Knowledge Distillation Improves Graph Structure Augmentation for Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Graph (structure) augmentation aims to perturb the graph structure through heuristic or probabilistic rules, enabling the nodes to capture richer contextual information and thus improving generalization performance. While there have been a few graph structure augmentation methods proposed recently, none of them are aware of a potential *negative augmentation* problem, which may be caused by overly severe distribution shifts between the original and augmented graphs. In this paper, we take an important graph property, namely graph homophily, to analyze the distribution shifts between the two graphs and thus measure the severity of an augmentation algorithm suffering from negative augmentation. To tackle this problem, we propose a novel *Knowledge Distillation for Graph Augmentation* (KDGA) framework, which helps to reduce the potential negative effects of distribution shifts, i.e., negative augmentation problem. Specifically, KDGA extracts the knowledge of any GNN teacher model trained on the augmented graphs and injects it into a partially parameter-shared student model that is tested on the original graph. As a simple but efficient framework, KDGA is applicable to a variety of existing graph augmentation methods and can significantly improve the performance of various GNN architectures. For three popular graph augmentation methods, namely GAUG, MH-Aug, and GraphAug, the experimental results show that the learned student models outperform their vanilla implementations by an average accuracy of 4.6% (GAUG), 4.2% (MH-Aug), and 4.6% (GraphAug) on eight graph datasets.

1 Introduction

In many real-world applications, including social networks, chemical molecules, and citation networks, data can be naturally modeled as graphs. Recently, the emerging Graph Neural Networks (GNNs) [5, 13, 22, 45, 23, 25, 47, 59] have demonstrated their powerful capability due to their superior performance in various graph-related tasks, including link prediction [55], node classification [22], and graph classification [7]. Despite their great success, GNNs usually suffer from weak generalization due to its heavy reliance on the quantity of annotated labels and the quality of the graph structure. To boost generalization capabilities, a natural solution is to increase the amount of training data by creating plausible variations of existing data, which have been widely adopted in fields such as computer vision [31, 28, 9, 37, 26, 29, 4, 15] and natural language processing [44, 1, 34, 6, 32].

The data augmentation on graphs can be mainly divided into two branches: node feature augmentation and graph structure augmentation. While the former has been well studied by directly extending existing approaches for image and text data to graph data [51, 20, 16], comparatively little work has been done to study graph structure augmentation [33, 2, 58, 30]. Following the nomenclature of existing works [58, 30], we directly abbreviate *graph (structure) augmentation* to *graph augmentation* for the sake of brevity in this paper. The purpose of graph augmentation is to reasonably perturb the graph structure through heuristic or probabilistic rules, enabling the nodes to capture richer contextual information and thus improving generalization performance. For example, *DropEdge* [33] randomly removes a fraction of edges before each training epoch, in an approach reminiscent of dropout [38]. Besides, *AdaEdge* [2] iteratively adds (removes) edges between nodes predicted to have the same (different) labels with high confidence. In contrast to these heuristic methods, *GAUG* [58] proposes to optimize the graph augmentation and GNN parameters in an end-to-end manner. Similarly, *MH-Aug* [30] proposes a novel framework that draws a sequence of augmented graphs from an explicit target distribution, which enables flexible control of the strength and diversity of augmentation.

In this paper, we identify a potential *negative augmentation* problem for existing graph augmentation methods, i.e., the augmentation may cause overly severe *distribution shift* between the augmented graphs used for training and the original graph used for testing, which leads to suboptimal generalization. Moreover, we conduct extensive experiments to demonstrate the existence and hazard of distribution shifts and find that the direction of distribution shifts may be opposed on homophily and heterophily graphs. We propose a solution to the identified problem by adopting a *Knowledge Distillation for Graph Augmentation* (KDGA) framework, which helps to reduce the potential negative effects of distribution shifts. Specifically, it extracts the knowledge of any GNN teacher model trained on the augmented graphs and injects it into a partially parameter-shared student model that is tested on the original graph. As a general framework, KDGA can significantly improve the vanilla implementations of various popular graph augmentation methods and GNN architectures.

Our contributions are summarized as follows: (1) We are the first to identify a potential negative augmentation problem for graph augmentation, and more importantly, we have described in detail what it represents, how it arises, what impact it has, and how to deal with it. (2) We proposes a novel *Knowledge Distillation for Graph Augmentation* (KDGA) framework for the identified problem by directly distilling contextual information from augmented graphs. (3) We provide comprehensive experimental results showing that KDGA is applicable to a variety of graph augmentation methods and GNN models; it substantially outperforms the vanilla implementations across various datasets.

2 Background and Related Work

Structure Augmentation for Graphs. Data augmentation is an effective technique to improve generalization. Despite the great progress on node feature augmentation [51, 20, 16], comparatively little work study graph (structure) augmentation [33, 2, 58, 30] due to the non-Euclidean property of structures. For graph data, the mainstream algorithms for structure augmentation are divided into two categories: heuristic and learning-based. As a typical heuristic algorithm, *DropEdge* [33] randomly remove edges according to the hand-crafted probability. In a similar way, *AdaEdge* [2] iteratively adds (removes) edges between nodes predicted to have the same (different) labels. Different from the above heuristic methods, *GAUG* [58] propose to optimize the graph augmentation and learnable GNN parameters in an end-to-end manner. Instead, *MH-Aug* [30] proposes a sampling-based augmentation, where a sequence of augmented graphs are directly drawn from an explicit target distribution.

Graph Structure Learning and Graph Contrastive Learning. Two closely related topics to graph augmentation are Graph Structure Learning [19, 24, 53, 21, 3] and Graph Contrastive Learning [20, 16, 52, 27, 60, 54], but *they are quite different in terms of learning objectives and evaluation protocols*. The learning goal of structure learning is to estimate a new structure with high quality [10, 8]. Instead, graph augmentation aims to reasonably perturb the graph structure during training to produce a set of augmented graphs, enabling nodes to receive richer contextual information; such augmentations allow the model to generalize better across those variations. As for the evaluation protocol, the augmented graphs are only used during training and are not available during testing. In contrast, for graph structure learning, the learned structure is used during both training and testing.

There are also some recent works [39, 62, 50] exploring how to perform data augmentation for graph contrastive learning, but they focus on automatically selecting the most appropriate transformations from a given pool to improve contrastive learning, rather than learning customized augmentation strategies for GNNs. More importantly, graph contrastive learning aims to learn transferable knowledge from abundant unlabeled data in an *unsupervised* setting and then generalize the learned knowledge to downstream tasks. Instead, graph augmentation usually works in a *semi-supervised* setting, i.e., the label information is available during training. The graph structure learning and contrastive learning are not newly born topics, and we refer readers to the recent surveys [46, 61].

3 Preliminaries

Notions. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of $N = |\mathcal{V}|$ nodes with features $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N] \in \mathbb{R}^{N\times d}$ and $\mathcal{E}$ denotes the edge set. Each node $v_i \in \mathcal{V}$ is associated with a d-dimensional features vector $\mathbf{x}_i$, and each edge $e_{i,j} \in \mathcal{E}$ denotes a connection between node v_i and v_j. The graph structure can also be denoted by an adjacency matrix $\mathbf{A} \in [0,1]^{N\times N}$ with $\mathbf{A}_{i,j} = 1$ if $e_{i,j} \in \mathcal{E}$ and $\mathbf{A}_{i,j} = 0$ if $e_{i,j} \notin \mathcal{E}$. Consider a semi-supervised node classification task where only a subset of node $\mathcal{V}_L$ with corresponding labels $\mathcal{Y}_L$ are known, we denote the labeled set as $\mathcal{D}_L = (\mathcal{V}_L, \mathcal{Y}_L)$ and unlabeled set as $\mathcal{D}_U = (\mathcal{V}_U, \mathcal{Y}_U)$, where $\mathcal{V}_U = \mathcal{V} \backslash \mathcal{V}_L$. The node classification task aims to learn a mapping $\Phi : \mathcal{V} \rightarrow \mathcal{Y}$ on labeled data $\mathcal{D}_L$, so that it can be used to infer labels $\mathcal{Y}_U$.

