# Graph Contrastive Learning with Generative Adversarial Network

Cheng Wu
wuc22@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Chaokun Wang*
chaokun@tsinghua.edu.cn
Tsinghua University
Beijing, China

Jingcao Xu
xjc20@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Ziyang Liu
liuzy21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Kai Zheng
zhengkai@kuaishou.com
Kuaishou Inc.
Beijing, China

Xiaowei Wang
wangxiaowei03@kuaishou.com
Kuaishou Inc.
Beijing, China

Yang Song
yangsong@kuaishou.com
Kuaishou Inc.
Beijing, China

Kun Gai
gai.kun@qq.com
Unaffiliated
Beijing, China



## ABSTRACT

Graph Neural Networks (GNNs) have demonstrated promising results on exploiting node representations for many downstream tasks through supervised end-to-end training. To deal with the widespread label scarcity issue in real-world applications, Graph Contrastive Learning (GCL) is leveraged to train GNNs with limited or even no labels by maximizing the mutual information between nodes in its augmented views generated from the original graph. However, the distribution of graphs remains unconsidered in view generation, resulting in the ignorance of *unseen* edges in most existing literature, which is empirically shown to be able to improve GCL's performance in our experiments. To this end, we propose to incorporate graph generative adversarial networks (GANs) to learn the distribution of views for GCL, in order to i) automatically capture the characteristic of graphs for augmentations, and ii) jointly train the graph GAN model and the GCL model. Specifically, we present GACN, a novel **G**enerative **A**dversarial **C**ontrastive learning **N**etwork for graph representation learning. GACN develops a view generator and a view discriminator to generate augmented views automatically in an adversarial style. Then, GACN leverages these views to train a GNN encoder with two carefully designed self-supervised learning losses, including the graph contrastive loss and the Bayesian personalized ranking Loss. Furthermore, we design an optimization framework to train all GACN modules jointly. Extensive experiments on seven real-world datasets show that GACN is able to generate high-quality augmented views for GCL and is superior to twelve state-of-the-art baseline methods. Noticeably, our proposed GACN surprisingly discovers that the generated views in data augmentation finally conform to the well-known *preferential attachment* rule in online networks.

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Neural networks**; • **Information systems** → **Data mining**; **Social networks**; **Retrieval models and ranking**.

*Chaokun Wang is the corresponding author.



*KDD '23, August 6–10, 2023, Long Beach, CA, USA*

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599370

## KEYWORDS

Graph Representation Learning, Graph Neural Networks, Generative Adversarial Network, Contrastive Learning

**ACM Reference Format:**
Cheng Wu, Chaokun Wang, Jingcao Xu, Ziyang Liu, Kai Zheng, Xiaowei Wang, Yang Song, and Kun Gai. 2023. Graph Contrastive Learning with Generative Adversarial Network. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3580305.3599370

## 1 INTRODUCTION

In recent years, graph representation learning has attracted increasing attention from both academia and industry to deal with network-based data [12, 13, 32, 35, 42]. Graph Neural Networks (GNNs) [14, 17, 23] have shown effectiveness in supervised end-to-end training. However, task-specific labels can be extremely scarce for graph datasets [21, 54]. To this end, research efforts start exploring self-supervised learning for GNNs, where only limited or even no labels are needed [43].

Recently, graph contrastive learning (GCL) [39, 43, 47, 52] has become one of the most popular self-supervised approaches, which leverages the mutual information maximization principle (InfoMax) [27] to maximize the correspondence between the representations of a graph (or a node) in its different augmented views. There are a large amount of view augmentation strategies explored by different GCL methods, including node dropping, edge perturbation, subgraph sampling and feature masking. Furthermore, views can be generated by random sampling [47], or under the guide of domain knowledge [15, 53], or by a view learner [34].

However, the evolution and the distribution of graphs remains unconsidered in existing view generation strategies. Intuitively, a graph is formed with nodes and edges created in succession, and the observed graph is a snapshot of this procedure. Thus, the non-connected node pairs are possible to form new edges in future. Case 1 in Figure 1 demonstrates this intuition. Furthermore, the evolution of graphs varies (e.g., Case1, Case2, or other possible cases in Figure 1) based on the distribution of graphs. Then, exploring the distribution of graphs helps to search unseen but should existing edges, which benefits the variety of generated views and boosts the performance of GCL.

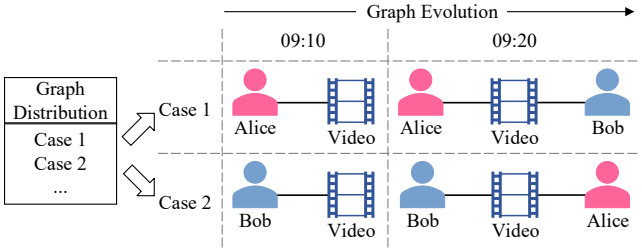



**Figure 1: A toy example of the evolution and the distribution of graphs. In detail, suppose a new video is uploaded at 09:00. Next, different observed graphs, such as Case 1 and Case 2, are sampled from a graph distribution and then evolve respectively. For instance, in Case 1 (Case 2), Alice (Bob) watches the video at 09:10 and Bob (Alice) watches it at 09:20. If the observed graph is obtained at 09:15 (09:17), the watching behavior from Bob (Alice) is unseen.**

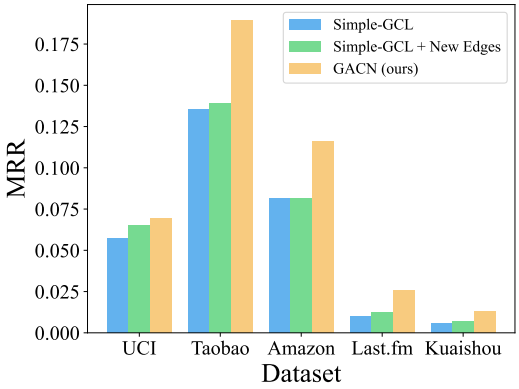



**Figure 2: Empirical experiments show that replacing some existing edges with random new edges in one of the augmented views can improve Simple-GCL on most datasets. However, it requires a trial-and-error selection of the new edge rate to get the best performance. In contrast, our GACN can automatically learn the graph distribution and precisely add edges for better graph representation learning.**

As illustrated in Figure 2, we consider replacing some existing edges with new edges randomly in one of the augmented views of a graph contrastive learning method (i.e., Simple-GCL in Sec 5.1.2) and evaluate the link prediction performance. It is observed that replacing a certain amount of existing edges with new edges can improve Simple-GCL on most datasets, showing the benefit of the supplement of unseen edges. However, different rates of new edges are required to get the best performance on different datasets due to different graph distributions.

In this work, we argue that the process of data augmentation for GCL should systematically consider graph evolution, and then propose to leverage graph generative adversarial networks (GANs) for graph distribution learning. Clearly, it is not a trivial task considering the following two major challenges:

**Automatically capturing the characteristic of graphs for augmentations.** On the one hand, compared to image data and text data, graph data are more abstract [34], making it hard to characterize the distribution of graph. On the other hand, graph data are discrete (e.g., the value of the adjacent matrix is binary), and sampling-based graph generators are usually hard to be trained end-to-end. Thus, more explorations are required to design a graph GAN to generate high-quality augmented views.

**Jointly training the graph GAN model and the GCL model.** A simple idea to combine graph GANs with GCL is to train the two models separately. However, in this way, the connection between the two models is weak and there is no guarantee that the generated views, which deceive the discriminator of the GAN model well, can be well encoded by the GCL model. Furthermore, training graph GAN and GCL separately needs to maintain two groups of GNN parameters, which is unnecessary. Thus, it is better to explore a parameter sharing strategy and a jointly learning framework for the sake of effectiveness and efficiency.

