. In addition, these generators only consider the class-level connection proportions, so nodes in a class have the same node-class membership proportions.

Generator for graphs with community structure and node attributes. There are few generators that take both community structure and node attributes into account. This type of graph generators generates edges between nodes according to the similarity of their attributes. ANC [68] is a generator for attributed graphs with community structure, and DANCer [16] is an extended generator of ANC for generating dynamic graphs. In the design of ANC, the community structure of generated graphs only depends on the attributes. This indicates that it cannot flexibly generate a variety of graphs because users cannot explicitly control the connection proportions for each class. In GenCAT, users can flexibly control the connection proportions between nodes and classes in graphs.

Generators for large-scale graphs. There are a number of generators for large-scale graphs [13, 20, 70, 92]. gMark [12, 13], pgMark [14], and TrillionG [92] are schema-driven graph generation methods that support class labels and edge predicates. pgMark allows users to flexibly generate graphs by leveraging an optional schema definition, called a graph configuration, and supports node attributes. TrillionG can generate large-scale graphs ef-

ficiently by leveraging a recursive vector model. However, the generators cannot explicitly take community structure into account.

Neural network-based graph generators. Recently, neural network-based graph generators [19, 60, 102, 103, 118, 124] have been developed to reproduce real-world graphs. VGAE [60] and GraphVAE [103] construct generative models by leveraging a variational autoencoder. The main idea of them is that they consist of a graph encoder of GCN and a decoder that outputs an adjacency matrix. Also, NetGAN [19] is proposed to learn the generation of walks from biased random walks instead of graphs. However, the existing graph generators aim to reproduce synthetic graphs from given input graphs, so these generators cannot flexibly generate various graphs because users cannot explicitly control the characteristics of generated graphs. Additionally, they cannot generate large-scale graphs due to the large training time of their models.

The use of null models in the study of graph generators

A typical way to analyze empirical data is to compare it with a randomized version of the data, often called *null model* [1]. As for a graph generation problem, given the same numbers of nodes and edges as the original data, the edges in a null model are generated unbiasedly in terms of node degrees and classes, i.e., all nodes have similar degrees and the same connection proportions for all classes.

As we showed in our experiments, GenCAT can flexibly control the node degrees and class structure in generated graphs and the generated graphs can be biased by the node degrees and class structure. In this sense, graphs generated by GenCAT are different from the null model. Moreover, GenCAT can mimic a null model by appropriately setting its parameter, i.e., we set node degrees to the same value for all nodes and set class preference means to the same value for all pairs of classes. Graphs generated by other existing graph generators also differ from the null model in their focus. For example, LFR controls the ratio of intra- and inter-edges so the connection proportions between classes are biased in its generated graphs, unlike a null model. As for the traditional graph generators, they can generate graphs biased by user-specified node degree, and/or cluster coefficient. In summary, the use of a null model enables us to understand how generated graphs are biased compared with their randomized versions.

4.6.2 Empirical Study for GNNs

Several studies [30, 38, 90] have addressed benchmarking GNNs. A study [30] supports a variety of graph tasks, i.e., node classification, graph classification, link prediction, and graph regression. However, since only one or two datasets are used for each task, no deep analysis for node classification is provided. GraphWorld [90] provides limited insights for node classification with GNNs because it ignores three aspects. First, it ignores recent GNNs [23, 72, 82, 135] generalizing to heterophilic graphs, which have attracted much attention from the community. Second, the study does not care about class size distributions though they largely affect classification results. Note that a class imbalance problem has been explored in the machine learning field including graph mining [95, 109, 130]. Third, GraphWorld has not explored the efficiency of GNNs, which is one of the most common concerns of machine learning methods. Another study [38] focuses on GNNs for materials chemistry. Due to the characteristics of the data, the study considers only graph regression.

In summary, no work has extensively conducted experiments for node classification with GNNs in order to clarify their applicability/limitations by using graphs with various characteristics.