Background on Graph Homophily Ratio. The homophily ratio is an important graph property that reflects the extent to which the graph structure adheres to the "label smoothness" criterion. The graph homophily ratio r can be defined as the fraction of intra-class edges in the graph, as follows

$$r = \frac{|\{(i,j) : (i,j) \in \mathcal{E} \wedge y_i = y_j\}|}{|\mathcal{E}|} \tag{1}$$

where y_i and y_j are the ground-truth labels of node v_i and v_j. In practice, the distribution space size of a discrete graph structure $\mathbf{A} \in [0,1]^{N\times N}$ is 2^{N^2}, making it tractable to directly estimate the distribution differences between two discrete graph structures. In this paper, we take the graph homophily as a desirable option to analyze the distribution shifts between the original and augmented graphs, thus measuring the severity of an algorithm suffering from the *negative augmentation* problem.

4 Methodology

In this section, we first make problem statements for graph augmentation in Sec. 4.1, highlight our motivations by analyzing the distribution shift between the original and augmented graphs in Sec. 4.2, then present a novel teacher-student *<u>K</u>nowledge <u>D</u>istillation for <u>G</u>raph <u>A</u>ugmentation* (KDGA) framework in Sec. 4.3, and finally provide one of its specific instantiations in Sec. 4.4.

4.1 Problem Statement

Graph Representation Learning. From the perspective of statistical learning, the key of node classification is to learn a mapping $p(Y \mid \mathbf{X}, \mathbf{A})$ based on node features $\mathbf{X}$ and graph structure $\mathbf{A}$. The learned mapping can be used to infer labels $\mathcal{Y}_U$ on the graph structure $\mathbf{A}$ as shown in Fig. 1(a).

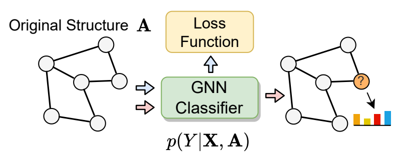
Graph Structure Learning. The goal of graph structure learning is to estimate a more accurate structure $\widehat{\mathbf{A}}$ by another mapping $p(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$ and then feed it into the mapping $p(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ along with node features $\mathbf{X}$. Finally, the learned mapping $p(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ can be used to infer labels $\mathcal{Y}_U$ on the estimated (high-quality) structure $\widehat{\mathbf{A}}$ instead of the original strucute $\mathbf{A}$ as shown in Fig. 1(b).

Graph Augmentation. Instead of directly working with the original graph, we would like to leverage graph augmentation to reasonably perturb the graph structure and learn more generalizable representations. In other words, we are interested in the following variant, as follows

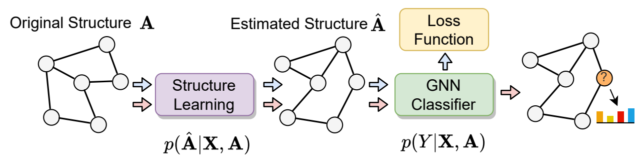
$$p(Y \mid \mathbf{X}, \mathbf{A}) = \sum_{\widehat{\mathbf{A}} \in [0,1]^{N\times N}} p(Y \mid \mathbf{X}, \widehat{\mathbf{A}}) p(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A}) \tag{2}$$

where $\widehat{\mathbf{A}} \in [0,1]^{N\times N}$ is the augmented graph (structure). In practice, the distribution space size of $\widehat{\mathbf{A}}$ is 2^{N^2}, and it is intractable to enumerate all possible $\widehat{\mathbf{A}}$ as well as estimate the exact values of the mappings $p(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ and $p(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$. Therefore, we approximate them by tractable functions as

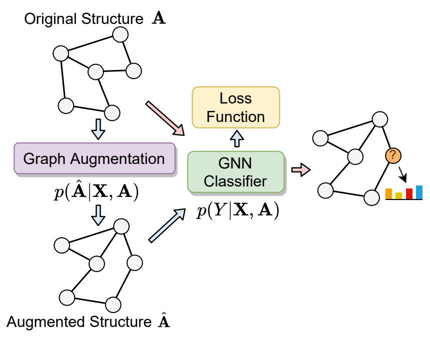
$$p(Y \mid \mathbf{X}, \mathbf{A}) = \sum_{\widehat{\mathbf{A}} \in [0,1]^{N\times N}} q_\theta(Y \mid \mathbf{X}, \widehat{\mathbf{A}}) q_\phi(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A}) \tag{3}$$



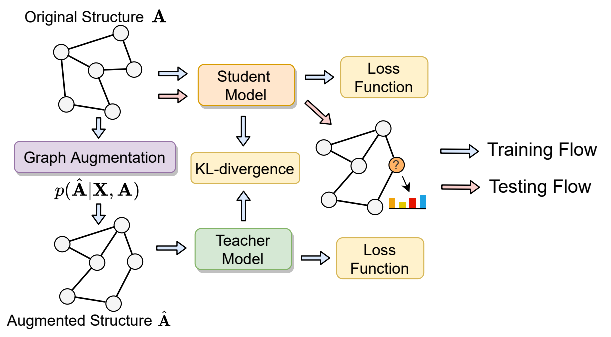
(a) Graph Representation Learning


Original Structure $\mathbf{A}$
Estimated Structure $\hat{\mathbf{A}}$
Structure Learning
$p(\hat{\mathbf{A}}|\mathbf{X},\mathbf{A})$
Loss Function
GNN Classifier
$p(Y|\mathbf{X},\mathbf{A})$

(b) Graph Structure Learning


Original Structure $\mathbf{A}$
Graph Augmentation
$p(\hat{\mathbf{A}}|\mathbf{X},\mathbf{A})$
Augmented Structure $\hat{\mathbf{A}}$
Loss Function
GNN Classifier
$p(Y|\mathbf{X},\mathbf{A})$

(c) Graph Augmentation


Original Structure $\mathbf{A}$
Graph Augmentation
$p(\hat{\mathbf{A}}|\mathbf{X},\mathbf{A})$
Augmented Structure $\hat{\mathbf{A}}$
Student Model
Loss Function
KL-divergence
Teacher Model
Loss Function
Training Flow
Testing Flow

(d) Knowledge Distillation for Graph Augmentation (KDGA)

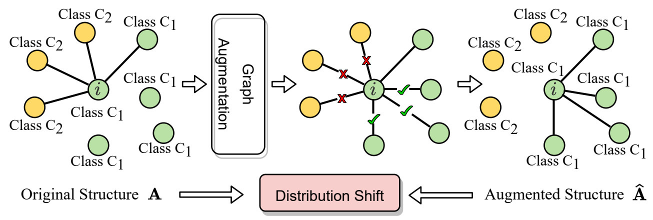
Figure 1: Illustrations of graph representation learning, graph structure learning, graph augmentation, and the proposed KDGA framework. For the sake of chart brevity, we omitted the node features $\mathbf{X}$.

where $q_\theta(\cdot)$ and $q_\phi(\cdot)$ are approximation functions for $p(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ and $p(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$ parameterized by θ and ϕ, respectively. In practice, the function $q_\theta(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ can be generally implemented by GNNs, and the function $q_\phi(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$ can be implemented by graph augmentation methods to model the distributions of augmented graph structures. Once the model training is finished, the mapping $q_\theta(Y \mid \mathbf{X}, \mathbf{A})$ can be used to infer labels $\mathcal{Y}_U$ on the original structure $\mathbf{A}$ as shown in Fig. 1(c).

