Why Does Dropping Edges Usually Outperform Adding Edges in Graph Contrastive Learning?

Yanchen Xu[1,2*], **Siqi Huang**[1,2*], **Hongyuan Zhang**[2,3†], **Xuelong Li**[2†]

[1] School of Artificial Intelligence, OPtics and ElectroNics (iOPEN), Northwestern Polytechnical University
[2]Institute of Artificial Intelligence (TeleAI), China Telecom
[3]The University of Hong Kong
{yanchenxu.tj, 4777huang, hyzhang98}@gmail.com, xuelong_li@ieee.org

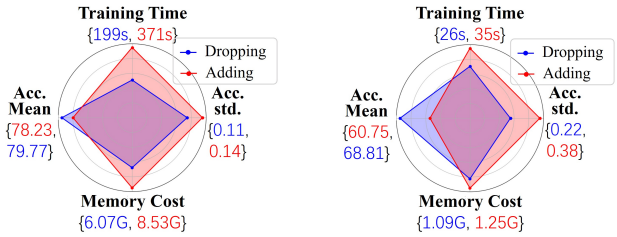
Abstract

Graph contrastive learning (GCL) has been widely used as an effective self-supervised learning method for graph representation learning. However, how to apply adequate and stable graph augmentation to generating proper views for contrastive learning remains an essential problem. Dropping edges is a primary augmentation in GCL while adding edges is not a common method due to its unstable performance. To our best knowledge, there is no theoretical analysis to study why dropping edges usually outperforms adding edges. To answer this question, we introduce a new metric, namely Error Passing Rate (EPR), to quantify how a graph fits the network. Inspired by the theoretical conclusions and the idea of positive-incentive noise, we propose a novel GCL algorithm, Error-PAssing-based Graph Contrastive Learning (EPAGCL), which uses both edge adding and edge dropping as its augmentations. To be specific, we generate views by adding and dropping edges based on the weights derived from EPR. Extensive experiments on various real-world datasets are conducted to validate the correctness of our theoretical analysis and the effectiveness of our proposed algorithm.

Code — https://github.com/hyzhang98/EPAGCL
Extended version — https://arxiv.org/abs/2412.08128

1 Introduction

Graph contrastive learning (GCL) has been a hot research topic in graph representation learning (Zhu et al. 2021; Thakoor et al. 2022; Zhang et al. 2022; Zhang, Zhu, and Li 2024). It originates from contrastive learning (Chen et al. 2020b; He et al. 2020; Gao, Yao, and Chen 2021; Qu et al. 2021; Radford et al. 2021), which generates different augmented views and maximizes the similarity between positive pairs (van den Oord, Li, and Vinyals 2018). In particular, compared with CL, it has been shown that how to generate proper graph augmentation views is a crucial problem in GCL (Tian et al. 2020; Wu et al. 2020). Due to the particularity of the structure of graph data, the reliable unsupervised data augmentation schemes are greatly limited. Inspired by Dropout (Srivastava et al. 2014), many researchers chose to randomly mask the features as the attribute augmentations. Moreover, for many GCL methods (Zhu et al. 2020; Thakoor et al. 2022; You et al. 2020), the most popular scheme is to change the graph topology, most of which randomly drops edges to generate views. Apart from random edge-dropping (Rong et al. 2020), researchers have tried various methods to generate stable views. MVGRL (Hassani and Khasahmadi 2020) introduced diffusion kernels for augmentation. GCA (Zhu et al. 2021) augmented the graph based on the importance of edges. And GCS (Wei et al. 2023) adaptively screened the semantic-related substructure in graphs for contrastive learning. Although various works have been done to investigate augmentation schemes for GCL, edge adding, as a simple way of graph topology augmentation, is only used in supervised learning (Zhao et al. 2021; Chen et al. 2020a). **Why is edge dropping a widely accepted way for GCL other than edge adding?** Can edge adding be modified to achieve similar effect as edge dropping? In this paper, we attempt to answer the above questions theoretically.



(a) Performance on WikiCS (b) Performance on CiteSeer

Figure 1: Performance on WikiCS and CiteSeer. All the hyper-parameters are fixed. Edge adding and edge dropping are the only augmentation employed respectively.

To begin with, a simple experiment is conducted on WikiCS (Mernyei and Cangea 2020) and CiteSeer (Sen et al. 2008) to compare the two strategies of augmentations. We use the same hyper-parameters and employ Adding Edges or Dropping Edges as the only augmentation, respectively. The result is shown in Figure 1. It is easy to conclude the disadvantage of edge adding in all respects.

Generally speaking, there are two main challenges for edge adding as augmentation:

*: Equal Contribution.
†: Corresponding authors.

$\mathcal{C}1$ For most graphs, the number of edges takes only a small portion of all node pairs, which means that there are much more choices for edge-adding than edge-dropping. Hence, the memory and time burden of edge-adding are significant.

$\mathcal{C}2$ Without labels, we have no idea about the influence of the edge perturbation. In that case, a suitable metric is needed to measure it.

To deal with the challenges above, we introduce a metric of graph, namely Error Passing Rate (EPR), to quantify the fitness between the graph and Graph Neural Network (GNN). To be specific, EPR measures *how much message is wrongly passed during the aggregation in GNNs*. For a certain graph, a lower EPR means that the aggregation is more efficient, which is what we expect. Consequently, *the aim of augmentation can be converted to generating a view with low EPR*.

In this paper, through mathematical derivation, we prove that *the degree attribute of two nodes can measure the effect on EPR of adding or dropping the edge between them*. To maintain a relatively low EPR, it is important to ensure that the EPR of the graph will not increase too much even if an edge is wrongly added or dropped. To this end, for stability, node pairs that correspond to low-level effect will be chosen to add or drop the edges between them, as a solution to $\mathcal{C}2$. As a result, the nodes can be pre-screened to reduce the memory and time burden, which resolves $\mathcal{C}1$.