4.7 Conclusion

Toward the comprehensive evaluation of GNNs with graphs having various characteristics, in this chapter we proposed a flexible graph generator GenCAT and conducted empirical studies of GNNs by using the generator. As for the graph generator, we validated that it can generate high-quality graphs with controlled characteristics and scales linearly to the number of edges. As for the empirical study, we clarified the pros/cons of GNNs and raised open questions for current GNNs. We hope this work offers interesting insights for future research.

Chapter 5

Concluding Remarks

5.1 Summary of This Thesis

In this thesis, we have performed studies with the goal of providing effective and efficient graph machine learning methods and an evaluation framework for such methods. We tackled three significant limitations of current graph machine learning methods: hardly leveraging class structure, limited scalability, and no comprehensive evaluation.

In Chapter 1, we first clarified the limitations of existing works on graph machine learning methods for node classification towards real applications. Then, we described the organization of this thesis.

In Chapter 2, we proposed an effective attributed graph clustering method that bridges the topology space and the attribute space. The proposed method captures a non-linear relationship between the topology and attributes and the possibility of missing positive edges in a given graph while existing clustering methods ignore them. Thanks to the non-linearity and positive unlabeled learning in the loss function, the proposed method achieved better clustering results than existing methods.

In Chapter 3, we proposed a GNN transformation framework that automatically rewrites the formulation of a given non-scalable GNN into that of the scalable version of the given GNN. Also, we proposed a block-wise precomputation scheme for further acceleration of precomputation-based GNNs. Existing studies addressing the scalability issue of GNNs typically proposed their specific model architectures. Though their basic idea is the same, it is burdensome to apply the idea to existing non-scalable GNNs that have

been widely studied. To address this, the proposed framework automatically transforms non-scalable GNNs into scalable GNNs.

In Chapter 4, we proposed a flexible synthetic graph generator that supports the class structure in attributed graphs and then conducted empirical studies of GNNs for node classification by using the proposed generator. While existing works evaluate GNNs with limited benchmark datasets, our proposed generator and evaluation framework allow users to investigate how largely each characteristic of graphs affects node classification results.

In summary, the techniques and frameworks proposed in this thesis successfully provides baseline comparisons of graph machine learning methods and reduce the burden of researchers and developers for evaluating their methods from various aspects. We hope our proposals will be beneficial for future research on graph machine learning methods.

5.2 Future Work

Through the advancements made in this thesis, we identify several challenging research opportunities regarding the three limitations in this section, which are hardly leveraging class structure, limited scalability, and no comprehensive evaluation. We also discuss future research opportunities regarding the rest of several limitations as we described in Chapter 1, which are limited data types, limited support for time-series of graphs, and no general-purpose pre-trained models.

5.2.1 Class Structure

Our proposed method in Chapter 2 incorporates non-linearity and PU learning into the model to capture the interplay between classes, attributes, and topology. However, in real-world graphs, the contributions of attributes and topology to classification results may vary. In this sense, models should adaptively control the parameter balancing the effects from the attributes and topology in the learning step. How to develop such adaptive algorithms is still an open question.

5.2.2 Scalability

In Chapter 3, we proposed a framework improving the efficiency and scalability of existing non-scalable GNNs. Though we focused on a single machine and single thread to execute GNN algorithms, distributed/parallelized computation can accelerate the execution time of GNNs. Hence, it is our future work to combine our proposed approach and distributed computation for further scale-up of graphs.

5.2.3 Evaluation

Though we focused on node classification which is one of the hottest topics, other tasks such as link prediction and graph classification are also important. They are useful for many applications, e.g., link prediction is used to predict the interactions between proteins and graph classification is used to predict the functions of molecules. Similarly to node classification, comprehensive evaluations using various graphs are required for the other tasks. Also in this thesis, our empirical studies focus on a traditional setting in which a whole graph and node attributes can be stored in a GPU memory at the same time. It is our future work to address benchmarking scalable GNNs in terms of efficiency and effectiveness on large-scale graphs.