To tackle the above challenges, this paper proposes GACN, a graph **G**enerative **A**dversarial **C**ontrastive learning **N**etwork. Specifically, GACN develops a graph generative adversarial network with a view generator and a view discriminator to learn generating augmented views through a minimax game. Then, GACN adapts a GNN as the graph encoder and designs two self-supervised learning losses to optimize the parameters. To train GACN, a jointly learning framework is proposed to optimize the view generator, the view discriminator and the graph encoder sequentially and iteratively.

The main contributions of this paper are highlighted as follows:

- We explore the benefit of leveraging unseen edges to boost GCL, and first propose to incorporate graph GANs to learn and generate views for GCL.
- We present GACN, a new graph neural network that develops a view generator and a view discriminator to learn generating views for the graph encoder. All these modules are trained jointly with a novel framework (Section 4).
- We conduct comprehensive experiments to evaluate GACN with twelve state-of-the-art baseline methods. The experimental results show that GACN is superior to other methods, and is able to generate views satisfying the famous preferential attachment rule (Section 5).

## 2 RELATED WORK

In this section, the related work to this study is briefly summarized, including graph contrastive learning and graph generative adversarial network.

### 2.1 Graph Contrastive Learning

Contrastive Learning (CL) [3, 18, 19, 27, 37, 38] is an emerging paradigm to learn quality discriminative representations based on augmented ground-truth samples. It initially showed the promising capability in the field of computer vision (CV) and natural language processing (NLP) while recently researchers have applied CL to graph domains to fully exploit graph structure information and rich unlabeled data. The core idea of GCL is to maximize the mutual information between instances (e.g., node, subgraph, and graph) of different views augmented from the original graph.

Similar to the visual domain [6, 37], there are various augmentation techniques on attributes or topologies and contrastive pretext tasks on different granularities. For example, DGI [39] performs the row-wise shuffling on the attribute matrix and conducts node-graph level contrast while MVGRL [15] applies an edge diffusion augmentation to obtain contrasting views. On top of attribute masking, GraphCL [47] proposes several topology-based augmentations including node dropout, edge dropout and subgraph sampling to incorporate various priors. Rather than contrasting views at the graph level, GRACE [52], GCA [53] and GROC [22] conduct node-level same-scale contrast, which is the most adopted method to learn node-level representations. Very recently, JOAO [46] adopts a bi-level optimization framework to learn graph data augmentations.

However, many GCL methods require a trial-and-error selection or domain knowledge to augment views, which limits the application of GCL.

## 2.2 Graph Generative Adversarial Network

By designing a game-theoretical minimax game, Generative Adversarial Networks (GAN) [11] have achieved success in various applications, such as image generation [8], sequence generation [48], dialogue generation [26], information retrieval [41], and domain adaption [50]. More recently, GANs have been applied on graph-based tasks.

In terms of the graph generation task, Liu et al. [28] stack multiple GANs to form a hierarchical architecture for preserving topological features of training graphs. To preserve the distribution of links with minimal risk of privacy breaches, Tavakoli et al. [36] utilize GANs to learn the probability of link formation. Aiming at better capturing the essential properties and preserving the patterns of real graphs, Bojchevski et al. [4] introduce NetGAN to learn a distribution of network via the random walks.

Another line of applications is graph embedding. ANE [7] treats GANs as an regularization term to learn robust representations. GraphGAN [40] designs a generator to learn node embeddings and a discriminator to predict link probabilities. NetRA [49] and ProGAN [9] preserve and learn the underlying node similarity in the model of GAN. Besides, Lei et al. [25] and Yang et al. [45] combine GANs with various encoders to refine the performance of temporal link prediction. Sun et al. [33] introduce MEGAN for multi-view network embedding, which accounts for the information from individual views and correlations among different views.

Recently, GASSL [44] and AD-GCL [34] incorporate adversarial learning with graph contrastive learning to avoid capturing redundant information by optimizing adversarial graph augmentation strategies. In computer vision, researchers have tried to make a combination of GAN and CL to boost the performance of GAN [24] or CL [31]. However, in graph mining, adapting GANs to learn graph distribution and generate views for GCL remains unexplored.

# 3 PRELIMINARIES

In this section, we introduce some preliminary concepts and notations. In this work, we denote a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a node set and $\mathcal{E}$ is an edge set. $\mathcal{G}$ may have node attributes $\{\mathcal{X}_v \in \mathbb{R}^F | v \in \mathcal{V}\}$. The adjacent matrix of $\mathcal{G}$ is denoted as $A \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$,

**Table 1: Notations in this paper.**

| Notations | Definitions |
|---|---|
| $\mathcal{V}$ | the set of nodes of a graph |
| $\mathcal{E}$ | the set of edges of a graph, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ |
| $\mathcal{G}$ | a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ |
| $\mathcal{X}$ | the node attributes that $\mathcal{G}$ may have |
| $F$ | the dimension of node attributes |
| $D$ | the dimension of node representations |
| $A$ | the adjacent matrix of $\mathcal{G}$, $A \in \mathbb{R}^{\|\mathcal{V}\|\times\|\mathcal{V}\|}$ |
| $P$ | the approximation of the generated adjacent matrix |
| $d_v$ | the node representation of node $v \in \mathcal{V}$ |
| $\theta_g$ | the parameters of the view generator |
| $\theta_h$ | the parameters of the view discriminator |
| $\theta_f$ | the parameters of the graph encoder |

where:

$$A_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E}; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

**Graph Representation Learning.** Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the aim of graph representation learning is to learn an encoder $f : \mathcal{V} \to \mathbb{R}^D$, where $\{f(v)|v \in \mathcal{V}\}$ can be further used in downstream tasks, such as node classification and link prediction.

**Graph Neural Networks (GNNs).** In this work, we focus on using GNNs as the encoder $f$. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each node $v \in \mathcal{V}$ is paired with a representation $d_v$ initialized as $d_v^{(0)} = \mathcal{X}_v$. The idea is to apply the neighborhood aggregation scheme on $\mathcal{G}$, updating the representation of node by aggregating the representations of neighbor nodes:

$$d_v^{(l)} = AGG(d_v^{(l-1)}, \{d_u^{(l-1)} | u \in \mathcal{N}_v\}), \tag{2}$$

where $d_v^{(l)}$ denotes the representation of node $v$ in the $l$-th layer, $\mathcal{N}_v$ is the set of neighbors of node $v$, and $AGG$ is the aggregation function. After obtaining $L$ layers presentations, a readout function is adopted to generate the final representation of node $i$:

$$d_v = readout(\{d_v^{(l)} | l = \{0, \cdots, L\}\}). \tag{3}$$

**Graph Contrastive Learning (GCL).** GCL aims to maximize the mutual information between instances (e.g., node, subgraph and graph) of different views augmented from the original graph. Typically, GCL methods adopt graph augmentation strategies to construct positive pairs and negative pairs, and utilize GNNs to encode them into representations. Then, a contrastive loss function is defined to enforce maximizing the consistency between positive pairs compared with negative pairs.

In this paper, we use LightGCN [17] as the GNN encoder and focus on node-level GCL. Note that several important notations used in this paper are summarized in Table 1.

# 4 METHODS

In this section, we first present the overview of the proposed GACN model (see Figure 3), and then bring forward the details of its three modules. Finally, we propose the optimization framework of GACN.