Inspired by our theoretical analysis, we propose a novel adaptive augmentation framework for GCL, where a graph is augmented through selective edge perturbation and random feature mask. For edge perturbation, the possibility that an edge is added to or dropped from the graph is based on the magnitude of its effect on EPR. Briefly speaking, the possibility is high if the magnitude of effect is relatively low. This helps to maintain the EPR of augmented views in a low level. The augmentation is then equipped with an InfoNCE-like objective (van den Oord, Li, and Vinyals 2018; Zhu et al. 2020) for contrastive learning, namely Error-PAssing-rate-based Graph Contrastive Learning (EPAGCL).

2 Related Works

2.1 Graph Contrastive Learning

Various graph contrastive learning methods have been proposed (Gao et al. 2021; Zhang et al. 2021; Liu et al. 2023) in recent years. In general, Graph Contrastive Learning (GCL) methods aims to learn representations by contrasting positive and negative samples.

The researchers mainly focus on the contrastive objective. For the sake of simplicity, random edge dropping and feature masking ars widely used (Rong et al. 2020; Zhu et al. 2020; Thakoor et al. 2022). Furthermore, GraphCL (You et al. 2020) and InfoGCL (Xu et al. 2021) employ two more augmentations, node dropping and subgraph sampling, which change the node attribute and structure property of the graph at the same time. There are also some methods using their own augmentation methods. MVGRL (Hassani and Khasahmadi 2020) generates views through graph diffusion. SimGCL (Yu et al. 2022) adds uniform noise to the embedding space to create contrastive views.

2.2 Adaptive Data Augmentations for Graph

Some researchers have investigated methods for adaptive data augmentations (Li 2022; Zhang et al. 2024) for GCL. JOAO (You et al. 2021) adaptively and dynamically selects data augmentations for specific data. GCA (Zhu et al. 2021) sets the probability of dropping edges and masking features according to their importance. ADGCL (Suresh et al. 2021) optimizes adversarial graph augmentation strategies for contrastive learning. GCS (Wei et al. 2023) screens substructure in graphs with the help of a gradient-based graph contrastive saliency. NCLA (Shen et al. 2023) learns graph augmentations by multi-head graph attention mechanism.

Our proposed augmentation method is similar to GCA. However, edges that EPAGCL tends to add or drop are likely to be preserved in GCA. What's more, our theory is rigorously derived quantitatively, and our augmentation scheme is proved to be more stable.

2.3 Adding Edges as Augmentation

Adding edges is also used as augmentation for graph data is some supervised learning methods. AdaEdge (Chen et al. 2020a) and GAUG (Zhao et al. 2021) both add edges based on prediction metrics, which cannot be used in self-supervised learning.

Both methods tend to minimize the negative impact of the augmentation, which is the same as our proposed framework. To be specific, both of them reduce the probability of error augmentation, while EPAGCL, inspired by pi-noise (a theoretical framework to learn beneficial noise) (Li 2022), reduces the magnitude of the impact if an error occurs.

3 The Proposed Method

In this section, we first introduce the formal definition of Error Passing Rate (EPR) to quantify how a graph fits the GNN. Then, the augmentations of graph topology are interpreted based on EPR. Finally, the Error-PAssing-based Graph Contrastive Learning (EPAGCL) method induced by the theoretical analysis is elaborated. The Framework is illustrated in Figure 2.

3.1 Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote an undirected graph without isolated points, whose node set is $\mathcal{V} = \{v_1, v_2, \cdots, v_N\}$ and edge set is $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The adjacency matrix of $\mathcal{G}$ is denoted as $A \in \mathbb{R}^{N \times N}$ with $A_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and $A_{ij} = 0$ otherwise. The degree matrix of $\mathcal{G}$, denoted as D, is a diagonal matrix with $D_{ii} = \sum_j A_{ij}$. Furthermore, let $\widetilde{D} = D + I$, $\widetilde{A} = A + I$, where I is an N-dimensional identity matrix. $\forall v \in \mathcal{V}$, the set of its neighbours in graph $\mathcal{G}$ is denoted as N_v with $|N_v| = d_v = D_{vv}$. Let $d_{min} = \min_{v \in V} d_v$, $d_{max} = \max_{v \in V} d_v$.

For $v_i, v_j \in \mathcal{V}$ $s.t.\,(v_i, v_j) \notin \mathcal{E}$, suppose that adding edges (v_i, v_j) to graph $\mathcal{G}$ yields graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$, where $\mathcal{E}' = \mathcal{E} \cup \{(v_i, v_j)\}$. Relatively, A', D', $\widetilde{A}'$, and $\widetilde{D}'$ of graph $\mathcal{G}'$ are defined in the same way as above.