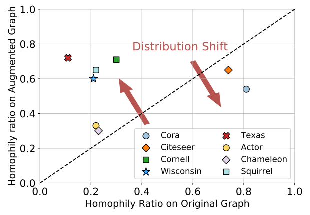
In summary, unlike graph representation and graph structure learning that leverage the same structure ($\mathbf{A}$ or $\widehat{\mathbf{A}}$) for both training and testing, the graph structures for training and testing in graph augmentation are completely different, which may lead to a potential negative augmentation problem.

4.2 Motivation: Potential Negative Augmentation Problem

One may create a model by specifying specific implementations for functions $q_\theta(Y \mid \mathbf{X}, \widehat{\mathbf{A}})$ and $q_\phi(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$ and then optimize it by maximizing the posterior $p(Y \mid \mathbf{X}, \mathbf{A})$ defined in Eq. (3). As we will explain here, however, this model may suffer from a potential negative augmentation problem caused by overly severe distribution shifts between the original and augmented graphs.



(a) Distribution shift on the real-world Wisconsin dataset



(b) Statistics of homophily ratios

Figure 2: Illustrations of how the distribution shift is arising and how it behaves on different datasets.

The distribution shift itself is not necessarily harmful; it is actually a neutral phenomenon. A proper distribution shift helps the model "see" more different graphs, enabling the nodes to receive more contextual information, thus improving generalization; however, an overly severe distribution shift can lead to a potential negative augmentation problem. To illustrate it, we consider a node v_i (id 129) of class C_1 from the real-world Wisconsin dataset in Fig. 2(a), it is initially connected to a node with the same class C_1 and three nodes from another class C_2 in the original structure $\mathbf{A}$. During the training process, the original structure $\mathbf{A}$ and node features $\mathbf{X}$ are fed together into $q_\phi(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$ to generate an augmented structure $\widehat{\mathbf{A}}$. Under the downstream supervision, it disconnects from three nodes from class C_2 and reconnects with three nodes from the same class C_1, resulting in an augmented structure $\widehat{\mathbf{A}}$ with a much higher homophily ratio, that is, an overly severe distribution shift between the original and augmented graphs from the perspective of graph homophily property. As a result, a model trained on the augmented structure $\widehat{\mathbf{A}}$ can successfully predict node i as class C_1, but make a wrong prediction C_2 for node i when tested on the original structure $\mathbf{A}$, which is termed as *"negative augmentation"*. Furthermore, we plot the homophily ratios of the original structure $\mathbf{A}$ and augmented structure $\widehat{\mathbf{A}}$ on eight datasets in Fig. 2(b), from which we can observe significant distribution shifts between the two graphs. Moreover, while the above analysis is developed on a heterophily (Wisconsin) graph, we find that the identified distributional shift also exists in homophily graphs, only in a different direction. Please see Sec. 5.3 for detailed experimental settings and results.

4.3 Knowledge Distillation for Graph Augmentation (KDGA)

The distribution shift is essentially a trade-off between better generalizability and higher risks of negative augmentation. However, the *optimal* distribution shift may vary from dataset to dataset, or even from node to node, making it challenging to directly control the levels of distribution shifts. In this paper, we have not attempted to control or prevent distribution shifts. Instead, we allow for the existence of any level of distribution shifts, but we reduce their negative impact, i.e., the potential negative augmentation problem, by the proposed KDGA framework, which gradually distills the contextual information from the augmented graphs into a student model tested on the original graph. The idea of KDGA is straightforward, yet as we will see, extremely effective. In our case, we first generate soft distributions $\mathbf{z}_i^T$ and $\mathbf{z}_i^S$ for node v_i with the teacher and student models, respectively. The knowledge distillation is first introduced in [14], where knowledge was transferred from a cumbersome teacher to a simpler student by optimizing the following objective function, as follows

$$\mathcal{L}_{\mathrm{KD}} = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \mathcal{D}_{KL}\left(\operatorname{softmax}\left(\mathbf{z}_i^T\right), \operatorname{softmax}\left(\mathbf{z}_i^S\right)\right) \tag{4}$$

In this paper, not to get a simpler student model, we adopt the knowledge distillation framework to address the identified negative augmentation problem caused by overly severe distribution shifts between the two graphs. In short, we extract the knowledge of any teacher model trained on the augmented graphs and inject it into a student model tested on the original graph as in Fig. 1(d).

Teacher Model. The teacher model can be implemented by any GNN, which takes node features $\mathbf{X}$ and augmented structure $\widehat{\mathbf{A}}$ as input and learn latent node representations via neighborhood feature aggregation. Considering a L-layer GNN $f_\theta(\mathbf{X}, \widehat{\mathbf{A}})$, the formulation of the l-th layer is as follows

$$\mathbf{h}_{i,T}^{(l+1)} = \mathrm{UPDATE}^{(l)}\left(\mathbf{h}_{i,T}^{(l)}, \mathrm{AGGREGATE}^{(l)}\left(\left\{\mathbf{h}_{j,T}^{(l)} : v_j \in \mathcal{N}_i^{\widehat{\mathbf{A}}}\right\}\right)\right) \tag{5}$$

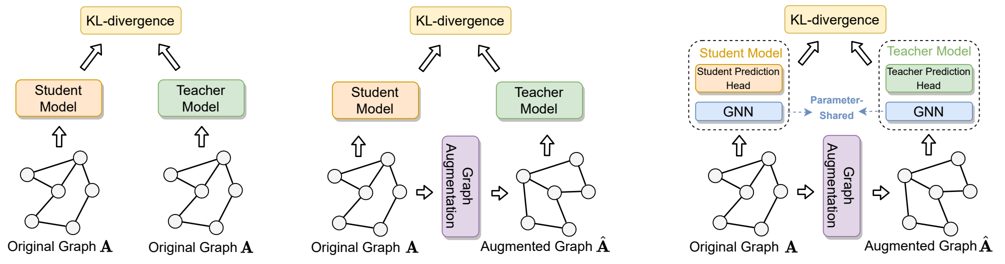
where $0 \leq l \leq L-1$, $\mathbf{h}_{i,T}^{(0)} = \mathbf{x}_i$ is the input feature, and $\mathcal{N}_i^{\widehat{\mathbf{A}}}$ is the neighborhood of node v_i in the augmented structure $\widehat{\mathbf{A}}$. After L message-passing layers, the final node embedding $\mathbf{h}_{i,T}^{(L)}$ is passed to a linear prediction head $g^T(\cdot)$ to obtain logits $\mathbf{z}_i^T = g^T(\mathbf{h}_{i,T}^{(L)})$, and the model is trained by a cross-entropy loss $\mathcal{H}(\cdot)$ with ground-truth labels $\mathcal{Y}_L$, given by $\mathcal{L}_{SUP}^T = \sum_{i \in \mathcal{V}_L} \mathcal{H}\left(y_i; \operatorname{softmax}\left(\mathbf{z}_i^T\right)\right)$.