5.2.4 Data Type

Though in this thesis, we focused on homogeneous graphs where nodes represent a single object and edges represent a single relationship between nodes, graphs potentially include more flexible representations because nodes and edges can represent multiple types. Considering practical use cases, it is important to incorporate various objects and various relationships into a single graph, i.e., a heterogeneous graph. Several studies [48, 77, 125] have addressed developing graph neural networks for heterogeneous graphs. However, a study [77] clarified that such methods do not significantly outperform existing GNNs ignoring node/edge types in terms of classification quality. This implies that there is room for improvement since heterogeneous graphs are more informative than homogeneous graphs due to their node/edge types. Also, an evaluation framework for such graphs is required for future research.

5.2.5 Time-series of Graphs

All graphs are potentially time-series graphs. The timestamps of edges may positively affect the model prediction performance. Hence, time-series analysis on graphs has attracted attention since it is useful for practical settings such as recommendation and anomaly detection. Several studies [53, 97, 108] have addressed developing GNNs for graphs with time-series information. However, they do not effectively leverage node attributes, i.e., node attributes are not used in their experiments. Moreover, they suffer from poor scalability. In their experiments, graphs with up to 64,000 nodes are used, which are not large graphs for real applications. Hence, an interesting direction would be to effectively incorporate time-series information into GNN architectures while keeping their efficiency reasonable.

5.2.6 General-purpose Pre-trained Models for Graphs

We need to train graph machine learning methods so that they fit a given graph from scratch. To this end, we require large computational resources and execution time if the graph is large-scale, which is usual in practical settings. In the NLP domain, pre-trained language models [26] largely mitigate computational costs and training labels to tune the models for various downstream tasks. So, pre-trained models allow users who have relatively little computational resources to develop their new algorithms. However, there are no general-purpose pre-trained models in the graph domain. This is because graphs can potentially represent various phenomena, e.g., texts are node sequences and images are grids, leading to little common background knowledge. Hence, it is still an open question of what is essential information across different graphs.

Acknowledgment

First, I am indebted to my supervisor, Prof. Makoto Onizuka, who gave me plenty of opportunities to study abroad and always encouraged me to go ahead. Definitely, he positively changed my life. This thesis and all other papers I wrote would never have been completed without his unfailingly wise support. I owe a very important debt to Associate Prof. Yuki Arase who provided me valuable advice and guidance. I would like to express my deepest appreciation to Associate Prof. Chuan Xiao who gave me insightful guidance to my research work. I am deeply grateful to Assistant Prof. Yuya Sasaki who frequently had discussions with me and suggested an exciting direction every time. I would especially appreciate his daily assistance. My deepest appreciation goes to Prof. George Fletcher (Eindhoven University of Technology) who accepted my two-month visit to his university and continuously collaborate with me since then. Through my collaboration with him, I have gained confidence in my ability to work outside of Japan.

I would like to thank the committee members of my thesis, Prof. Takahiro Hara, Prof. Kaname Harumoto, Prof. Toru Fujiwara, Prof. Yasuyuki Matsushita, and Prof. Shinji Shimojo at the Department of Multimedia Engineering of the Graduate School of Information Science and Technology of Osaka University. Their careful comments improved the quality of the thesis.

I am very grateful for my team members including people who already graduated. Fun discussions with them inspired me to make good research progress. I also would like to thank the talented students and the secretary at Onizuka laboratory. It was my pleasure to study and work with them.

My heartfelt appreciation goes to my family including my mother, father, grandparents, uncle, aunt, and cousins. I would like to specially thank my brothers, Shoichi and Takeru. They have been my aspirations since I was born. Without their help and encouragement, I could never have decided to start my Ph.D. program.

Reference

[1] Katharina A. Zweig. *Network Analysis Literacy*. Springer Vienna, 2016.

[2] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

[3] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[4] Tameem Adel, Isabel Valera, Zoubin Ghahramani, and Adrian Weller. One-network adversarial fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2412–2420, 2019.

[5] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180, 2000.

[6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[7] Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 439–450. SIAM, 2012.

[8] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[9] Renzo Angles, Peter Boncz, Josep Larriba-Pey, Irini Fundulaki, Thomas Neumann, Orri Erling, Peter Neubauer, Norbert Martinez-Bazan, Venelin Kotsev, and Ioan Toma. The linked data benchmark council: a graph and rdf industry benchmarking effort. *ACM SIGMOD Record*, 43(1):27–31, 2014.