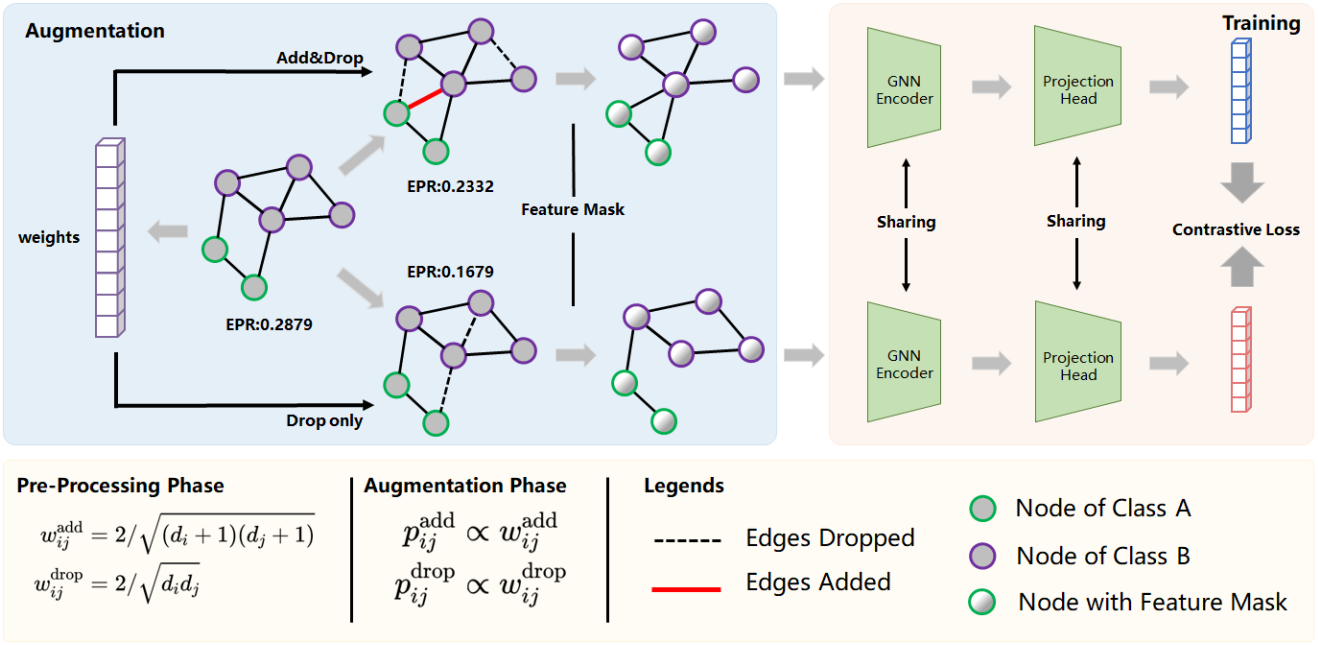


Figure 2: Framework of EPAGCL: Before training, the weight of all existing edges and candidate edge for adding is computed according to the graph structure. We then generate two views adaptively based on the weights. Specifically, we add edges to and drop edges from the graph to obtain one view while drop edges only from the graph to obtain another. A random feature mask is then employed. After that, the two views are fed to a shared Graph Neural Network (GNN) with a projection head for representation learning. The model is trained with a contrastive objective.

3.2 Error Passing Rate

As message passing is a popular paradigm for GNNs, we introduce Error Passing Rate based on message passing mechanisms to measure how the graph fits the network:

Definition 3.1. For a graph $\mathcal{G}$, the Error Passing Rate (EPR) $r_{\mathcal{G}}$ denotes the ratio of the amount of message wrongly passed in the network, i.e.,

$$r_{\mathcal{G}} = M_{wp}/M,$$

where M_{wp} is the amount of message wrongly passed, while M is the amount of all the message passed.

Note that $r_{\mathcal{G}}$ will change with different network structure. In this paper, Graph Convolutional Networks (GCN) (Kipf and Welling 2017) are taken as the target. The feed forward propagation in GCN can be recursivly conducted as

$$H^{(l+1)} = \sum(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}H^{(l)}W^{(l)}), \qquad (1)$$

where $H^{(l+1)} = \{h_1^{(l+1)}, \cdots, h_N^{(l+1)}\}$ is the hidden vector of the $(l+1)$-th layer with $h_i^{(l)}$ as the hidden feature of node v_i, and $\sum(\cdot)$ is a nonlinear function. Eq. (1) suggests that the topological structure of the graph is represented as $\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2} \triangleq \hat{A}$ in the network, corresponding to the message-passing mechanism. Specifically, the amount of message passed from node v_i to v_j can be measured by $\hat{A}_{ij}$. Thus, we can obtain the representation of $r_{\mathcal{G}}$.

Proposition 3.2. Given an undirected graph $\mathcal{G}$, its EPR $r_{\mathcal{G}}$ in GCN can be formulated as

$$r_{\mathcal{G}} = \sum_{(v_i,v_j)\in E} \hat{A}_{ij} / \sum_{(v_i,v_j)\in \mathcal{E}} \hat{A}_{ij}, \qquad (2)$$

where E denotes the set of edges in $\mathcal{E}$ that links nodes of different, underlying classes which are agnostical during the self-supervised training, namely error edge set.

3.3 Why is Adding Edges Usually Worse?

An ordinary idea is to persue a smaller $r_{\mathcal{G}}$. This naturally leads to a question: *how will $r_{\mathcal{G}}$ change after adding (v_i, v_j) into graph $\mathcal{G}$ as an augmentation?*

It is complex to calculate the change of $r_{\mathcal{G}}$ with Eq. (2). To simplify the calculation, we group the edges according to their relationship with v_i and v_j, between which no edge lies. To be specific, for graph $\mathcal{G}$, we divide $\mathcal{E}$ into three parts:

$$\mathcal{E}_1 = \{(u,v)|u,v \in V\backslash\{v_i,v_j\}\} \cap \mathcal{E},$$
$$\mathcal{E}_2 = \{(v_i,w)|w \in V\} \cap \mathcal{E},$$
$$\mathcal{E}_3 = \{(v_j,w)|w \in V\} \cap \mathcal{E}.$$

Note that $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3 = \mathcal{E}$. Similarly, the error edge set E is also divided into three parts, E_1, E_2, and E_3. As a result, $\mathcal{E}_1$ and E_1 are sets of edges whose endpoints are not v_i or v_j; $\mathcal{E}_2$ and E_2 are edge sets with v_i as an endpoint; $\mathcal{E}_3$ and E_3 are edge sets with v_j as an endpoint.

As for graph $\mathcal{G}'$ which yields from adding edge (v_i, v_j) to $\mathcal{G}$, the split is completed in the same way: $\mathcal{E}' = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{E}_3 \cup$

$\mathcal{E}_{ij}$, $E' = E_1 \cup E_2 \cup E_3 \cup E_{ij}$. Among them, $\mathcal{E}_{ij} = \{(v_i, v_j)\}$, $E_{ij} = \mathcal{E}_{ij}$ if v_i and v_j are of different classes and $E_{ij} = \varnothing$ if they are of the same class.