Student Model. The student model $f_\theta(\mathbf{X}, \mathbf{A})$ shares the parameters θ with the teacher model, but differs in that the it takes the original structure $\mathbf{A}$ as input, as shown in Fig. 3(c). Besides, an additional linear prediction head $g^S(\cdot)$ is used to map the node embedding $\mathbf{h}_{i,S}^{(L)}$ to logits $\mathbf{z}_i^S = g^S(\mathbf{h}_{i,S}^{(L)})$. As already explained earlier, the augmented graphs enable the teacher model to receive richer contextual information, which helps to improve model generalization. To allow the student model tested on the original structure to also benefit from it, we consider the contextual neighborhood information from both original and augmented structures and distill them into the student model, defined as

$$\mathcal{L}_{\mathrm{GKD}} = \frac{\tau_1^2}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \sum_{j \in (\mathcal{N}_i^{\mathbf{A}} \cap \mathcal{N}_i^{\widehat{\mathbf{A}}}) \cup i} \mathcal{D}_{KL}\left(\operatorname{softmax}\left(\mathbf{z}_j^T / \tau_1\right), \operatorname{softmax}\left(\mathbf{z}_i^S / \tau_1\right)\right) \tag{6}$$

where τ_1 is the distillation temperature, and τ_1^2 is used to keep the gradient stability of this loss [14].

Discussions. While a large number of methods on graph knowledge distillation [57, 48, 56] have been proposed, most of them adopt the standard teacher-student knowledge distillation framework as shown in Fig. 3(a), where the inputs to both teacher and student models are the same (structure). Despite many progresses, their contributions have mostly focused on the special design of the teacher or student models. For example, CPF [49] proposes to distill knowledge from a teacher GNN to a student MLP, but it specifically incorporates label propagation [17] into the student model to improve performance. In contrast, GDK [11] utilizes label propagation in the teacher model to fully exploit both feature and topological information. In our proposed KDGA framework, the graph structures fed to the teacher and student models are completely different. Moreover, unlike the scheme in Fig. 3(b) where two parameter-independent teacher and student models are used, we adopt the architecture shown in Fig. 3(c) where the GNN parameters are shared but with two independent prediction heads to increase discriminability. The behind motivation is that a parameter-independent student model has the risk of quickly fitting with the original structure under the optimization of downstream supervision, while failing to take full advantage of rich contextual information from the augmented graphs. A detailed comparison of parameter-independent and parameter-shared schemes is reported in Table. 2.



(a) Standard Scheme (b) Parameter-Independent (Param-I) (c) Parameter-Shared (Param-S)

Figure 3: (a) standard teacher-student distillation; (b) distillation with parameter-independent teacher and student models; (c) distillation with parameter-shared teacher and student models.

4.4 Instantiating KDGA with GraphAug

In practice, any existing graph augmentation method can be used to instantiate the proposed KDGA framework and achieve consistent improvements over the vanilla implementations, as shown in Table. 1. In this subsection, we adopt a probabilistic generative-based graph augmentation method to model the function $q_\phi(\widehat{\mathbf{A}} \mid \mathbf{X}, \mathbf{A})$, termed GraphAug, and use it to instantiate our KDGA framework. Specifically, we introduce a set of discrete variables $\Lambda = \{\lambda_{i,j}\}_{i,j=1}^N$ to model the distribution of the augmented graph, where $\lambda_{i,j} \in \{0,1\}$ denotes the augmentation probability between node v_i and v_j. Moreover, we avoid estimating the probability $p\left(\lambda_{i,j} \mid \mu_{i,j}\right)$ using independent local parameter $\mu_{i,j}$ and instead fits a shared neural network to estimate it. Specifically, we first transform the input to a low-dimensional hidden space, done by multiplying the node features with a parameter matrix $\mathbf{W} \in \mathbb{R}^{F \times d}$, that is, $\mathbf{e}_i = \mathbf{W} x_i$. Then, we directly parameterize the probability $\lambda_{i,j}$ as

$$p\left(\lambda_{i,j} \mid \mathbf{X}, \mathbf{A}\right) = \sigma\left(\mathbf{e}_i \mathbf{e}_j^T\right) \tag{7}$$

where $\sigma(\cdot)$ is an element-wise sigmoid function. Next, to sample discrete augmented graphs from the learned augmentation distribution and make the sampling process differentiable, we adopt Gumbel-Softmax sampling [18]. Specifically, the sampling process can be formulated as

$$\widehat{\mathbf{A}}_{i,j} = \left\lfloor \frac{1}{1+\exp^{-\left(\log \mathbf{M}_{i,j}+G\right)/\tau_2}} + \frac{1}{2} \right\rfloor, \text{where} \quad \mathbf{M}_{i,j} = \alpha p\left(\lambda_{i,j} \mid \mathbf{X}, \mathbf{A}\right) + (1-\alpha)\mathbf{A}_{i,j} \tag{8}$$

where $\alpha \in [0,1]$ is the fusion factor to control the intensity of the graph augmentation, τ_2 is the augmentation temperature, and $G \sim \text{Gumbel}(0,1)$ is a gumbel random variate.

To warm-up the proposed GraphAug module, we first pre-train it with loss $\mathcal{L}_{Aug} = \frac{1}{N^2}\mathcal{H}(\mathbf{A}_{i,j}, \widehat{\mathbf{A}}_{i,j})$, where $\mathcal{H}(\cdot)$ denotes the cross-entropy loss. Besides, we use classification loss $\mathcal{L}_{Cla} =$

Table 1: Accuracy ± std (%) on eight datasets (as well as their homophily ratios), with three GNN architectures and five graph augmentation methods considered. The best metrics are marked by **bold**.