[10] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[11] Feras M Awaysheh, Mamoun Alazab, Sahil Garg, Dusit Niyato, and Christos Verikoukis. Big data resource management & networks: Taxonomy, survey, and future directions. *IEEE Communications Surveys & Tutorials*, 2021.

[12] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. Generating flexible workloads for graph databases. *Proceedings of VLDB*, 9(13):1457–1460, 2016.

[13] Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George Fletcher, Aurélien Lemay, and Nicky Advokaat. gMark: Schema-driven generation of graphs and queries. *IEEE Transactions on knowledge and data engineering*, 29(4):856–869, 2017.

[14] Guillaume Bagan, Angela Bonifati, Radu Ciucanu, George Fletcher, Aurélien Lemay, and Nicky Advokaat. The pgMark repository. *Github*, `https://github.com/ThomHurks/pgMark`, 2018.

[15] Amir Ben-Dor and Zohar Yakhini. Clustering gene expression patterns. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 33–42, 1999.

[16] Oualid Benyahia, Christine Largeron, Baptiste Jeudy, and Osmar R Zaïane. Dancer: Dynamic attributed network with community structure generator. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 41–44. Springer, 2016.

[17] Matthias Boehm, Michael W Dusenberry, Deron Eriksson, Alexandre V Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R Reiss, Prithviraj Sen, Arvind C Surve, et al. Systemml: Declarative machine learning on spark. *Proceedings of the VLDB Endowment*, 9(13):1425–1436, 2016.

[18] Aleksandar Bojchevski and Stephan Günnemann. Bayesian Robust Attributed Graph Clustering: Joint Learning of Partial Anomalies and Group Structure. In *Proceedings of AAAI*, 2018.

[19] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International conference on machine learning*, pages 610–619. PMLR, 2018.

[20] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM, 2004.

[21] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[22] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[23] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

[24] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[25] Pieter Delobelle, Paul Temple, Gilles Perrouin, Benoît Frénay, Patrick Heymans, and Bettina Berendt. Ethical adversaries: Towards mitigating unfairness with adversarial machine learning. *ACM SIGKDD Explorations Newsletter*, 23(1):32–41, 2021.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *North American Chapter of the Association for Computational Linguistics*, 2018.

[27] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[28] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135, 2006.

[29] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. In *Proceedings of Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

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

[31] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220, 2008.

[32] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[33] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[34] Gary William Flake, Steve Lawrence, C Lee Giles, and Frans M Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–70, 2002.

[35] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[36] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1433–1445, 2018.

[37] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

[38] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby G Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7(1):1–8, 2021.

[39] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[40] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *10th USENIX symposium on operating systems design and implementation (OSDI 12)*, pages 17–30, 2012.

[41] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

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

[43] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269, 2008.

[44] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[45] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[46] Cho-Jui Hsieh, Nagarajan Natarajan, and Inderjit Dhillon. Pu learning for matrix completion. In *International conference on machine learning*, pages 2445–2453. PMLR, 2015.

[47] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020.

[48] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020.

[49] Xiao Huang, Jundong Li, and Xia Hu. Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 633–641. SIAM, 2017.

[50] Zhichao Huang, Yunming Ye, Xutao Li, Feng Liu, and Huajie Chen. Joint weighted nonnegative matrix factorization for mining attributed graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 368–380. Springer, 2017.

[51] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[52] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5308–5317, 2016.

[53] Ming Jin, Yuan-Fang Li, and Shirui Pan. Neural temporal walks: Motif-aware representation learning on continuous-time dynamic graphs. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[54] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[55] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[56] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[57] George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[58] C Maria Keet, W Dubitzky, O Wolkenhauer, KH Cho, and H Yokota. Open world assumption. *Encyclopedia of Systems Biology*, pages 1567–1567, 2013.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[60] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

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

[62] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

[63] Da Kuang, Chris Ding, and Haesun Park. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 106–117. SIAM, 2012.

[64] Da Kuang, Sangwoon Yun, and Haesun Park. SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering. *Journal of Global Optimization*, 62(3):545–574, 2015.