Owing to the split, the $\sum_{(i,j)} \hat{A}_{ij}$ in Eq. (2) can be simplified. To begin with, denote the normalized adjacency matrix of $\mathcal{G}'$, $\widetilde{D}'^{-1/2}\widetilde{A}'\widetilde{D}'^{-1/2}$ as $\hat{A}'$. Furthermore, define

$$m_l = \sum_{(v_i,v_j)\in\mathcal{E}_l} \hat{A}_{ij},\ e_l = \sum_{(v_i,v_j)\in E_l} \hat{A}_{ij},\ l = 1, 2, 3,$$

and

$$m'_l = \sum_{(v_i,v_j)\in\mathcal{E}_l} \hat{A}'_{ij},\ e'_l = \sum_{(v_i,v_j)\in E_l} \hat{A}'_{ij},\ l = 1, 2, 3.$$

Thus, $r_{\mathcal{G}}$ and $r_{\mathcal{G}'}$ can be rewritten as

$$r_{\mathcal{G}} = \frac{\sum_{i=1}^{3} e_i}{\sum_{i=1}^{3} m_i},\ r_{\mathcal{G}'} = \frac{\sum_{i=1}^{3} e'_i + \sum_{(v_i,v_j)\in E_{ij}} \hat{A}'_{ij}}{\sum_{i=1}^{3} m'_i + \sum_{(v_i,v_j)\in\mathcal{E}_{ij}} \hat{A}'_{ij}}.$$

Considering the change in node degrees in graph $\mathcal{G}$ after adding edge (v_i, v_j) to it, it is easy to know that each node except v_i and v_j maintains its own degree while the degrees of v_i and v_j are increased by 1, indicating that

$$\begin{cases} m'_1 = m_1, \\ m'_2 = \sqrt{\frac{d_i}{d_i+1}} m_2, \\ m'_3 = \sqrt{\frac{d_j}{d_j+1}} m_3, \\ \hat{A}'_{ij} = \hat{A}'_{ji} = \frac{1}{\sqrt{(d_i+1)(d_j+1)}}. \end{cases}$$

As a result, $r_{\mathcal{G}}$ and $r_{\mathcal{G}'}$ can be formulated as

$$\begin{aligned} r_{\mathcal{G}} &= \frac{e_1 + e_2 + e_3}{m_1 + m_2 + m_3}, \\ r_{\mathcal{G}'} &= \frac{e_1 + \sqrt{\frac{d_i}{d_i+1}} e_2 + \sqrt{\frac{d_j}{d_j+1}} e_3 + \xi \cdot \frac{2}{\sqrt{(d_i+1)(d_j+1)}}}{m_1 + \sqrt{\frac{d_i}{d_i+1}} m_2 + \sqrt{\frac{d_j}{d_j+1}} m_3 + \frac{2}{\sqrt{(d_i+1)(d_j+1)}}}, \end{aligned} \tag{3}$$

where $\xi = 0$ if v_i and v_j are of the same class and $\xi = 1$ otherwise.

Let $m = \sum_{i=1}^{3} m_i$, $m' = \sum_{i=1}^{3} m'_i + \frac{2}{\sqrt{(d_i+1)(d_j+1)}}$, which are the sum of elements in $\hat{A}$ and $\hat{A}'$, respectively. To compare $r_{\mathcal{G}}$ with $r_{\mathcal{G}'}$, we introduce

$$\delta_{\mathcal{G},\mathcal{G}'} = m \cdot m' \cdot (r_{\mathcal{G}} - r_{\mathcal{G}'}). \tag{4}$$

As $\delta_{\mathcal{G},\mathcal{G}'} \propto (r_{\mathcal{G}} - r_{\mathcal{G}'})$, it can effectively reflects the trends of changes in EPR. With a simple constraint, $\delta_{\mathcal{G},\mathcal{G}'}$ can be related to the edge added, which is formally summarzied as Theorem 3.3.

Theorem 3.3. Given a graph $\mathcal{G}$ with $d_{max} \leq 4d_{min} - 1$ and $d_{min} \geq 1$, then:

1. If $\mathcal{G}'$ yields from adding an edge between two same-class nodes to $\mathcal{G}$ (i.e., $\xi = 0$), then $\delta_{\mathcal{G},\mathcal{G}'} > 0$.
2. If $\mathcal{G}'$ yields from adding an edge between two different-class nodes to $\mathcal{G}$ (i.e., $\xi = 1$), then $\delta_{\mathcal{G},\mathcal{G}'} < 0$.

The proof of Theorem 3.3 is in Appendix A.

Theorem 3.3 indicates that adding edge between same-class nodes decreases EPR of the graph, while adding edge between different-class nodes leads to the opposite result. The conclusion also meets our expectation of EPR and edge adding as well.

Moreover, motivating by the proof of Theorem 3.3.2, an additional assumption is used to precisely quantify $\delta_{\mathcal{G},\mathcal{G}'}$. With the assumption, we can further compare the effect of edge adding and dropping.

Assumption 3.4. For each node in the graph $\mathcal{G}$, constantly k of all the message passed in is error message, i.e.

$$M_{wp,i}/M_i = k\ ,\ \forall i \in \{1, 2, \cdots, N\},$$

where $M_{wp,i}$ is the amount of message wrongly passed to v_i, while M_i is the amount of all the message passed to v_i.

With Assumption 3.4, it can be derived that

$$\delta_{\mathcal{G},\mathcal{G}'} = (k - \xi) \cdot \alpha_{i,j} \cdot m, \tag{5}$$

where $\alpha_{i,j} = 2/\sqrt{(d_i+1)(d_j+1)}$. The detailed derivation is in Appendix B.

Now we apply the above analysis to edge dropping to answer the question raised in the title of this subsection. Specifically, we may consider $\mathcal{G}'$ as the original graph and $\mathcal{G}$ as the graph augmented by dropping edge (v_i, v_j). Hence, for edge dropping, it holds that

$$\delta_{\mathcal{G}',\mathcal{G}} = (\xi - k) \cdot \alpha_{i,j} \cdot m. \tag{6}$$

The detailed derivation is in Appendix C. Hence, the following conclusions are drawn from Eq. (4), Eq. (5), and Eq. (6):

Theorem 3.5. Under Assumption 3.4, if graph $\mathcal{G}$ is obtained through adding edge (v_i, v_j) to $\mathcal{G}$, it holds that

$$\begin{cases} r_{\mathcal{G}'} - r_{\mathcal{G}} = \frac{k}{m'} \cdot \alpha_{i,j} & \text{if } c(v_i) = c(v_j), \\ r_{\mathcal{G}'} - r_{\mathcal{G}} = \frac{1-k}{m'} \cdot \alpha_{i,j} & \text{if } c(v_i) \neq c(v_j), \end{cases}$$

where $c(v)$ denotes the class of node v.