## 4.1 Overview

As shown in Figure 3a, GACN consists of three modules, namely *View Generator*, *View Discriminator* and *Graph Encoder*. Specifically,

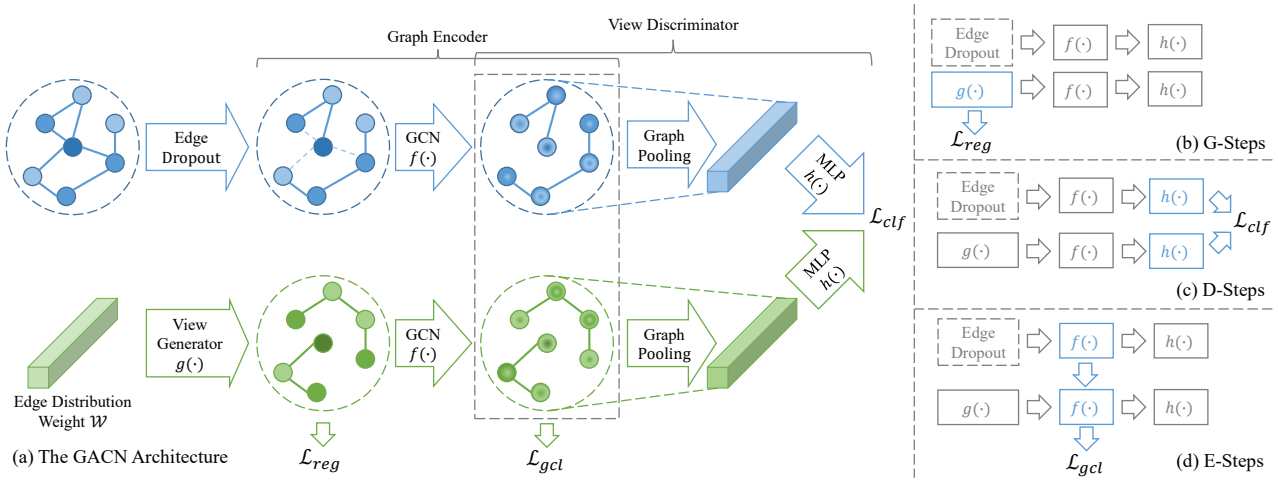



**Figure 3: The architecture and the training steps of GACN. There are three modules in GACN, including the view generator, the view discriminator and the graph encoder, which are optimized by the G-Steps, the D-Steps and the E-Steps, respectively.**

the view generator learns the distributions of edges and generates augmented views by edge sampling. Then the view discriminator is designed to distinguish views generated by the generator from those generated by predefined augmentation strategies (e.g., edge dropout). The view generator and the view discriminator are trained in an adversarial style to generate high-quality views. These views are used to train robust node representations in the graph encoder, which shares the same node representations with the view discriminator.

Note that we do not explicitly encode any graph generative principles into the model design. However, surprisingly our proposed GACN learns the graph distribution to generate views that follow the well-known preferential attachment rule [2] (see Section 5.5).

## 4.2 View Generator

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the view generator is designed to generate a set of augmented views. For a specific view $\mathcal{G}_g$, we assume that each edge $(v_i, v_j)$ in $\mathcal{G}_g$ is associated with a random variable $P_{i,j} \sim \text{Bernoulli}(\mathcal{W}_{i,j})$, where $\mathcal{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a learnable matrix, $P$ is a binary matrix with size $|\mathcal{V}| \times |\mathcal{V}|$, $(v_i, v_j)$ is in $\mathcal{G}_g$ if $P_{i,j} = 1$ and is dropped otherwise. To train the view generator in an end-to-end fashion, we relax the discrete $P_{i,j}$ to be a continuous variable in $(0, 1)$ as follows:

$$P = \sigma(\frac{\mathcal{W} - X_g}{\tau_g}), \tag{4}$$

where $X_g \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a random matrix with each element sampled from a uniform distribution: $X_{i,j} \sim U(0, 1)$, $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and $\tau_g \in (0, 1]$ is a hyper-parameter to make $P_{i,j}$ close to 0 or 1. Here, $P$ can be treated as an approximation of the generated adjacent matrix.

To constrain the structure of the generated views, we propose two regularization losses, namely the **Edge Count Loss** and the **New Edge Loss**, to train the parameters of the generator, i.e., $\Theta_g = \{\mathcal{W}\}$.

**Edge Count Loss.** This loss is designed to limit the number of edges in $\mathcal{G}_g$. Inspired by the edge-dropout strategy [52], we set a ratio $\lambda_g$ and train $\mathcal{W}$ to generate views with $\lambda_g \cdot |\mathcal{E}|$ edges. Formally, the edge count loss is computed as:

$$\mathcal{L}_{cnt} = |\lambda_g \cdot |\mathcal{E}| - \sum_{i,j} P_{i,j}|. \tag{5}$$

**New Edge Loss.** This loss is proposed to avoid generating views that are aggressively different from $\mathcal{G}$. Specifically, we calculate a penalty for each new edge in $\mathcal{G}_g$, i.e., edge $(v_i, v_j)$ with $A_{i,j} = 0$ and $P_{i,j} = 1$. Then, the new edge loss is the sum of all the penalties:

$$\mathcal{L}_{new} = \sum_{i,j} (1 - A_{i,j}) \cdot P_{i,j} \tag{6}$$

Then, the regularization loss is the combination of the above two losses:

$$\mathcal{L}_{reg} = \lambda_{cnt} \cdot \mathcal{L}_{cnt} + \lambda_{new} \cdot \mathcal{L}_{new}, \tag{7}$$

where $\lambda_{cnt}$ and $\lambda_{new}$ are hyper-parameters to balance the influences of $\mathcal{L}_{cnt}$ and $\mathcal{L}_{new}$, respectively.

Besides, for the sake of efficiency, we initialize $\mathcal{W}$ instead of training from scratch. Specifically, we set an initialization rate $\gamma \in [0, 1]$ to constrain the number of new edges at the beginning, i.e., $\gamma \cdot \lambda_g \cdot |\mathcal{E}|$ new edges and $(1-\gamma) \cdot \lambda_g \cdot |\mathcal{E}|$ existing edges are expected in the generated views. Thus, $\mathcal{W}$ is initialized as follows:

$$\mathcal{W}_{i,j} = \begin{cases} \frac{(1-\gamma)\cdot\lambda_g\cdot|\mathcal{E}|}{|\mathcal{E}|} & , \text{ if } (v_i, v_j) \in \mathcal{E}; \\ \frac{\gamma\cdot\lambda_g\cdot|\mathcal{E}|}{|C|} & , \text{ if } (v_i, v_j) \in C; \\ 0 & , \text{ otherwise,} \end{cases} \tag{8}$$

where $C \subseteq (\mathcal{V} \times \mathcal{V} - \mathcal{E})$ is a candidate set of new edges. Note that we do not consider all the possible new edges as candidates because maintaining a dense $\mathcal{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ for large graphs is memory-unfriendly. In our implementation, we choose edges related to nodes with top-2, 000 degrees as the candidate set. Note that we also try

top-5, 000 and top-10, 000. However, the performance gain of top-5, 000 or top-10, 000 over that of top-2, 000 is almost nothing with a huge increment of training time.