[65] Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semi-supervised graph clustering: a kernel approach. *Machine learning*, 74(1):1–22, 2009.

[66] Krishna Kumar P., Paul Langton, and Wolfgang Gatterbauer. Factorized graph representations for semi-supervised learning from sparse data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1383–1398, New York, NY, USA, 2020. Association for Computing Machinery.

[67] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

[68] Christine Largeron, Pierre-Nicolas Mougel, Reihaneh Rabbany, and Osmar R Zaïane. Generating attributed networks with communities. *PloS one*, 10(4):e0122777, 2015.

[69] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[70] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of machine learning research*, 11(Feb):985–1042, 2010.

[71] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. Community detection in attributed graphs: An embedding approach. In *AAAI Conference on Artificial Intelligence*, 2018.

[72] Derek Lim, Felix Matthew Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Prasad Bhalerao, and Ser-Nam Lim. Large scale

learning on non-homophilous graphs: New benchmarks and strong simple methods. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[73] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. New benchmarks for learning on non-homophilous graphs. *In Workshop on Graph Learning Benchmarks (GLB 2021) at WWW*, 2021.

[74] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S Yu. Building text classifiers using positive and unlabeled examples. In *Third IEEE international conference on data mining*, pages 179–186. IEEE, 2003.

[75] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, apr 2012.

[76] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint*, 2021.

[77] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2021.

[78] Seiji Maekawa, Koki Noda, Yuya Sasaki, and Makoto Onizuka. Beyond real-world benchmark datasets: An empirical study of node classification with GNNs. In *Proceedings of Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[79] Seiji Maekawa, Yuya Sasaki, George Fletcher, and Makoto Onizuka. Gencat: Generating attributed graphs with controlled relationships between classes, attributes, and topology. *arXiv preprint arXiv:2109.04639*, 2021.

[80] Seiji Maekawa, Yuya Sasaki, George Fletcher, and Makoto Onizuka. Benchmarking gnns with gencat workbench. *Demo Track of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2022.

[81] Seiji Maekawa, Yuya Sasaki, George Fletcher, and Makoto Onizuka. GNN transformation framework for improving efficiency and scalability. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2022.

[82] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Improving graph neural networks with simple architecture design. *arXiv preprint*, 2021.

[83] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[84] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[85] Jan Motl and Oliver Schulte. The CTU prague relational learning repository. *CoRR*, abs/1511.03086, 2015.

[86] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., USA, 2010.

[87] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of National Academy of Sciences*, 103(23):8577–8582, 2006.

[88] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

[89] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *nature*, 435(7043):814–818, 2005.

[90] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. Graphworld: Fake graphs bring real insights for gnns. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and*

Data Mining, KDD '22, page 3691–3701, New York, NY, USA, 2022. Association for Computing Machinery.

[91] M. Parimala and Daphne Lopez. Graph clustering based on structural attribute neighborhood similarity (SANS). In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–4, 2015.

[92] Himchan Park and Min-Soo Kim. Trilliong: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 913–928, New York, NY, USA, 2017. Association for Computing Machinery.

[93] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[94] Zhi Qiao, Shuwen Liang, Hai Jiang, and Song Fu. A customizable mapreduce framework for complex data-intensive workflows on gpus. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2015.

[95] Liang Qu, Huaisheng Zhu, Ruiqi Zheng, Yuhui Shi, and Hongzhi Yin. Imgagn: Imbalanced network embedding via generative adversarial graph networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, page 1390–1398, New York, NY, USA, 2021. Association for Computing Machinery.

[96] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint*, 2019.

[97] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. 2020.

[98] Maekawa Seiji, Takeuchi Koh, and Onizuka Makoto. New attributed graph clustering by bridging attribute and topology spaces. *Journal of Information Processing (TOD)*, 28, 2020.

[99] Neha Sengupta, Michael Hamann, and Dorothea Wagner. Benchmark generator for dynamic overlapping communities in networks. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 415–424, 2017.

[100] Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. Community structure and scale-free collections of erdős-rényi graphs. *Physical Review E*, 85(5):056109, 2012.