BaseGNN	Method	Cora	Citeseer	Cornell	Chameleon	Squirrel	Actor	Wisconsin	Texas
		0.81	**0.74**	**0.30**	**0.23**	**0.22**	**0.22**	**0.21**	**0.11**
GCN	Vanilla	81.5±0.8	71.6±0.3	57.0±4.7	59.8±2.6	36.9±1.3	30.3±0.8	59.8±7.0	59.5±5.3
	DropEdge [33]	82.2±0.7	71.9±0.3	59.3±3.9	61.2±1.8	38.1±1.5	30.9±1.0	61.8±5.4	62.3±4.6
	AdaEdge [2]	82.3±0.8	69.7±0.9	57.8±4.3	59.5±2.3	37.6±1.4	31.4±1.2	60.4±4.7	58.8±4.0
	SSL [60]	83.8±0.7	72.9±0.6	58.8±3.2	60.4±2.1	39.5±1.9	30.5±1.2	62.8±4.5	63.3±4.6
	GraphMix [42]	83.9±0.6	**74.7±0.6**	60.5±3.7	61.2±2.3	41.1±1.5	31.4±0.9	62.4±5.0	62.3±4.6
	GAUG [58]	83.6±0.5	73.3±1.1	55.8±4.0	59.3±1.4	36.3±0.8	29.7±0.9	57.5±5.1	58.0±4.2
	GAUG (w/ KDGA)	**85.4±0.7**	73.6±0.6	63.2±3.6	63.0±1.2	46.2±0.9	33.3±0.8	65.0±2.5	67.4±3.8
	Δ_{Acc}	1.8	0.3	7.4	3.7	9.9	3.6	7.5	9.4
	MH-Aug [30]	83.6±0.3	73.0±0.5	58.4±3.5	59.2±2.0	35.9±1.0	31.2±0.7	58.1±5.3	58.9±3.9
	MH-Aug (w/ KDGA)	85.0±0.5	73.8±0.8	63.5±2.7	**63.3±1.7**	45.4±1.1	**34.8±1.0**	65.7±2.7	67.2±2.6
	Δ_{Acc}	1.4	0.8	5.1	4.1	9.5	3.6	7.6	8.3
	GraphAug	83.2±0.9	73.2±0.8	56.6±2.4	58.8±1.8	37.2±1.2	28.8±0.9	59.3±2.6	59.4±3.3
	GraphAug (w/ KDGA)	85.2±0.7	73.9±0.7	**63.8±3.2**	62.7±1.5	**46.9±0.6**	32.5±0.6	**66.3±1.9**	**68.0±2.3**
	Δ_{Acc}	2.0	0.7	7.2	3.9	9.7	3.7	6.9	8.6
SAGE	Vanilla	79.8±0.7	71.1±0.6	76.0±5.0	58.7±1.7	41.6±0.7	34.2±1.0	81.2±5.6	82.4±6.1
	DropEdge [33]	80.4±0.8	71.5±0.6	77.4±3.6	60.2±2.0	42.5±1.3	36.4±1.3	82.7±4.4	83.0±4.8
	AdaEdge [2]	80.2±1.2	69.4±0.8	76.5±4.6	59.5±1.6	40.3±1.6	34.9±0.8	82.0±5.3	81.6±5.3
	SSL [60]	82.5±0.8	71.2±0.5	76.8±3.4	59.1±1.8	42.0±1.5	35.2±1.2	82.4±3.6	82.6±4.4
	GraphMix [42]	82.3±0.6	69.6±0.4	78.0±4.2	59.9±2.0	42.6±1.6	35.8±1.0	83.1±4.1	83.5±3.9
	GAUG [58]	82.0±0.5	72.7±0.7	74.8±4.2	58.2±1.3	40.5±0.9	34.4±1.1	80.7±4.6	82.0±4.5
	GAUG (w/ KDGA)	84.5±0.8	73.4±0.7	80.6±3.5	61.8±1.6	**46.4±1.1**	36.4±0.7	**85.5±3.2**	84.5±3.6
	Δ_{Acc}	2.5	0.7	5.8	3.6	5.9	2.0	4.8	2.5
	MH-Aug [30]	82.6±0.7	72.1±1.0	75.3±3.9	59.4±1.5	41.0±0.8	33.8±0.8	80.5±5.0	81.2±5.2
	MH-Aug (w/ KDGA)	84.3±0.7	**73.7±0.8**	80.3±3.2	**62.1±1.3**	45.9±1.4	35.9±0.7	84.9±4.0	83.8±4.4
	Δ_{Acc}	1.7	1.6	5.0	2.7	4.9	2.1	4.4	2.6
	GraphAug	82.4±1.0	72.4±0.9	75.8±3.0	58.8±1.4	40.2±1.3	33.2±0.7	79.9±4.2	81.9±4.6
	GraphAug (w/ KDGA)	**84.8±0.8**	73.5±0.5	**81.4±2.8**	61.0±1.8	45.6±0.9	**36.9±1.4**	84.5±3.3	**84.8±3.8**
	Δ_{Acc}	2.4	1.1	5.6	2.2	5.4	3.7	4.6	2.9
GAT	Vanilla	82.2±0.5	71.4±0.9	58.9±3.3	54.7±2.0	30.6±2.1	26.3±1.7	55.3±8.7	58.4±4.5
	DropEdge [33]	83.0±0.4	72.2±0.9	60.2±3.8	55.6±2.5	34.1±1.7	28.2±1.5	57.8±5.5	60.5±3.8
	DropEdge [33]	77.9±2.0	69.1±0.8	57.7±4.5	54.0±2.2	32.8±2.0	27.5±1.4	56.4±6.1	57.8±4.2
	SSL [60]	83.7±0.6	72.7±0.7	60.6±3.2	55.8±2.2	35.0±1.3	27.6±1.3	57.2±5.1	60.5±3.3
	GraphMix [42]	83.3±0.2	73.1±0.2	61.0±4.1	56.4±1.7	35.6±1.0	28.7±0.9	58.5±4.5	61.1±2.8
	GAUG [58]	82.2±0.8	71.6±1.1	57.6±3.8	53.4±1.4	30.1±1.5	25.8±1.0	54.8±5.7	56.9±3.6
	GAUG (w/ KDGA)	84.2±1.1	73.0±0.7	62.2±3.4	58.2±1.1	**39.1±1.3**	**31.3±1.2**	60.9±5.3	63.1±3.2
	Δ_{Acc}	2.0	1.4	4.6	4.8	9.0	5.5	6.1	6.2
	MH-Aug [30]	83.5±0.7	72.8±1.0	58.0±4.0	55.3±1.8	29.5±1.1	25.7±1.2	55.8±4.0	57.8±4.0
	MH-Aug (w/ KDGA)	84.5±0.9	**73.4±0.8**	62.7±2.8	**59.5±1.6**	37.3±0.8	30.8±0.9	61.4±5.0	**64.4±2.8**
	Δ_{Acc}	1.0	0.6	4.7	4.2	7.8	5.1	5.6	6.6
	GraphAug	83.2±0.8	72.5±0.7	58.6±3.4	54.0±1.7	29.8±1.6	24.8±1.3	54.4±3.6	57.1±4.4
	GraphAug (w/ KDGA)	**84.7±0.7**	73.2±0.8	**63.1±2.5**	58.8±1.3	38.9±1.4	30.0±1.0	**61.8±4.7**	62.7±2.0
	Δ_{Acc}	1.5	0.7	4.5	4.8	9.1	5.2	7.4	5.6

$\frac{1}{|\mathcal{V}_L|}\sum_{i\in\mathcal{V}_L}\left(\mathcal{H}\left(y_i,\mathrm{softmax}(\mathbf{z}_i^T)\right)+\mathcal{H}\left(y_i,\mathrm{softmax}(\mathbf{z}_i^S)\right)\right)$ to pre-train the teacher and student models until it converges. Finally, the total loss to train the whole framework is defined as follows

$$\mathcal{L}_{total}=\mathcal{L}_{Cla}+\lambda\mathcal{L}_{Aug}+\kappa\mathcal{L}_{GKD} \tag{9}$$

where λ and κ are the weights to balance the influence of the two losses $\mathcal{L}_{Aug}$ and $\mathcal{L}_{GKD}$.

5 Experiments

Datasets. The effectiveness of the proposed KDGA framework is evaluated on *eight* datasets. We use two commonly used homophily graph datasets, namely *Cora* [36] and *Citeseer* [12] as well as six heterophily graph datasets: *Cornell, Texas, Wisconsin*, *Aactor* [40], *Chameleon* and *Squirrel* [35]. A statistical overview of these datasets is available in **Appendix A**. We defer the implementation details and the best hyperparameter settings for each dataset to **Appendix B** and supplementary material.

Baselines. As a general framework, KDGA can be combined with any GNN architecture and existing graph augmentation methods. In this paper, we consider three GNN architectures, GCN [22], GraphSAGE [13], and GAT [41]. Besides, to demonstrate the applicability of KDGA to various graph augmentation methods in addition to the proposed GraphAug, we also consider two state-of-the-art learning-based baselines, GAUG [58] and MH-Aug [30]. In particular, two heuristics methods, DropEdge and AdaEdge, are also included in the comparison as baselines. Moreover, we also compare KDGA with two semi-supervised methods: (1) GraphMix [42], a regularization method that performs linear interpolation between two data on graphs, and (2) SSL [60], that proposes two self-supervised tasks to fully exploit available information embedded in the graph structure. Each set of experiments is run five times with different random seeds, and the average performance is reported.