For most graphs, the EPR is less than 0.5, i.e., $k < 0.5$, which ***explains why dropping edges is more stable than adding edges for most of the time***. What's more, as mentioned in Section 2.2, GCA tends to drop edges with low importance, which corresponds to a higher $\alpha_{i,j}$ and leads to a more unstable result according to Theorem 3.5.

3.4 Adding Edges with Retaining EPR

As shown in Section 3.3, adding edges is usually worse than dropping edges. Since the edge dropping may not work when the graph is too sparse, a direct question is that *whether edge adding can work like edge dropping*? According to Theorem 3.5, for dropping edge, the change of EPR $\propto \alpha_{i,j}$, while for adding edges, the change of EPR $\propto \alpha_{i,j}/m' \approx \alpha_{i,j}/m \propto \alpha_{i,j}$.

Following the theoretical conclusion, we propose Error-PAssing-based Graph Contrastive Learning (EPAGCL),

Algorithm 1: Algorithm to select added edges

Input: Vertex set $\mathcal{V}$, edge set $\mathcal{E}$.
Output: Drop-edge weight w^d, add-edge weight w^a, set of edges to be added $\mathcal{E}_a$.

1: $l \leftarrow |\mathcal{E}|$
2: **for** $e_{i,j}$ in $\mathcal{E}$ **do**
3: $\quad w^d_{ij} \leftarrow \alpha_{i,j;\,\text{drop}}$ // $\alpha_{i,j}$ of $(\mathcal{V}, \mathcal{E})$ as $\mathcal{G}'$.
4: **end for**
5: $\mathcal{V}_a \leftarrow$ vertex in $\mathcal{V}$ of top $\sqrt{2l}$ degrees
6: $\mathcal{E}_a \leftarrow \mathcal{V}_a \times \mathcal{V}_a - \mathcal{E}$ // $l \le |\mathcal{E}_a| \le 2l$.
7: **for** $e_{i,j}$ in $\mathcal{E}_a$ **do**
8: $\quad w^a_{ij} \leftarrow \alpha_{i,j;\,\text{add}}$ // $\alpha_{i,j}$ of $(\mathcal{V}, \mathcal{E})$ as $\mathcal{G}$.
9: **end for**
10: **return** w^d, w^a, $\mathcal{E}_a$

which generates views for GCL based on the $\alpha_{i,j}$ corresponding to each edge, ensuring that the EPR of the graph will not increase too much even if the edge is wrongly added or dropped.

To be specific, edges are added or dropped by sampling from the corresponding probability (Zhu et al. 2020). The edge set $\widetilde{\mathcal{E}}$ of the generated view can be formulated as $\widetilde{\mathcal{E}} = \widetilde{E} \cup \widetilde{E}'$, with probability

$$P\{(v_i, v_j) \in \widetilde{E}\} = 1 - p^d_{ij} \tag{7}$$

and

$$P\{(v_i, v_j) \in \widetilde{E}'\} = p^a_{ij}, \tag{8}$$

where $\widetilde{E}$ is a subset of the original edge set $\mathcal{E}$ and $\widetilde{E}'$ is a subset of the to-be-added edge set $\mathcal{E}_a$. p^a_{ij} and p^d_{ij} stand for the probability of dropping and adding (v_i, v_j) respectively. Algorithm 1 shows how to get the to-be-added edge set $\mathcal{E}_a$ along with the weights w_d and w_a. Note that for edges to be dropped, $\alpha_{i,j} = 2/\sqrt{d_i d_j}$, while for edges to be added, $\alpha_{i,j} = 2/\sqrt{(d_i+1)(d_j+1)}$ (refer to Appendix C). The weights are then transformed into probability through a normalization step (Zhu et al. 2021).

$$\begin{cases} p^a_{ij} = \min\left(\dfrac{\max(w^a) - w^a_{ij}}{\max(w^a) - \mu_{w^a}} \cdot p_{\text{add}},\ p_\tau\right), \\ p^d_{ij} = \min\left(\dfrac{w^d_{ij} - \min(w^d)}{\mu_{w^d} - \min(w^d)} \cdot p_{\text{drop}},\ p'_\tau\right). \end{cases} \tag{9}$$

In Eq. (9), p_{add} and p_{drop} are hyper-parameters that controls the overall probability. p_τ, p'_τ are cut-off probability that is no greater than 1. μ_{w^a} and μ_{w^d} stand for the average of w^a and w^d, respectively.

Note that the weights is obtained based on the original graph, they will be computed only once, which adds almost nothing to the burden. As for graphs with a significant number of nodes, thanks to the nodes filtering steps (line 5 in Algorithm 1), the computation is greatly accelerated and can be finished within an acceptable timeframe.

The training algorithm is summarized as pseudo-code in Algorithm 2. As Theorem 3.5 shows, dropping edges is a more stable way to generate augmented views, so it is used

Algorithm 2: The EPAGCL training algorithm

Input: Original graph $G = (V, E)$ with feature X, weights w^a, w^d, to-be-added edge set $\mathcal{E}_a$.