[101] Martin Sevenich, Sungpack Hong, Oskar van Rest, Zhe Wu, Jayanta Banerjee, and Hassan Chafi. Using domain-specific languages for analytic graph databases. *Proc. VLDB Endow.*, 9(13):1257–1268, sep 2016.

[102] Han Shi, Haozheng Fan, and James T Kwok. Effective decoding in graph auto-encoder using triadic closure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 906–913, 2020.

[103] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pages 412–422. Springer, 2018.

[104] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

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

[106] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint*, 2017.

[107] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th annual international conference on machine learning*, pages 1073–1080, 2009.

[108] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. In *International Conference on Learning Representations*, 2021.

[109] Zheng Wang, Xiaojun Ye, Chaokun Wang, Jian Cui, and S Yu Philip. Network embedding with completely-imbalanced labels. *IEEE Transactions on knowledge and data engineering*, 33(11), 2020.

[110] Boris Weisfeiler and A. Lehmann, A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, 2(9):12–16*, 1968.

[111] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019.

[112] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 478–487. JMLR.org, 2016.

[113] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? 2019.

[114] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Kenichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018.

[115] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833, 2007.

[116] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. A model-based approach to attributed graph clustering. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, page 505–516, New York, NY, USA, 2012. Association for Computing Machinery.

[117] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. GBAGC: A general bayesian framework for attributed graph clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(1):1–43, 2014.

[118] Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. Conditional structure generation through graph variational generative adversarial nets. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[119] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network representation learning with rich text information. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 2111–2117. AAAI Press, 2015.

[120] Zhao Yang, Juan I Perotti, and Claudio J Tessone. Hierarchical benchmark graphs for testing community detection algorithms. *Physical review E*, 96(5):052311, 2017.

[121] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 40–48. JMLR.org, 2016.

[122] Fanghua Ye, Chuan Chen, and Zibin Zheng. Deep autoencoder-like nonnegative matrix factorization for community detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 1393–1402, New York, NY, USA, 2018. Association for Computing Machinery.

[123] Fanghua Ye, Chuan Chen, and Zibin Zheng. Deep autoencoder-like nonnegative matrix factorization for community detection. In *Proceedings of CIKM*, pages 1393–1402, 2018.

[124] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5694–5703. PMLR, 2018.

[125] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970, 2019.

[126] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor K. Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. *CoRR*, abs/2012.01380, 2020.

[127] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

[128] Ge Zhang, Di Jin, Jian Gao, Pengfei Jiao, Francoise Fogelman-Soulié, and Xin Huang. Finding communities with hierarchical semantics by distinguishing general and specialized topics. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 3648–3654. AAAI Press, 2018.

[129] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. Anrl: Attributed network representation learning via deep neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 3155–3161. AAAI Press, 2018.

[130] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 833–841, New York, NY, USA, 2021. Association for Computing Machinery.

[131] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

[132] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*, 2(1):718–729, aug 2009.

[133] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Clustering large attributed graphs: An efficient incremental approach. In *2010 IEEE International Conference on Data Mining*, pages 689–698, 2010.

[134] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. *arXiv preprint*, 2020.

[135] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[136] 前川政司, George Fletcher, and 鬼塚真. コミュニティ構造を考慮した属性付きグラフ汎用生成機構. **第 *11* 回データ工学と情報マネジメントに関するフォーラム（*DEIM Forum)***, 2019.

[137] 前川政司, 佐々木勇和, George Fletcher, and 鬼塚真. コミュニティ構造を制御する属性付きグラフ生成. **第 *13* 回データ工学と情報マネジメントに関するフォーラム（*DEIM Forum)***, 2021.

[138] 前川政司, 佐々木勇和, and 鬼塚真. コミュニティ構造を制御可能な属性付きグラフ生成. **情報処理学会第 *83* 回全国大会**, 2021.

[139] 前川政司, 竹内孝, 佐々木勇和, and 鬼塚真. 属性付きグラフのための非線形関数を用いた接合加重非負値行列分解. **第 *10* 回データ工学と情報マネジメントに関するフォーラム（*DEIM Forum)***, 2018.