5.1 Comparative Results

To evaluate the powerful capabilities of the proposed KDGA framework, we instantiate it with three learning-based graph augmentation methods, GAUG, MH-Aug, and GraphAug. The experiments are conducted on eight datasets with three different GNN architectures. From the experimental results shown in Table. 1, we can make the following observations: (1) Two heuristic graph augmentation methods, DropEdge and AdaEdge, can improve the performance of the vanilla GNNs overall. However, such improvements are usually very limited and do not work for all datasets and GNN architectures. For example, on the Citeseer dataset, the performance of AdaEdge drops over the vanilla GNNs by 1.9% (GCN), 1.7% (GraphSAGE), and 2.3% (GAT), respectively. (2) There are huge gaps in the effectiveness of three learning-based augmentation methods on homophily and heterophily graphs. While these methods can significantly improve performance on homophily graphs, their performance gains on heterophily graphs are greatly reduced and even detrimental. For example, with GCN as the GNN architecture, the performance of GAUG improves by 2.1% on Cora, but drops by 1.5% and 1.2% on Texas and Cornell. Such negative augmentation is mainly caused by the overly severe distribution shift between the original and augmented graphs as analyzed in Sec. 4.2. (3) The proposed KDGA framework can consistently improve the performance of vanilla graph augmentation methods across three GNN architectures on all eight datasets, especially for those heterophily graphs. For example, with GCN as the GNN architecture, the performance of GraphAug can be improved by 9.7% and 8.6% on the Squirrel and Texas datasets. (4) Two semi-supervised approaches, SSL and GraphMix, can achieve comparable or even better performance than learning-based graph augmentation, especially on heterophily graphs. However, by combining with KDGA, GAUG, MH-Aug, and GraphAug outperform both SSL and GraphMix by a large margin overall.

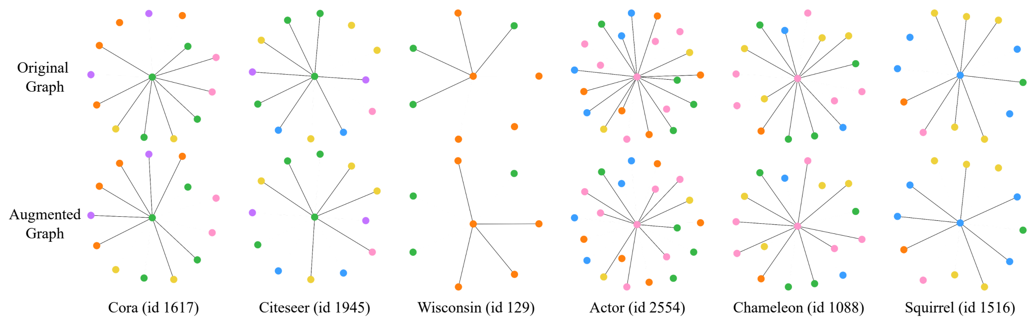


Figure 4: Case studies for each dataset, where we pick a node with the most drastic neighborhood variations and visualize its neighborhood on the original graph structure (top) and augmented graph structure (bottom), where each node is colored according to its ground-truth label.

5.2 Analysis on the Distribution Shift and Negative Augmentation

Next, we qualitatively and quantitatively analyze the distribution shift between the original and augmented graphs and explain how it can cause a potential negative augmentation. Without loss of generality, we consider GCN as the GNN architecture and GraphAug as the augmentation method.

Visualizations of Neighborhood Variations . First, we pick a node with the most drastic neighborhood variations and visualize its neighborhood of the original and augmented graphs in Fig. 4, where each node is colored according to its ground-truth label. The visualizations show that there would be a huge gap between the neighborhoods of the original and augmented graphs, which causes a model that is well trained on the augmented graph to predict poorly on the original graph during testing. Taking the Wisconsin dataset as an example, the selected node is connected to four nodes from the same class in the augmented graph, so it can be well trained to make correct predictions. However, its neighborhood context is completely changed in the original graph, where the node is connected to three nodes from different classes, so it will be predicted with high confidence to an incorrect class.

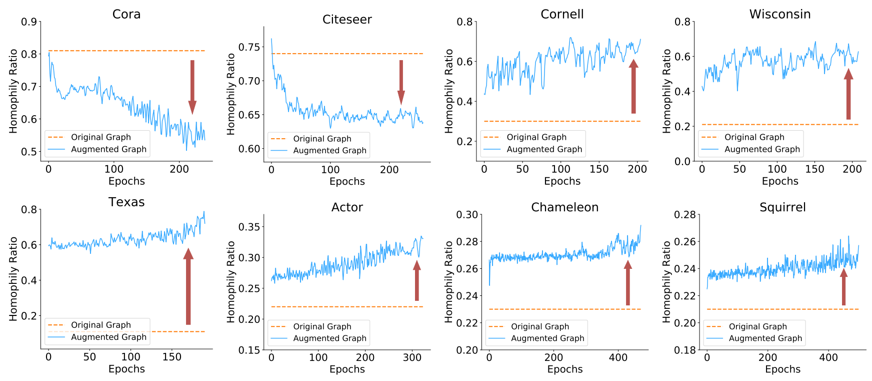


Figure 5: Training curves (w/o GKD Loss) of homophily ratios in the original and augmented graphs.

Training Curves of Homopgily Ratios. We plot in Fig. 5 the training curves (w/o GKD Loss) of the homophily ratios of the original and augmented graphs during training. It can be seen that their gaps are enlarged as training proceeds, which indicates that the distribution of the augmented graphs is gradually shifting from the original graph. This shift may even reach 0.5 for some datasets (e.g., Texas), in which case the graph homophily property is completely reversed. More importantly, we find that the direction of distribution shifts may be completely opposite for homophily and heterophily graphs, which makes it more challenging to solve the negative augmentation problem. Moreover, due to space limitations, we have placed the training curves (trained with GKD Loss) in **Appendix C**.

5.3 Ablation Study and Parameter Sensitivity

Ablation on Student Model Designs. The parameter-shared GNN shown in Fig. 3(c) is adopted as the student model by default in this paper for a fair comparison. In this subsection, we delve into the applicability of the proposed KDGA framework to different student model designs. Specifically, with the vanilla GCN as the base architecture and GraphAug as the graph augmentation method, we compare the performance of the parameter-shared model (w/ Param-S) in Fig. 3(c) and the parameter-independent model (w/ Param-I) in Fig. 3(b) on five datasets. It can be seen from Table. 2 that although the Param-I model can also improve the performance of GraphAug overall, it may fail on a few datasets, such as a 0.5% accuracy drop on Citeseer; more importantly, its performance gain falls far behind the Param-S model on all five datasets. The reason behind this may be that a parameter-independent model may be quickly fitted with the original graph structure while failing to take full advantage of the rich contextual information embedded in the augmented graphs. Moreover, we also consider a variant of the Param-I model by directly taking a parameter-independent MLP (w/ MLP) as the student mode. We find from Table. 2 that even with a simple MLP, it can still benefit from the augmented graphs and achieves performance beyond that of its vanilla implementations.

Table 2: Ablation study on student model designs.