1: **for** epoch $\leftarrow 1, 2, \cdots$ **do**
2: $\quad$ Obtain p^a_{ij}, p^d_{ij} according to w^a, w^d through Eq. (9).
3: $\quad$ Obtain E_1, E_2 according to p^d_{ij}, E through Eq. (7).
4: $\quad$ Obtain E' according to $p^a_{ij}, \mathcal{E}_a$ through Eq. (8).
5: $\quad$ Apply random mask on feature X and get X_1, X_2.
6: $\quad$ Two views are generated: $G_1 = (V, E_1)$ with feature X_1 and $G_2 = (V, E_2 \cup E')$ with feature X_2.
7: $\quad$ Compute the contrastive loss $\mathcal{L}$ between G_1 and G_2.
8: $\quad$ Update the parameters to minimize $\mathcal{L}$.
9: **end for**

when generating both the views, while adding edges is used only for generating one views. Other than edge perturbation, random feature mask, which is widely used in graph presentation learning (Zhu et al. 2020) (Hassani and Khasahmadi 2020), is also employed. After the views are generated, an InfoNCE-like objective (van den Oord, Li, and Vinyals 2018) is employed. For each positive pair (u_i, v_i) in G_1, G_2, which is the embedding corresponds the same node of the original graph, we define $s(u_i, v_i)$ as the cosine similarity of $g(u_i)$ and $g(v_i)$, where $g(\cdot)$ is a projection head (Tschannen et al. 2020), and

$$l(u_i, v_i) = \log \frac{e^{s(u_i, v_i)/\tau}}{e^{s(u_i, v_i)/\tau} + \sum\limits_{i \neq j} e^{s(u_i, v_j)/\tau} + \sum\limits_{i \neq j} e^{s(u_i, u_j)/\tau}}, \tag{10}$$

where τ is a temperature parameter. The contrastive loss is then computed by added up $l(u_i, v_i)$ and $l(v_i, u_i)$ for all $i \in \{1, 2, \cdots, N\}$.

4 Experiments

In this section, we perform experiments to investigate the following questions:

Q1 Does EPAGCL outperforms the existing baseline methods on node classification?

Q2 What is the time and memory burden of EPAGCL?

Q3 How does each part of the proposed augmentation strategy affect the effectiveness of training?

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,278	1,433	7
CiteSeer	3,327	4,552	3,703	6
PubMed	19,717	44,324	500	3
WikiCS	11,701	216,123	300	10
Amazon-Photo	7,650	119,081	745	8
Coauthor-Physics	34,493	247,962	8,415	5
Ogbn-Arxiv	169,343	1,166,243	128	40

Table 1: Statistics of datasets used in experiments.

Method	Cora	CiteSeer	PubMed	WikiCS	Amz. Photo	Co. Physics	ogbn-arxiv	Rank
GCN	84.15 ± 0.31	72.00 ± 0.39	85.39 ± 0.28	79.93 ± 0.53	92.69 ± 0.26	95.14 ± 0.41	69.44 ± 0.06	-
GAT	84.08 ± 0.51	72.17 ± 0.39	84.91 ± 0.20	80.84 ± 0.21	92.39 ± 0.30	95.48 ± 0.14	OOM	-
DGI	82.47 ± 0.38	71.03 ± 0.88	85.64 ± 0.39	79.43 ± 0.27	91.00 ± 0.37	OOM	OOM	7.8
GMI	82.90 ± 0.69	69.51 ± 0.72	83.37 ± 0.52	80.23 ± 0.29	90.88 ± 0.31	OOM	OOM	8.6
MVGRL	81.46 ± 0.40	70.78 ± 0.53	84.33 ± 0.25	80.10 ± 0.26	90.91 ± 0.68	94.91 ± 0.16	OOM	8.2
GRACE	85.34 ± 0.29	71.66 ± 0.35	$\underline{86.74 \pm 0.18}$	80.65 ± 0.20	93.13 ± 0.12	95.63 ± 0.05	68.49 ± 0.01	3.6
GCA*	84.55 ± 0.43	70.81 ± 0.57	86.48 ± 0.17	$\underline{81.45 \pm 0.15}$	93.08 ± 0.18	95.20 ± 0.09	$\underline{69.26 \pm 0.01}$	4.6
GCA	$\underline{85.59 \pm 0.35}$	71.21 ± 0.55	86.60 ± 0.19	79.18 ± 0.34	93.19 ± 0.24	95.28 ± 0.19	69.18 ± 0.01	4.7
BGRL	84.34 ± 0.36	70.02 ± 0.70	85.88 ± 0.14	80.43 ± 0.47	92.78 ± 0.65	95.56 ± 0.07	68.80 ± 0.10	6.1
GREET	80.42 ± 0.25	71.48 ± 0.99	86.27 ± 0.32	79.91 ± 0.50	$\mathbf{93.56 \pm 0.14}$	$\mathbf{96.06 \pm 0.11}$	OOM	5.0
EPAGCL*	85.04 ± 0.33	$\mathbf{71.97 \pm 0.62}$	86.72 ± 0.11	$\mathbf{81.81 \pm 0.18}$	93.05 ± 0.23	95.41 ± 0.03	$\mathbf{69.29 \pm 0.01}$	$\underline{3.0}$
EPAGCL	$\mathbf{86.07 \pm 0.32}$	$\underline{71.94 \pm 0.57}$	$\mathbf{86.77 \pm 0.14}$	81.19 ± 0.11	$\underline{93.42 \pm 0.12}$	$\underline{95.87 \pm 0.04}$	69.25 ± 0.01	**2.0**

Table 2: Results in terms of classification accuracy (in percent $\pm$ standard deviation) on seven datasets. * means that we do not employ feature mask to augment graph data. OOM indicates Out-Of-Memory on a 40GB GPU. The best and runner-up results of self-supervised methods on each dataset are highlighted with **bold** and underline, respectively.

4.1 Datasets

Seven benchmark graph datasets are utilized for experimental study, including three citation network **Cora**, **CiteSeer** and **PubMed** (Sen et al. 2008), a reference network **WikiCS** (Mernyei and Cangea 2020), a co-purchase network **Amazon-Photo** (Shchur et al. 2018), a co-authorship network **Coauthor-Physics** (Shchur et al. 2018) and a large-scale citation network **ogbn-arxiv** (Hu et al. 2020). The details of the datasets are summarized in Table 1.