## 4.3 View Discriminator

The view discriminator is a graph-level classifier to recognize the generated views. More precisely, the discriminator takes an adjacent matrix as input and judges whether the matrix is a true matrix (i.e., a matrix generated by predefined augmentation strategies) or a false matrix (i.e., a matrix generated by the view generator). Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we denote the set of views generated by predefined augmentation strategies (i.e., edge dropout in this work) as $\vec{\mathcal{G}}_p$, and the set of views generated by the view generator as $\vec{\mathcal{G}}_g$. Thus, for each $G \in \vec{\mathcal{G}}_p \cup \vec{\mathcal{G}}_g$, a GNN encoder $f$ is used to encode the representations of each node:

$$\{d_v | v \in \mathcal{V}\} = f(G). \tag{9}$$

Then, we calculate the graph representation by concatenating the mean pooling and the maximum pooling of the node representations:

$$d_G = (\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} d_v) \oplus \text{MaxPool}(\{d_v | v \in \mathcal{V}\}), \tag{10}$$

where $\oplus$ is the concatenate operation. With the graph representation, we compute the probability of $G$ using an $L_h$-layer Multilayer Perceptron (MLP) $h$:

$$p_G = h(d_G), \tag{11}$$

where $\Theta_h = \{(W_i, b_i) | i = 1, 2, \ldots, L_h\}$ is the set of parameters in $h$.

To train the discriminator, we label the views in $\vec{\mathcal{G}}_p$ with 1 and the views in $\vec{\mathcal{G}}_g$ with 0. Suppose that the label of $G$ is $y_G$. Then, the classification loss is defined as follows:

$$\mathcal{L}_{clf} = -y_G \cdot \log(p_G) - (1 - y_G) \cdot \log(1 - p_G). \tag{12}$$

## 4.4 Graph Encoder

The graph encoder is designed to learn the node representations, i.e., the set of parameters of the encoder is $\Theta_f = \{d_v^{(0)} | v \in \mathcal{V}\}$, and is trained by two self-supervised losses, including the **Graph Contrastive Loss** and the pairwise **Bayesian Personalized Ranking (BPR) Loss**.

**Graph Contrastive Loss.** This loss is proposed to learn robust node representations through maximizing the agreement between different views of the same node compared to that of other nodes. Specifically, we generate two views $\mathcal{G}_p$ and $\mathcal{G}_g$ using the predefined augmentation strategies and the view generator, respectively. Encoding $\mathcal{G}_p$ and $\mathcal{G}_g$, we have two set of node representations:

$$\begin{aligned} \{d_v^p | v \in \mathcal{V}\} &= f(\mathcal{G}_p), \\ \{d_v^g | v \in \mathcal{V}\} &= f(\mathcal{G}_g). \end{aligned} \tag{13}$$

Then the graph contrastive loss is defined as :

$$\mathcal{L}_{gcl} = -\sum_{v \in \mathcal{V}} \log \frac{\exp(\frac{d_v^{p\top} d_v^g}{\tau_f})}{\sum_{u \in \mathcal{V}} \exp(\frac{d_u^{p\top} d_v^g}{\tau_f})}, \tag{14}$$

where $\tau_f$ is the temperature hyper-parameter in softmax.

**Algorithm 1:** GACN Framework.

**Input:** graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, dimension of embedding $s$, hyper-parameters $\tau_g, \tau_f, \lambda_g, \lambda_{cnt}, \lambda_{new}, \lambda_{gcl}, \lambda_{bpr}$
**Output:** node representations $\{d_v | v \in \mathcal{V}\}$

1 Initialize $\Theta_g, \Theta_h, \Theta_f$;
2 **while** *GACN not converge* **do**
3  **for** *G-Steps* **do**
4  Sample $X_g$ and calculate $P$ according to Eq. (4) ;
5  Compute $p_{\mathcal{G}_g}$ using $P$ according to Eq. (9), (10) and (11);
6  Update $\Theta_g$ according to Eq. (7) and (17);
7  **end**
8  **for** *D-Steps* **do**
9  Generate and label $\vec{\mathcal{G}}_p, \vec{\mathcal{G}}_g$ ;
10  Encode $G \in \vec{\mathcal{G}}_p \cup \vec{\mathcal{G}}_g$ using Eq. (9), (10) and (11);
11  Update $\Theta_h$ according to Eq. (12);
12  **end**
13  **for** *E-Steps* **do**
14  Generate $\mathcal{G}_p, \mathcal{G}_g$ for Eq. (14) and sample $O$ for Eq. (15) ;
15  Update $\Theta_f$ according to Eq. (16);
16  **end**
17 **end**
18 $\{d_v | v \in \mathcal{V}\} = f(\mathcal{G})$ ;
19 **return** $\{d_v | v \in \mathcal{V}\}$.

**Bayesian Personalized Ranking Loss.** This loss is introduced to learn representations that are suitable for downstream tasks, especially for link prediction, and the intuition is to maximize the similarity of connected nodes, while minimize the similarity of disconnected nodes. Formally, the bpr loss is defined as:

$$\mathcal{L}_{bpr} = -\frac{1}{|O|} \sum_{(i,j,k) \in O} \log \sigma(d_i^\top d_j - d_i^\top d_k), \tag{15}$$

where $O = \{(i, j, k) | A_{i,j} = 1, A_{i,k} = 0\}$ is the training data.

Then, the self-supervised loss is the combination of the above two losses:

$$\mathcal{L}_{ssl} = \lambda_{gcl} \cdot \mathcal{L}_{gcl} + \lambda_{bpr} \cdot \mathcal{L}_{bpr}, \tag{16}$$

where $\lambda_{gcl}$ and $\lambda_{bpr}$ are hyper-parameters to balance the influences of $\mathcal{L}_{gcl}$ and $\mathcal{L}_{bpr}$, respectively.

## 4.5 Model Optimization

In this subsection, we present the parameter optimization procedure of GACN. As shown in Algorithm 1, the view generator, the view discriminator and the graph encoder are optimized sequentially and iteratively.

**G-Steps (Lines 3–7)** (see Figure 3b) optimize the parameters of the view generator. Specifically, in each iteration, an augmented view $\mathcal{G}_g$ is generated and then the regularization loss is computed. In consideration of generating high-quality views, an adversarial classification loss is incorporated to cheat the view discriminator by labeling $\mathcal{G}_g$ with 1. According to Eq. (12), we have:

$$\mathcal{L}'_{clf} = -\log(p_{\mathcal{G}_g}). \tag{17}$$

**D-Steps (Lines 8–12)** (see Figure 3c) optimize the parameters of the view discriminator by generating $\vec{\mathcal{G}}_p, \vec{\mathcal{G}}_g$ and training the discriminator to classify them.

**E-Steps (Lines 13–17)** (see Figure 3d) first prepare the training data for the self-supervised losses and then update the parameters of the graph encoder.

**Table 2: The statistics of datasets.**

| Datasets | $\lvert\mathcal{V}\rvert$ | $\lvert\mathcal{E}\rvert$ | Task |
|---|---|---|---|
| Cora | 2,708 | 5,429 | Node Classification |
| Citeseer | 3,312 | 4,714 | Node Classification |
| UCI | 1,677 | 56,617 | Link Prediction |
| Taobao | 12,611 | 20,890 | Link Prediction |
| Amazon | 10,099 | 148,659 | Link Prediction |
| Last.fm | 127,786 | 720,537 | Link Prediction |
| Kuaishou | 138,812 | 1,779,639 | Link Prediction |

Note that all the parameters are optimized using the back propagation algorithm. After converging, we obtain the learned node representations $\{d_v|v \in \mathcal{V}\}$ by encoding graph $\mathcal{G}$ (Lines 18–19).