Method	**Cora**	**Citeseer**	**Chameleon**	**Squirrel**	**Actor**
Vanilla GCN	$81.5_{\pm 0.8}$	$71.6_{\pm 0.3}$	$59.8_{\pm 2.6}$	$36.9_{\pm 1.3}$	$30.3_{\pm 0.8}$
GraphAug	$83.2_{\pm 0.9}$	$73.2_{\pm 0.8}$	$58.8_{\pm 1.8}$	$37.2_{\pm 1.2}$	$28.8_{\pm 0.9}$
KDGA w/ Param-S	$85.2_{\pm 0.7}$	$73.9_{\pm 0.7}\}$	$62.7_{\pm 1.5}$	$46.9_{\pm 0.6}\}$	$32.5_{\pm 0.6}$
Δ_{Acc}	2.0	0.7	3.9	9.7	3.7
KDGA w/ Param-I	$84.0_{\pm 0.6}$	$72.7_{\pm 0.5}\}$	$60.6_{\pm 1.7}$	$40.5_{\pm 1.0}\}$	$30.7_{\pm 0.9}$
Δ_{Acc}	0.8	-0.5	1.8	3.3	1.9
Vanilla MLP	$55.2_{\pm 0.5}$	$46.5_{\pm 0.5}$	$46.4_{\pm 2.5}$	$29.7_{\pm 1.8}$	$35.8_{\pm 1.0}$
KDGA w/ MLP	$83.2_{\pm 1.1}$	$73.5_{\pm 0.7}\}$	$58.1_{\pm 1.0}$	$38.8_{\pm 0.7}\}$	$38.1_{\pm 0.8}$
Δ_{Acc}	28.0	27.0	11.7	9.1	2.3

Sensitivity Analysis on Hyperparameters. We have evaluated the parameter sensitivity w.r.t two key hyperparameters: fusion factor α and loss weight κ. However, due to space limitations, we have placed the corresponding results of sensitivity analysis in **Appendix D**.

6 Conclusion

In this paper, we identified a potential negative augmentation problem for graph augmentation, which is caused by overly severe distribution shifts between the original and augmented graphs. To address this problem, we propose a novel *Knowledge Distillation for Graph Augmentation* (KDGA) framework by directly distilling contextual information from a teacher model trained on the augmented graphs into a partially parameter-shared student model. Extensive experiments show that KDGA outperforms the vanilla implementations of existing augmentation methods and GNN architectures. Limitations still exist, such as KDGA requires an initial raw graph structure for augmentation and cannot be applied to those structure-unknown scenarios, which will be left for future work.

References

[1] Markus Bayer, Marc-André Kaufhold, and Christian Reuter. A survey on data augmentation for text classification. *arXiv preprint arXiv:2107.03158*, 2021.

[2] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 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, pages 3438–3445, 2020.

[3] Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33:19314–19326, 2020.

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[8] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *arXiv preprint arXiv:2102.05034*, 2021.

[9] Alhussein Fawzi, Horst Samulowitz, Deepak Turaga, and Pascal Frossard. Adaptive data augmentation for image classification. In *2016 IEEE international conference on image processing (ICIP)*, pages 3688–3692. Ieee, 2016.

[10] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pages 1972–1982. PMLR, 2019.

[11] Mahsa Ghorbani, Mojtaba Bahrami, Anees Kazi, Mahdieh Soleymani Baghshah, Hamid R Rabiee, and Nassir Navab. Gkd: Semi-supervised graph knowledge distillation for graph-independent inference. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 709–718. Springer, 2021.

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

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[14] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[15] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning*, pages 2731–2741. PMLR, 2019.

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

[17] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5070–5079, 2019.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[19] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11313–11320, 2019.

[20] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

[21] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 66–74, 2020.

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

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

[24] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[25] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 338–348, 2020.

[26] Xiaofeng Liu, Yang Zou, Lingsheng Kong, Zhihui Diao, Junliang Yan, Jun Wang, Site Li, Ping Jia, and Jane You. Data augmentation via latent space interpolation for image classification. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 728–733. IEEE, 2018.

[27] Franco Manessi and Alessandro Rozza. Graph-based neural network models with multiple self-supervised auxiliary tasks. *arXiv preprint arXiv:2011.07267*, 2020.

[28] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*, pages 117–122. IEEE, 2018.

[29] Sarah O'Gara and Kevin McGuinness. Comparing data augmentation strategies for deep image classification. 2019.

[30] Hyeonjin Park, Seunghun Lee, Sihyeon Kim, Jinyoung Park, Jisu Jeong, Kyung-Min Kim, Jung-Woo Ha, and Hyunwoo J Kim. Metropolis-hastings data augmentation for graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[31] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[32] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[33] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[34] Ryan Robert Rosario. *A data augmentation approach to short text classification*. University of California, Los Angeles, 2017.

[35] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *arXiv preprint arXiv:1909.13021*, 2019.

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

[37] Jia Shijie, Wang Ping, Jia Peiyi, and Hu Siping. Research on data augmentation for image classification based on convolution neural networks. In *2017 Chinese automation congress (CAC)*, pages 4165–4170. IEEE, 2017.

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

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

[40] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816, 2009.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[42] Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10024–10032, 2021.

[43] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[44] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.

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

[46] Lirong Wu, Haitao Lin, Zhangyang Gao, Cheng Tan, Stan Li, et al. Self-supervised on graphs: Contrastive, generative, or predictive. *arXiv preprint arXiv:2105.07342*, 2021.

[47] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[48] Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. Tinygnn: Learning efficient graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1848–1856, 2020.

[49] Cheng Yang, Jiawei Liu, and Chuan Shi. Extract the knowledge of graph neural networks and go beyond it: An effective knowledge distillation framework. In *Proceedings of the Web Conference 2021*, pages 1227–1237, 2021.

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

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

[52] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks? In *International Conference on Machine Learning*, pages 10871–10880. PMLR, 2020.

[53] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. Graph-revised convolutional network. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 378–393. Springer, 2020.

[54] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[55] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018.

[56] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727*, 2021.

[57] Wentao Zhang, Xupeng Miao, Yingxia Shao, Jiawei Jiang, Lei Chen, Olivier Ruas, and Bin Cui. Reliable data distillation on graph convolutional network. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1399–1414, 2020.

[58] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830*, 2020.

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

[60] Qikui Zhu, Bo Du, and Pingkun Yan. Self-supervised training of graph convolutional networks. *arXiv preprint arXiv:2006.02380*, 2020.

[61] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. Deep graph structure learning for robust representations: A survey. *arXiv preprint arXiv:2103.03036*, 2021.

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Lines 56-62 in Section 1.
   (b) Did you describe the limitations of your work? [Yes] See Lines 314-315 in Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the guidelines and ensured that our paper conforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material submitted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The training details of the data splits and hyperparameter settings have been placed in **Appendix A.1** and **Appendix A.2**, respectively.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Each set of experiments is run five times with different random seeds, and the average accuracy and standard deviation are reported.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The implementation uses the PyTorch library running on NVIDIA v100 GPU, which is detailed in **Appendix A.2**.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] The assets in this paper come from some open-source platforms, such as Github, and follow the MIT license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] It does not contain such sensitive information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Dataset Statistics

Eight publicly available graph datasets are used to evaluate the proposed KDGA framework. An overview summary of the statistical characteristics of datasets is given in Tab. A1.

Cora, Citeseet, and Pubmed. These three are citation network benchmark datasets for node classification. In these datasets, nodes represent papers, and edges denote citations of one paper by another. Node features are the bag-of-words representation of papers, and the node label is the academic topic of a paper. The data splittings of these datasets are the same as [22].

Cornell, Texas, and Wisconsin. Cornell, Texas, and Wisconsin [1] are three sub-datasets of WebKB, which is a webpage dataset collected from computer science departments of various universities by Carnegie Mellon University. In these datasets, nodes represent web pages, and edges are hyperlinks between them. Node features are the bag-of-words representation of web pages. The nodes are manually classified into five categories: student, project, course, staff, and faculty.