4.2 Experimental Settings

Backbone For our proposed method, a two-layer GCN network (Kipf and Welling 2017) with PReLU activation is applied. The dimension of the hidden layer is 512 and the dimension of the final embedding is set as 256. Also, we employ a projection head, which consists of a 256-dimension fully connected layer with ReLU activation and a 256-dimension linear layer. The hyper-parameters of the training vary for different datasets, the details of which are shown in Appendix D.

Baselines We compare EPAGCL with two groups of baseline methods: (1) semi-supervised learning methods (i.e., **GCN** (Kipf and Welling 2017) and **GAT** (Veličković et al. 2018)); (2) contrastive learning methods (i.e., **DGI** (Veličković et al. 2019), **GMI** (Peng et al. 2020), **MVGRL** (Hassani and Khasahmadi 2020), **GRACE** (Zhu et al. 2020), **GCA** (Zhu et al. 2021), **BGRL** (Thakoor et al. 2022), and **GREET** (Liu et al. 2023)).

Evaluation Settings To evaluate the proposed method, we follow the standard linear evaluation scheme introduced in (Veličković et al. 2019). Firstly, each model is trained in an unsupervised manner. The resulting embeddings are then utilized to train and test a simple l_2-regularized logistic regression classifier, which is initialized randomly and trained with an Adam SGD optimizer (Kingma and Ba 2015) for 3000 epochs. The learning rate and weight decay factor of the optimizer are fixed to 0.01 and 0.0 respectively.

For each method on each dataset, the accuracy is averaged over 5 runs. For each run, the dataset is randomly split, where 10%, 10% and the rest 80% of the nodes are selected for the training, validation and test set, respectively.

The experiments are conducted on an NVIDIA A100 GPU with 40 GB memory.

4.3 Performance Analysis (Q1)

The classification accuracy is shown in Table 2 with a comparative rank. Specifically, despite the perturb target is different, our edge perturbation method is similar to GCA, while the feature mask is randomly applied. Further experiments are conducted for comparison. In the table, * means that we do not employ feature mask as one of the augmentations on graph data.

It's easy to find out that EPAGCL achieves better performance than baselines on almost every dataset. For instance, on Cora, our method achieves a 0.48% accuracy gain than the next best approach. What's more, with the same contrastive objective, EPAGCL outperforms GCA by an average 0.61% increase, which indicates that our augmentation method is more effective. Thirdly, our method shows a lower standard deviation than GCA on most datasets, revealing its stability. This also verifies the inference from Theorem 3.5 that dropping edges corresponding to higher $\alpha_{i,j}$ will lead to a more unstable result.

In additional, Figure 3 displays t-SNE (Van der Maaten and Hinton 2008) plots of the raw feature and learned embeddings on Cora and CiteSeer.

4.4 Efficiency Analysis (Q2)

To illustrate the efficiency of our model, we compare our method with other graph contrastive methods in terms of data pre-processing time, flops and training time of one epoch, and memory costs. MVGRL (Hassani and Khasahmadi 2020) is a method that makes use of graph diffusion for contrast, which is kind of similar to edge adding. GCA (Zhu et al. 2021) calculates different drop possibilities for edge dropping. And GREET (Liu et al. 2023) extracts information from feature and structure to benefit training. Experiments are conducted on Cora, PubMed, and ogbn-arxiv, corresponding to small, big and huge datasets. On ogbn-arxiv,

(a) Raw features of Cora (b) EPAGCL on Cora features (c) Raw features of CiteSeer (d) EPAGCL on CiteSeer features

Figure 3: t-SNE embeddings of the raw features and learned embeddings obtained through EPAGCL on Cora and CiteSeer.

Method	Cora				PubMed				ogbn-arxiv			
	Proc.	FLOPs	Time	Mem.	Proc.	FLOPs	Time	Mem.	Proc.	FLOPs	Time	Mem.
MVGRL	0.18s	5.55e6	0.03s	3.49G	15.88s	4.04e7	0.16s	21.25G	-	-	-	-
GCA	0.40s	3.57e8	0.04s	0.93G	0.14s	2.60e9	0.42s	13.87G	0.13s	2.23e10	3.89s	18.36G
GREET	4.46s	5.90e9	0.15s	1.47G	240.01s	2.18e10	6.38s	39.50G	15.95h	-	-	-
EPAGCL	0.40s	3.57e8	0.04s	0.90G	0.57s	2.60e9	0.42s	17.10G	21.94s	2.23e10	3.89s	22.93G

Table 3: Comparison in terms of data pre-processing time, flops and training time of one epoch, and memory costs between different graph contrastive methods. Proc. and Mem. stand for data processing time and Memroy cost, respectively. '-' indicates Out-Of-Memory on a 40GB GPU. On ogbn-arxiv, the model is trained with a batch size of 256 due to the memory limit.

the model is trained with a batch size of 256 because of the memory limit. The results are shown in Table 3.

The experiments show that although our methods takes more time to pre-process data and more memory for training than GCA, the additional burden is relatively small even when it is applied on a huge dataset like ogbn-arxiv. As for MVGRL and GREET, the pre-processing time and memory requirement are much higher. Specifically, on ogbn-arxiv, MVGRL reports Out-Of-Memory during the pre-processing phase. While GREET takes a lot of time to pre-process data and finally reports OOM when training in regardless of batch size. Moreover, the pre-processing time of EPAGCL grows in a slow rate with the increase of the scales of graph, which further indicates that EPAGCL also fits the huge datasets. What's more, EPAGCL also shows a relatively fast training speed, especially compared with GREET.

4.5 Ablation Study (Q3)

Further experiments are conducted to demonstrate the effect of augment strategies. We investigate the performance of the following six augmentations without feature mask on some benchmark datasets: randomly add and drop edges as 'Random Add'; add and drop edges adaptively on both views as 'Add to Both Views'; our method as 'EPAGCL'; drop edges adaptively as 'Drop Only'; drop edges randomly as 'Random Drop'; and add edges adaptively as 'Add Only'.