**Time Complexity Analysis.** For the view generator, the time complexity to generate a single view and calculate the regularization loss is $O(|\mathcal{V}|^2)$. For the view discriminator, the time complexity to encode a graph using LightGCN [17] is $O(|\mathcal{V}|^2 \cdot N_l D)$, where $N_l$ is the number of GCN layers. The time complexity to pool the graph and calculate $\mathcal{L}_{clf}$ is $O(|\mathcal{V}| \cdot D)$. For the graph encoder, the time complexity to encode the two augmented views is $O(|\mathcal{V}|^2 \cdot N_l D)$ and the time complexity to compute $\mathcal{L}_{ssl}$ is $O(|\mathcal{V}|^2 \cdot D) + |\mathcal{O}| \cdot D)$. Thus, the overall time complexity of Algorithm 1 is $O(N_{iter}[N_G + N_D \cdot (|\vec{\mathcal{G}}_p| + |\vec{\mathcal{G}}_g|) + N_E] \cdot |\mathcal{V}|^2 \cdot N_l D)$ , where $N_{iter}$ is the number of iteration round, $N_G$, $N_D$ and $N_E$ are the number of G-Steps, D-Steps and E-Steps, respectively.

## 5 EXPERIMENTS

In this section, we conduct extensive experiments and answer the following research questions:

- **RQ1**: How does GACN perform w.r.t. node classification task?
- **RQ2**: How does GACN perform w.r.t. link prediction task?
- **RQ3**: What are the benefits of the proposed modules of GACN?
- **RQ4**: Can the generator of GACN generate high-quality graphs for contrastive learning?
- **RQ5**: How do different settings influence the effectiveness of GACN?

### 5.1 Experimental Settings

*5.1.1 Datasets.* We evaluate the performance of GACN on seven real-world datasets, including two datasets for node classification namely Cora and Citeseer, and five datasets for link prediction namely UCI, Taobao, Amazon, Lastfm and Kuaishou. We summarize the statistics of all the datasets in Table 2. The detailed information of these datasets is listed as follows.

**Datasets for Node Classification.**

- **Cora** [29] consists of 2,708 scientific publications classified into one of seven classes. The citation network consists of 5,429 edges. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1,433 unique words.
- **Citeseer** [10] is similar to Cora. It consists of 3,312 nodes and 4,714 edges. The nodes are classified into one of six classes and the dimension of node feature is 3,703.

**Datasets for Link Prediction.**

**Table 3: The experimental results of node classification. The best results are illustrated in bold and the number underlined is the runner-up.**

| Dataset | Cora | | | Citeseer | | |
|---|---|---|---|---|---|---|
| Metric | P | R | F1 | P | R | F1 |
| DeepWalk | 0.7753 | 0.7012 | 0.7292 | 0.5579 | 0.4962 | 0.4869 |
| LINE | 0.7873 | 0.6970 | 0.7281 | 0.5992 | 0.4437 | 0.4413 |
| node2vec | 0.7744 | 0.7352 | 0.7516 | 0.4717 | 0.4581 | 0.4552 |
| LightGCN | 0.7615 | 0.7342 | 0.7453 | 0.4434 | 0.4513 | 0.4361 |
| Simple-GCL | 0.8491 | 0.8154 | 0.8287 | 0.7019 | 0.6930 | 0.6946 |
| DGI | 0.8320 | 0.8129 | 0.8212 | 0.6427 | 0.6357 | 0.6278 |
| GraphCL | 0.7993 | 0.7500 | 0.7689 | 0.6547 | 0.6156 | 0.6112 |
| GRACE | 0.8546 | 0.8377 | 0.8445 | 0.7232 | 0.6963 | 0.6948 |
| SGL | 0.8029 | 0.7769 | 0.7887 | 0.7177 | 0.7045 | 0.7063 |
| GraphGAN | 0.4195 | 0.2177 | 0.1745 | 0.3148 | 0.2994 | 0.2696 |
| AD-GCL | 0.4176 | 0.3672 | 0.3800 | 0.2842 | 0.2809 | 0.2763 |
| GraphMAE | 0.8667 | 0.8287 | 0.8447 | 0.7278 | 0.7048 | 0.7064 |
| GACN | **0.8705** | **0.8545** | **0.8614** | **0.7311** | **0.7187** | **0.7212** |

- **UCI** [1] contains the message communications between the students of the University of California, Irvine in an online community.
- **Taobao** [51] is offered by Alibaba with user behaviors collected from Taobao[1]. There are 1,000 users with all the corresponding interactive items in this dataset.
- **Amazon** [16] is a network that includes product metadata and links between products. We use the data provided by [5] which contains the product metadata of *Electronic* category.
- **Last.fm**[2] contains <user, artist, song> tuples collected from Last.fm API, which represents the whole listening habits of nearly 1,000 users. We use the <user, artist> pairs to construct a network.
- **Kuaishou**[3] is collected from the Kuaishou online video-watching platform. This dataset includes the interactions of 6,840 users and 131,972 videos.

*5.1.2 Baseline Methods.* To demonstrate the effectiveness and efficiency of GACN, we choose twelve state-of-the-art baseline methods, categorized into three groups. Graph representation learning models include DeepWalk [32], LINE [35], node2vec [12] and LightGCN [17]. Graph contrastive learning models include DGI [39], GraphCL [47], GRACE [52] and SGL [43]. Graph generative and adversarial learning models include GraphGAN [40] , AD-GCL [34] and GraphMAE [20]. The details of the baseline methods are listed as follows.

**Graph Representation Learning Models.**

- **DeepWalk** [32] is an embedding method for static homogeneous networks. It exploits the random walk strategy and the skip-gram model to learn node vector representations.
- **LINE** [35] learns node representations by modelling the first- and second-order proximity between node pairs.
- **node2vec** [12] adds two parameters to control the random walk process based on DeepWalk.

[1] https://www.taobao.com/
[2] https://www.last.fm/
[3] https://www.kuaishou.com/

**Table 4: The experimental results of link prediction. The best results are illustrated in bold and the number underlined is the runner-up.**

| Dataset | UCI | | Taobao | | Amazon | | Last.fm | | Kuaishou | |
|---|---|---|---|---|---|---|---|---|---|---|
| Metric | H@50 | MRR | H@50 | MRR | H@50 | MRR | H@50 | MRR | H@50 | MRR |
| DeepWalk | <u>0.2550</u> | 0.0474 | 0.3522 | <u>0.1764</u> | <u>0.4496</u> | 0.0744 | 0.1344 | 0.0180 | 0.0486 | 0.0055 |
| LINE | 0.1086 | 0.0249 | 0.3117 | 0.1758 | 0.3327 | 0.0661 | 0.0857 | 0.0112 | 0.0223 | 0.0031 |
| node2vec | 0.1808 | 0.0312 | 0.3533 | 0.1754 | 0.3026 | 0.0619 | 0.1184 | 0.0181 | 0.0623 | 0.0073 |
| LightGCN | 0.1093 | 0.0256 | 0.3433 | 0.1692 | 0.4359 | 0.0818 | 0.1368 | <u>0.0203</u> | <u>0.0745</u> | <u>0.0093</u> |
| Simple-GCL | 0.2549 | <u>0.0574</u> | 0.3330 | 0.1355 | 0.4079 | 0.0818 | 0.0581 | 0.0100 | 0.0511 | 0.0060 |
| DGI | 0.1972 | 0.0310 | 0.1657 | 0.0388 | 0.1463 | 0.0258 | 0.0972 | 0.0151 | 0.0485 | 0.0060 |
| GraphCL | 0.1669 | 0.0291 | 0.1659 | 0.0348 | 0.1692 | 0.0334 | 0.1012 | 0.0145 | 0.0468 | 0.0061 |
| GRACE | 0.1915 | 0.0270 | 0.2006 | 0.1056 | 0.3127 | 0.0553 | <u>0.1385</u> | 0.0198 | 0.0439 | 0.0055 |
| SGL | 0.2545 | <u>0.0574</u> | <u>0.3654</u> | 0.1741 | 0.4014 | 0.0811 | 0.0981 | 0.0150 | 0.0702 | 0.0086 |
| GraphGAN | 0.2543 | 0.0374 | 0.3538 | 0.1390 | 0.4380 | <u>0.0882</u> | 0.0781 | 0.0115 | 0.0544 | 0.0067 |
| AD-GCL | 0.1819 | 0.0323 | 0.1008 | 0.0214 | 0.0843 | 0.0118 | 0.0822 | 0.0111 | 0.0184 | 0.0024 |
| GraphMAE | 0.0170 | 0.0052 | 0.1366 | 0.0441 | 0.2660 | 0.0348 | 0.0307 | 0.0043 | 0.0206 | 0.0031 |
| GACN | **0.2836** | **0.0692** | **0.3794** | **0.1895** | **0.5593** | **0.1158** | **0.1568** | **0.0263** | **0.1067** | **0.0132** |

- **LightGCN** [17] is a light-weight graph convolution network, which is easy to train and has good generalization ability.