Actor. This dataset is a subgraph of the film-director-actor-writer network. In this dataset, nodes represent actors, and edges are their co-occurrence on the same Wikipedia page. Node features are some keywords in the Wikipedia pages. The nodes are manually classified into five categories in terms of the words of the actor's Wikipedia.

Chameleon and Squirrel. Chameleon and Squirrel are two page-page networks on specific topics in Wikipedia. In these datasets, nodes represent web pages, and edges are mutual links between pages. Node features are several informative nouns in the Wikipedia pages. The nodes are classified into five categories in terms of the number of the average monthly traffic of the web page.

Table A1: Statistical information of the datasets.

Dataset	Cora	Citeseer	Chameleon	Squirrel	Texas	Cornell	Wisconsin	Actor
# Nodes	2708	3327	2277	5210	183	183	251	7600
# Edges	5278	4614	3142	198493	279	277	450	26659
# Features	1433	3703	2325	2089	1703	1703	1703	932
# Classes	7	6	5	5	5	5	5	5
Homophily ratio r	0.81	0.74	0.23	0.22	0.11	0.30	0.21	0.22
Label Rate	5.2%	3.6%	48%	48%	48%	48%	48%	48%

B Hyperparameters and Search Space

All baselines and our approach are implemented based on the standard implementation in the DGL library [43] using the PyTorch 1.6.0 library with Intel(R) Xeon(R) Gold 6240R @ 2.40GHz CPU and NVIDIA V100 GPU. The following hyperparameters are set for all datasets: weight decay $decay$ = 5e-4; Maximum Epoch E = 500; Layer number L = 2, sampling temperature τ_2 = 1.0. The other dataset-specific hyperparameters are determined by a hyperparameter search tool - NNI for each dataset, including hidden dimension F, learning rate lr, loss weight λ and κ, fusion factor α, and distillation temperature τ_1. The hyperparameter search space is shown in Tab. A2, and the model with the highest validation accuracy is selected for testing. The best hyperparameter choices are available in the supplementary.

Table A2: Hyperparameter search space.

Hyperparameters	Search Space
Hidden Dimension F	[64, 128, 256]
Learning Rate lr	[1e-2, 5e-3, 1e-3]
Loss Weight λ	[0.1, 0.5, 1.0]
Loss Weight κ	[0.1, 0.5, 1.0, 5.0, 20.0]
Fusion Factor α	[0.1, 0.3, 0.5, 1.0]
Temperature τ_1	[1.0, 1.1, 1.2, 1.3, 1.4]

[1]Cornell, Texas, and Wisconsin are three sub-datasets of WebKB1 from `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb`.

D. Training Curves with GKD Loss

The GKD loss can be considered as a "bridge" between the teacher and student models. It regularizes the student model by gradually distilling knowledge from a teacher model (pre-trained on augmented graphs) to a student model, but does not directly affect the learning of the teacher model and graph augmentation. As a result, the trajectories trained with and without GKD loss will not be substantially different, i.e., they both still suffer from distribution shifts, as shown in Fig. 5 and Fig. A1. Essentially, the role of GKD loss is not to directly prevent the occurrence of distribution shifts, but to reduce their negative effects (potential negative augmentation) as much as possible and improve the model generalization by knowledge distillation in the presence of distribution shifts, as shown in Table. 1.

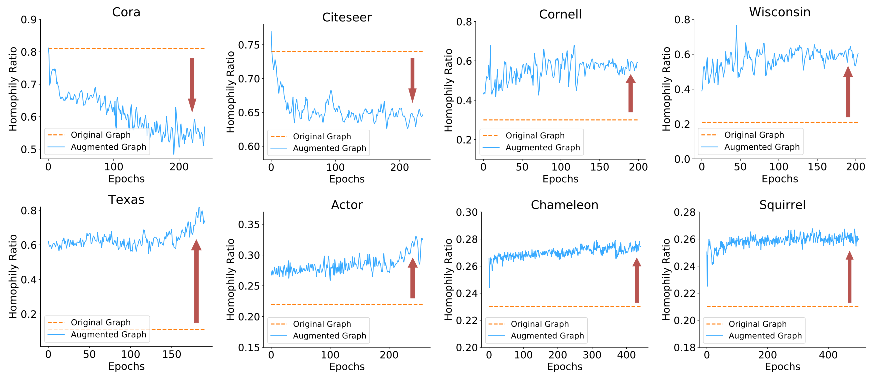
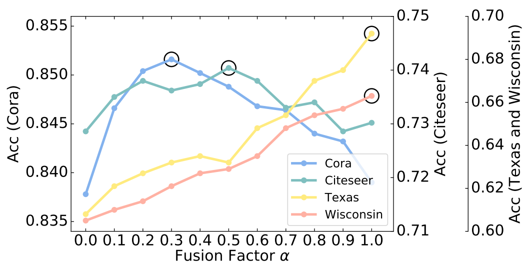


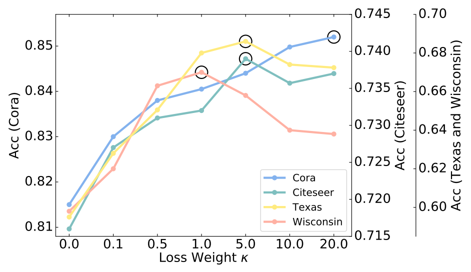
Figure A1: Training curves (w/ GKD Loss) of homophily ratios in the original and augmented graphs.

D. Parameter Sensitivity Analysis

We have evaluated the parameter sensitivity w.r.t two key hyperparameters: fusion factor α and loss weight κ, and results are reported in Fig. A2. In practice, we can determine fusion factor α and loss weight κ by selecting the model with the highest accuracy on the validation set.



(a) Fusion Factor α



(b) Loss Weight κ

Figure A2: Parameter sensitivity analysis on fusion factor α and loss weight κ.

Fusion Factor α**.** The parameter sensitivity w.r.t the fusion factor α defined in Eq. (8) is reported in Fig. 2(a), from which we can observe that (1) When α is set to a small value, the performance gains for all four datasets are reduced, due to the insufficient contextual variations in the augmented graphs. (2) When α is set to a large value, different types of graphs show different hyperparameter sensitivity to α. For homophily graphs, such as Cora and Citeseer datasets, a too-large value of α hurts performance, as it may cause overly severe distribution shifts in the direction of reducing the homophily ratio. For heterophily graphs, such as Texas and Wisconsin datasets, they generally reach the best performance at $\alpha = 1$, where the augmented graph tends to exhibit a higher homophily ratio than the original graph. The difference in the parameter sensitivity of these two types of graphs comes mainly from the difference in their directions of the distribution shifts, as shown in Fig. 5.

Loss Weight κ**.** As can be observed from Fig. 2(b), the loss weight κ plays a critical role in the KDGA framework. If we set the loss weight $\kappa = 0$, i.e., completely remove the GKD loss, the model performance will deteriorate to be the same as the vanilla implementations, resulting in the poorest performance compared to other settings. In practice, we find that setting κ to a non-zero value, i.e., training with GKD loss, always achieves better performance than training without GKD loss (setting $\kappa = 0$), which demonstrates the effectiveness of the proposed GKD loss. Moreover, we find that the model performance can be further improved as the loss weight κ increases, but the performance gains reduce when κ becomes too large, probably because a too large GKD loss weight κ tends to weaken the contribution of label information (i.e., the loss of $\mathcal{L}_{cla}$) in the semi-supervised learning.