The results are shown in Figure 4. To better manifest the difference between the performance, we illustrate the performance improvement of the five augmentations compared with 'Random Add'. It can be observed that our method achieves the best performance on each dataset. Moreover, 'Drop Only' has an advantage over 'Add Only'. This indicates that adding edges is a relatively poorer augmentation

Figure 4: Performance improvement of five augmentation strategies compared to 'Random Add'.

way, which is consistent with our theory. Thirdly, the adaptive strategy 'Add to Both Views' leads to a great accuracy increase compared with the random strategy 'Random Add', which proves the effectiveness of our method.

To sum up, our adaptive augmentation strategy is effective for edge-dropping and edge-adding as well. And adding edges to only one view fully utilizes the augmentation.

5 Conclusion

In this paper, we propose a novel algorithm EPAGCL for GCL. The main idea of EPAGCL is to use adding edges as one of the augmentations to generate views. We firstly introduce Error-Passing Rate (EPR) to measure the quality of a view. Based on EPR, the magnitude of effect of edge perturbation is quantified. Thus, we are able to add edges adaptively with low time and memory burden and without labels, which is also valid for edge dropping. Extensive experiments validate the correctness of our theoretical and reveal the effectiveness of our algorithm.

References

Chen, D.; Lin, Y.; Li, W.; Li, P.; Zhou, J.; and Sun, X. 2020a. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 3438–3445.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020b. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, 1597–1607. PMLR.

Gao, T.; Yao, X.; and Chen, D. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Gao, Z.; Bhattacharya, S.; Zhang, L.; Blum, R. S.; Ribeiro, A.; and Sadler, B. M. 2021. Training robust graph neural networks with topology adaptive edge dropping.

Hassani, K.; and Khasahmadi, A. H. 2020. Contrastive multi-view representation learning on graphs. In *International conference on machine learning*, 4116–4126. PMLR.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9729–9738.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Li, X. 2022. Positive-incentive noise. *IEEE Transactions on Neural Networks and Learning Systems*.

Liu, Y.; Zheng, Y.; Zhang, D.; Lee, V. C.; and Pan, S. 2023. Beyond smoothing: Unsupervised graph representation learning with edge heterophily discriminating. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 4516–4524.

Mernyei, P.; and Cangea, C. 2020. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*.

Peng, Z.; Huang, W.; Luo, M.; Zheng, Q.; Rong, Y.; Xu, T.; and Huang, J. 2020. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, 259–270.

Qu, Y.; Shen, D.; Shen, Y.; Sajeev, S.; Chen, W.; and Han, J. 2021. Co{DA}: Contrast-enhanced and Diversity-promoting Data Augmentation for Natural Language Understanding. In *International Conference on Learning Representations*.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; Krueger, G.; and Sutskever, I. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, 8748–8763. PMLR.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Magazine*, 29(3): 93.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Shen, X.; Sun, D.; Pan, S.; Zhou, X.; and Yang, L. T. 2023. Neighbor contrastive learning on learnable graph augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, 9782–9791.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958.

Suresh, S.; Li, P.; Hao, C.; and Neville, J. 2021. Adversarial Graph Augmentation to Improve Graph Contrastive Learning. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 15920–15933. Curran Associates, Inc.

Thakoor, S.; Tallec, C.; Azar, M. G.; Azabou, M.; Dyer, E. L.; Munos, R.; Veličković, P.; and Valko, M. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *International Conference on Learning Representations*.

Tian, Y.; Sun, C.; Poole, B.; Krishnan, D.; Schmid, C.; and Isola, P. 2020. What makes for good views for contrastive learning? *Advances in neural information processing systems*, 33: 6827–6839.

Tschannen, M.; Djolonga, J.; Rubenstein, P. K.; Gelly, S.; and Lucic, M. 2020. On Mutual Information Maximization for Representation Learning. In *International Conference on Learning Representations*.

van den Oord, A.; Li, Y.; and Vinyals, O. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR*, abs/1807.03748.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep Graph Infomax. In *International Conference on Learning Representations*.

Wei, C.; Wang, Y.; Bai, B.; Ni, K.; Brady, D.; and Fang, L. 2023. Boosting graph contrastive learning via graph contrastive saliency. In *International conference on machine learning*, 36839–36855. PMLR.

Wu, M.; Zhuang, C.; Mosse, M.; Yamins, D.; and Goodman, N. 2020. On mutual information in contrastive learning for visual representations. *arXiv preprint arXiv:2005.13149*.

Xu, D.; Cheng, W.; Luo, D.; Chen, H.; and Zhang, X. 2021. Infogcl: Information-aware graph contrastive learning. *Advances in Neural Information Processing Systems*, 34: 30414–30425.

You, Y.; Chen, T.; Shen, Y.; and Wang, Z. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*, 12121–12132. PMLR.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33: 5812–5823.

Yu, J.; Yin, H.; Xia, X.; Chen, T.; Cui, L.; and Nguyen, Q. V. H. 2022. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, 1294–1303.

Zhang, H.; Shi, J.; Zhang, R.; and Li, X. 2022. Non-Graph Data Clustering via $\mathcal{O}(n)$ Bipartite Graph Convolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(7): 8729–8742.

Zhang, H.; Wu, Q.; Yan, J.; Wipf, D.; and Philip, S. Y. 2021. From canonical correlation analysis to self-supervised graph neural networks. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Zhang, H.; Xu, Y.; Huang, S.; and Li, X. 2024. Data Augmentation of Contrastive Learning is Estimating Positive-incentive Noise. *arXiv preprint arXiv:2408.09929*.

Zhang, H.; Zhu, Y.; and Li, X. 2024. Decouple Graph Neural Networks: Train Multiple Simple GNNs Simultaneously Instead of One. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O.; Jiang, M.; and Shah, N. 2021. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, 11015–11023.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2020. Deep Graph Contrastive Representation Learning. In *ICML Workshop on Graph Representation Learning and Beyond*.

Zhu, Y.; Xu, Y.; Yu, F.; Liu, Q.; Wu, S.; and Wang, L. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the web conference 2021*, 2069–2080.