**Graph Contrastive Learning Models.**

- **Simple-GCL** is a variant of SGL-ED [43] implemented by us and is trained using the InfoNCE [30] loss upon two views generated via edge-dropping.
- **DGI** [39] relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs to learn node representations in an unsupervised manner.
- **GraphCL** [47] is a graph contrastive learning framework for learning unsupervised representations of graph data, which designs four types of graph augmentations to incorporate various priors.
- **GRACE** [52] generates two graph views by corruption and learn node representations by maximizing the agreement of node representations in these two views.
- **SGL** [43] explores self-supervised learning on graph structure and accordingly devises three unified augmentation operators including node dropout, edge dropout and random walk.

**Graph Generative and Adversarial Learning Models.**

- **GraphGAN** [40] is a graph representation learning framework unifying the generative model and the discriminative model, in which these two models play a game-theoretical minimax game.
- **AD-GCL** [34] proposes a principle to avoid capturing redundant information during the training by optimizing adversarial graph augmentation strategies used in GCL.
- **GraphMAE** [20] explores generative self-supervised learning in graphs and designs a state-of-the-art graph autoencoder using the masked feature reconstruction strategy with a scaled cosine error as the reconstruction criterion.

Note that we focus on node-level tasks in this paper, and methods designed for graph-level tasks such as GCA [53], JOAO [46], MVGRL [15] and GASSL [44] are not chosen as baselines.

*5.1.3 Parameter Settings.* We implement GACN with Pytorch and the model is optimized using the Adam optimizer with learning rate 0.001 during the training phase. By default, $\tau_g$ is set to 0.0001, $\tau_f$ is set to 0.5, $\lambda_g$ is set to 0.5, $\lambda_{cnt}$ is set to 1, $\lambda_{new}$ is set to 0.5, $\gamma$ is set to 0.75. For Cora, Citeseer and UCI, $\lambda_{gcl}$ is set to 1 and $\lambda_{bpr}$ is set to 0.0001, while for the other datasets, $\lambda_{gcl}$ is set to 0.0001 and $\lambda_{bpr}$ is set to 1. For all the baseline methods, we tune the parameters according to the validation set and report the best results. The dimension of embedding $s$ of all the model is set to 128, and all the experiments are conducted on a single GTX 1080Ti GPU.

*5.1.4 Metrics.* For the node classification task, we choose three widely-used metrics, namely *P(recision)*, *R(ecall)* and *F1*. For the link prediction task, we adopt two ranking metrics, include *MRR* and *H(it rate)@k*. In this paper, we report *H@50* and similar results are observed when $k = 20$ and $k = 100$.

## 5.2 Node Classification (RQ1)

We evaluate the performance of GACN w.r.t. the node classification task on the Cora and the Citeseer datasets. Following the experimental setting as Hassani and Khasahmadi [15] and Velickovic et al. [39], we first run different methods without supervision to generate all the nodes' embeddings. Then we train a linear classifier and report the mean accuracy on the test nodes through 10 random initialization. As shown in Table 3, GACN achieves the best results compared to the SOTA baseline methods in all benchmarks. Notably, GACN outperforms existing GCL methods by a large margin on two node classification datasets. Notice that AD-GCL is designed for graph-level tasks and has poor performances on node-level tasks.

## 5.3 Link Prediction (RQ2)

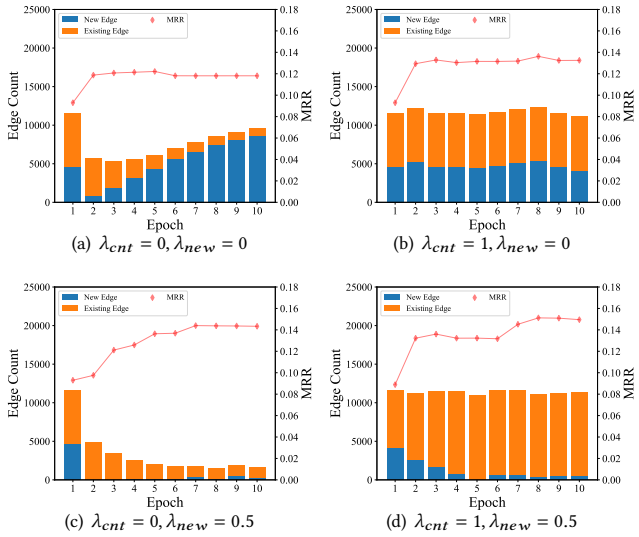
In the subsection, we conduct the link prediction on the UCI, Taobao, Amazon, Last.fm and the Kuaishou datasets. Table 4 reports the experimental results and it is observed that: 1) GACN consistently performs the best on all datasets compared to other methods. We attribute these results to the fact that GACN is able to explore unseen edges and generate high-quality views for graph contrastive learning. 2) Although SGL leverages the BPR loss and the contrastive loss for self-supervise learning, GACN still outperforms SGL, showing the effectiveness of incorporating GCL with graph GANs.

## 5.4 Ablation Study (RQ3)

In this subsection, we evaluate the effectiveness of different components of GACN, with five variants as follows:

**Table 5: Ablation Study. The best results are illustrated in bold and the number underlined is the runner-up.**

| Dataset | Cora | | | Citeseer | | | UCI | | Taobao | | Amazon | | Last.fm | | Kuaishou | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metric | P | R | F1 | P | R | F1 | H@50 | MRR | H@50 | MRR | H@50 | MRR | H@50 | MRR | H@50 | MRR |
| *w/o* REG | 0.8650 | 0.8385 | 0.8498 | 0.7250 | 0.7136 | 0.7155 | 0.0685 | 0.0127 | 0.3159 | 0.1308 | 0.4850 | 0.0997 | 0.1486 | 0.0230 | 0.0840 | 0.0103 |
| *w/o* GAN | 0.8670 | 0.8497 | 0.8571 | 0.7286 | 0.7170 | 0.7191 | 0.2389 | 0.0627 | 0.3654 | 0.1741 | 0.4025 | 0.0818 | 0.0982 | 0.0150 | 0.0693 | 0.0086 |
| *w/o* SSL | 0.8608 | 0.8368 | 0.8470 | 0.7265 | 0.7151 | 0.7168 | 0.0871 | 0.0160 | 0.1743 | 0.0553 | 0.0380 | 0.0074 | 0.0023 | 0.0004 | 0.0007 | 0.0001 |
| *w/o* GCL | 0.7226 | 0.3442 | 0.3565 | 0.7305 | 0.7175 | 0.7194 | 0.0817 | 0.0171 | 0.3775 | 0.1850 | 0.5064 | 0.1036 | 0.1515 | 0.0247 | 0.1001 | 0.0122 |
| *w/o* BPR | 0.8632 | 0.8392 | 0.8494 | 0.7307 | 0.7181 | 0.7207 | 0.2727 | 0.0672 | 0.2326 | 0.1270 | 0.3587 | 0.0718 | 0.0497 | 0.0081 | 0.0554 | 0.0066 |
| GACN | **0.8705** | **0.8545** | **0.8614** | **0.7311** | **0.7187** | **0.7212** | **0.2836** | **0.0692** | **0.3794** | **0.1895** | **0.5593** | **0.1158** | **0.1568** | **0.0263** | **0.1067** | **0.0132** |




(a) $\lambda_{cnt} = 0, \lambda_{new} = 0$ (b) $\lambda_{cnt} = 1, \lambda_{new} = 0$

(c) $\lambda_{cnt} = 0, \lambda_{new} = 0.5$ (d) $\lambda_{cnt} = 1, \lambda_{new} = 0.5$

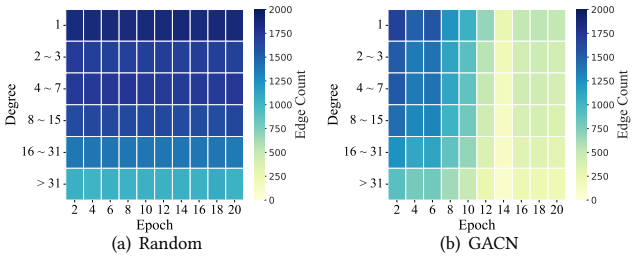
**Figure 4: The impact of $\lambda_{cnt}$ and $\lambda_{new}$.**

- ***w/o* REG**: The view generator is trained without the regularization loss $\mathcal{L}_{reg}$.
- ***w/o* GAN**: The contrastive loss $\mathcal{L}_{gcl}$ is optimized using views generated by predefined augmentation strategies only.
- ***w/o* SSL**: The self-supervised learning losses are ignored and the model is optimized using the G-Steps and D-Steps only.
- ***w/o* GCL**: The graph contrastive loss $\lambda_{gcl}$ is ignored during E-Steps.
- ***w/o* BPR**: The bayesian personalized ranking loss $\lambda_{bpr}$ is ignored during E-Steps.

Table 5 shows the experimental results. We can find that: 1) The regularization loss plays as an assistant role in performance, which indicates that $\mathcal{L}_{reg}$ helps to generate rational views for contrastive learning. 2) The graph GAN is important in GACN, showing the benefits of utilizing GAN to generate views and the joint learning framework. 3) The self-supervised learning losses are essential to GACN, because the GAN in GACN is based on graph-level classification and does not focus on learning node representations.

## 5.5 Quality of Generated Graphs (RQ4)

In this subsection, we investigate the quality of graphs generated by the view generator w.r.t. *Impact of $\lambda_{cnt}$ and $\lambda_{new}$*, *New Edge Distribution* and *Case Study*.




(a) Random (b) GACN

**Figure 5: The experimental results of the distribution of new edges.**

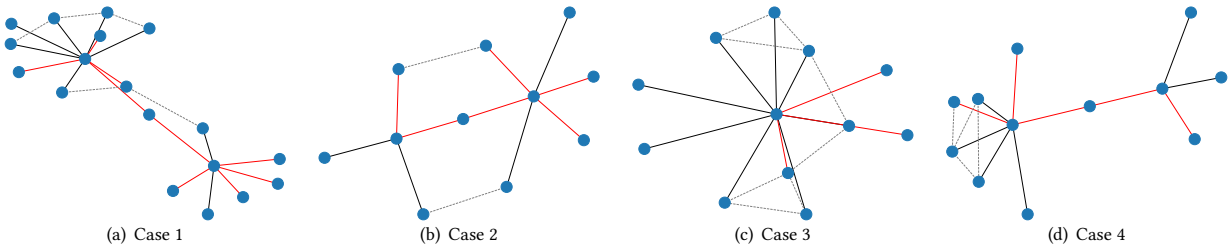
*5.5.1 Impact of $\lambda_{cnt}$ and $\lambda_{new}$.* In this part, we run the link prediction task on UCI under different settings of $\lambda_{cnt}$ and $\lambda_{new}$, while in each training epoch, we generate ten views and calculate the average amount of edges and the new edges. As shown in Figure 4, $\lambda_{cnt}$ contributes to the stability of the edge count, while $\lambda_{new}$ helps to limit the number of new edges. Specifically, when $\lambda_{cnt} = 1$ and $\lambda_{new} = 0.5$, the number of new edges decreases gradually as the training goes on while the edge count remains stable, which obtains the highest MRR compared to other settings.

*5.5.2 New Edge Distribution.* To further investigate the distribution of new edges, we count the number of new edges in groups divided by node degree. Figure 5 illustrates that compared to randomly adding new edges, GACN 1) is able to adjust the number of new edges during training, and 2) generates more edges for high degree nodes (i.e., the color of high degree nodes is more similar to low degree nodes in Figure 5(b) compared to that in Figure 5(a)), which is in agreement with the preferential attachment rule [2].

*5.5.3 Case Study.* For better insight of the views generated by GACN, we randomly sample some nodes in UCI and visualize their neighborhoods within two hops. Figure 6 shows that GACN tends to attach nodes to those with high degree and removes other edges, which confirms that GACN indeed learns the preferential attachment rule [2] and is able to generate reasonable alternative views for contrastive learning.

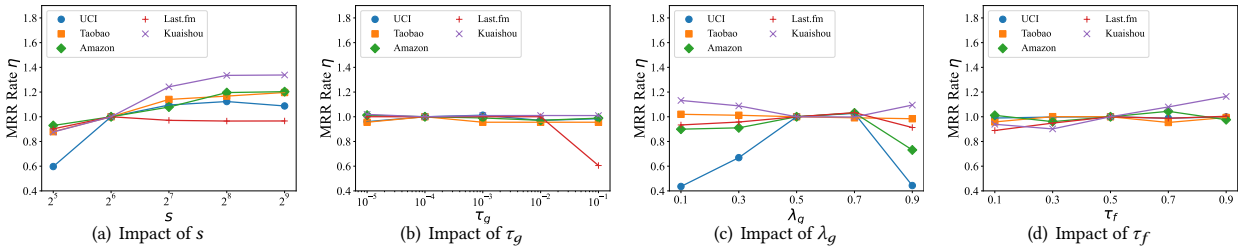
## 5.6 Parameter Sensitivity (RQ5)

In this subsection, we analyze the sensitivity of hyper-parameters in GACN. Specifically, we first examine the impact of the dimension of embedding $s$, the hyper-parameter of the view generator $\tau_g$ and $\lambda_g$, and the temperature of the contrastive learning $\tau_f$. We report the



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

**Figure 6: Each case is the neighborhood of a sampled node in the generated view. Edges in red are generated by GACN. Edges in black are those existing in the original graph and preserved by GACN. Edges in gray and dashed are those existing in the original graph but dropped by GACN. It is observed that GACN learns the preferential attachment rule and tends to attach nodes to those with high degree.**




(a) Impact of $s$ (b) Impact of $\tau_g$ (c) Impact of $\lambda_g$ (d) Impact of $\tau_f$

(e) Impact of $\lambda_{cnt}$ (f) Impact of $\lambda_{new}$ (g) Impact of $\gamma$

**Figure 7: The experimental results of parameter sensitivity.**

MRR rate $\eta = \frac{\text{MRR with current settings}}{\text{MRR with default settings}}$ w.r.t. the link prediction task in Figure 7.

As shown in Figure 7(a), the larger dimension of embedding yields the better performance due to the strengthened expression capability of the GACN model. Figure 7(b) shows that GACN is insensitive to $\tau_g$. However, a large $\tau_g$ may result in poor performance. As shown in Figure 7(c), GACN is sensitive to $\lambda_g$. Specifically, a small $\lambda_g$ results in sparse views, which are uninformative for contrastive learning, while a large $\lambda_g$ yields dense views, which do harm to robust node representation learning. Generally, setting $\lambda_g$ to $[0.5, 0.7]$ is a good choice. From Figure 7(d), it is observed that different datasets require different $\tau_f$ for best performance. In general, setting $\tau_f$ to 0.5 yields competitive performances.

We also analyze how the view generator influences GCL, i.e., the sensitivity of $\lambda_{cnt}, \lambda_{new}$ and $\gamma$. It is observed that GACN setting $\lambda_{cnt}$ to 1 can obtain competitive results (see Figure 7(e)), while a small $\lambda_{new}$ is preferred (see Figure 7(f)). However, setting $\lambda_{new}$ to 0 generates a large amount of unseen edges, and can result in poor performance in some datasets. Thus setting $\lambda_{new}$ to 0.25 is a better choice. In contrast, different datasets are sensitive to $\gamma$ (see Figure 7(g)). However, setting $\gamma$ to 0.75 produces satisfying results.

## 6 CONCLUSION

In this paper, we incorporated graph GANs with GCL w.r.t. node-level tasks, and presented GACN, a new GNN model that leveraged a graph GAN to generate augmented views for GCL. Specifically, GACN developed a view generator, a view discriminator and a graph encoder to learn node representations in a self-supervised learning style. Besides, a novel optimization framework was proposed to train the modules of GACN jointly. Through comprehensive experiments on seven real-world datasets, we empirically showed the superiority of GACN. In the future, GACN will be developed to deal with heterogeneous and dynamic graphs.

## ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (No. 61872207) and Kuaishou Inc. Chaokun Wang is the corresponding author.

# REFERENCES


[1] 2016. UC Irvine messages network dataset – KONECT. http://konect.uni-koblenz.de/networks/opsahl-ucsocial

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[3] S. Becker and G. E. Hinton. 1992. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature* 355, 6356 (1992), 161.

[4] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*. PMLR, 610–619.

[5] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *KDD*. ACM, 1358–1368.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.

[7] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2018. Adversarial network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[8] Emily L Denton, Soumith Chintala, Rob Fergus, et al. 2015. Deep generative image models using a laplacian pyramid of adversarial networks. *Advances in neural information processing systems* 28 (2015).

[9] Hongchang Gao, Jian Pei, and Heng Huang. 2019. Progan: Network embedding via proximity generative adversarial network. In *Proceedings of the 25th ACM SIGKDD International Conf. on Knowledge Discovery & Data Mining*. 1308–1316.

[10] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. 1998. CiteSeer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*. 89–98.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. ACM.

[13] Tiankai Gu, Chaokun Wang, Cheng Wu, Yunkai Lou, Jingcao Xu, Changping Wang, Kai Xu, Can Ye, and Yang Song. 2022. HybridGNN: Learning Hybrid Representation for Recommendation in Multiplex Heterogeneous Networks. In *ICDE*. IEEE, 1355–1367.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[15] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *ICML*. PMLR, 4116–4126.

[16] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *The World Wide Web Conference*. 507–517.

[17] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.

[18] Olivier Henaff. 2020. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*. PMLR, 4182–4192.

[19] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2019. Learning deep representations by mutual information estimation and maximization. In *ICLR*.

[20] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. 2022. GraphMAE: Self-Supervised Masked Graph Autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, NY, USA, 594–604. https://doi.org/10.1145/3534678.3539321

[21] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. 2019. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[22] Nikola Jovanović, Zhao Meng, Lukas Faber, and Roger Wattenhofer. 2021. Towards robust graph contrastive learning. *arXiv preprint arXiv:2102.13085* (2021).

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[24] Kwot Sin Lee, Ngoc-Trung Tran, and Ngai-Man Cheung. 2021. Infomax-gan: Improved adversarial image generation via information maximization and contrastive learning. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 3942–3952.

[25] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. 2019. GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 388–396.

[26] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547* (2017).

[27] Ralph Linsker. 1988. Self-organization in a perceptual network. *Computer* 21, 3 (1988), 105–117.

[28] Weiyi Liu, Pin-Yu Chen, Fucai Yu, Toyotaro Suzumura, and Guangmin Hu. 2019. Learning graph topological features via GAN. *IEEE Access* 7 (2019), 21834–21843.

[29] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.

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

[31] Tian Pan, Yibing Song, Tianyu Yang, Wenhao Jiang, and Wei Liu. 2021. Videomoco: Contrastive video representation learning with temporally adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11205–11214.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. 701–710.

[33] Yiwei Sun, Suhang Wang, Tsung-Yu Hsieh, Xianfeng Tang, and Vasant Honavar. 2019. Megan: A generative adversarial network for multi-view network embedding. *arXiv preprint arXiv:1909.01084* (2019).

[34] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. 2021. Adversarial graph augmentation to improve graph contrastive learning. *Advances in Neural Information Processing Systems* 34 (2021).

[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding.. In *WWW*. ACM.

[36] Sahar Tavakoli, Alireza Hajibagheri, and Gita Sukthankar. 2017. Learning social graph topologies using generative adversarial neural networks. In *International Conference on Social Computing, Behavioral-Cultural Modeling & Prediction*.

[37] Yonglong Tian, Dilip Krishnan, and Phillip Isola. 2020. Contrastive multiview coding. In *European conference on computer vision*. Springer, 776–794.

[38] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv e-prints* (2018), arXiv–1807.

[39] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. *ICLR (Poster)* 2, 3 (2019), 4.

[40] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[41] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 515–524.

[42] Cheng Wu, Chaokun Wang, Jingcao Xu, Ziwei Fang, Tiankai Gu, Changping Wang, Yang Song, Kai Zheng, Xiaowei Wang, and Guorui Zhou. 2023. Instant Representation Learning for Recommendation over Large Dynamic Graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 81–94.

[43] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 726–735.

[44] Longqi Yang, Liangliang Zhang, and Wenjing Yang. 2021. Graph adversarial self-supervised learning. *Advances in Neural Information Processing Systems* 34 (2021), 14887–14899.

[45] Min Yang, Junhao Liu, Lei Chen, Zhou Zhao, Xiaojun Chen, and Ying Shen. 2019. An advanced deep generative framework for temporal link prediction in dynamic networks. *IEEE transactions on cybernetics* 50, 12 (2019), 4946–4957.

[46] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. In *International Conference on Machine Learning*. PMLR, 12121–12132.

[47] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.

[48] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

[49] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. 2018. Learning deep network representations with adversarially regularized autoencoders. In *Proceedings of the 24th ACM SIGKDD international conf. on knowledge discovery & data mining*. 2663–2671.

[50] Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2017. Aspect-augmented adversarial networks for domain adaptation. *Transactions of the Association for Computational Linguistics* 5 (2017), 515–528.

[51] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *KDD*. ACM, 1079–1088.

[52] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).

[53] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.

[54] Marinka Zitnik, Rok Sosič, and Jure Leskovec. 2018. Prioritizing network communities. *Nature communications* 9, 1 (2018), 1–9.