Data Augmentation for Deep Graph Learning: A Survey

Kaize Ding[1], Zhe Xu[2], Hanghang Tong[2], Huan Liu[1]
[1]Arizona State University
[2]University of Illinois Urbana-Champaign
kaize.ding@asu.edu, zhexu3@illinois.edu, htong@illinois.edu, huanliu@asu.edu

ABSTRACT

Graph neural networks, a powerful deep learning tool to model graph-structured data, have demonstrated remarkable performance on numerous graph learning tasks. To address the data noise and data scarcity issues in deep graph learning, the research on graph data augmentation has intensified lately. However, conventional data augmentation methods can hardly handle graph-structured data which is defined in non-Euclidean space with multi-modality. In this survey, we formally formulate the problem of graph data augmentation and further review the representative techniques and their applications in different deep graph learning problems. Specifically, we first propose a taxonomy for graph data augmentation techniques and then provide a structured review by categorizing the related work based on the augmented information modalities. Moreover, we summarize the applications of graph data augmentation in two representative problems in data-centric deep graph learning: (1) reliable graph learning which focuses on enhancing the utility of input graph as well as the model capacity via graph data augmentation; and (2) low-resource graph learning which targets on enlarging the labeled training data scale through graph data augmentation. For each problem, we also provide a hierarchical problem taxonomy and review the existing literature related to graph data augmentation. Finally, we point out promising research directions and the challenges in future research.

1. INTRODUCTION

Graphs have been widely used for modeling a plethora of structured or relational systems, such as social networks, knowledge graphs, and academic graphs, where nodes represent the entities and edges denote the relations between entities. As a powerful deep learning tool to distill the knowledge behind graph-structured data, graph neural networks (GNNs) which generally follow a recursive message-passing scheme, have drawn a surge of research interest lately. Owing to its state-of-the-art performance, deep graph learning (DGL) nowadays has achieved remarkable success in a wide spectrum of graph analytical tasks [20; 104; 22].

Despite the superb power of GNNs, their effectiveness in DGL largely depends on high-quality input training graph(s) and ground-truth labels. The performance of GNNs tends to be delicate on real-world graphs, mainly because of their incapability of handling the following challenges: (1) On the one hand, prevailing DGL models are predominantly designed for the supervised or semi-supervised setting where sufficient ground-truth labels are available [23; 21]. Considering the fact that data labeling on graphs is always time-consuming, labor-intensive, and rarely complete, the overreliance on labeled data poses great challenges to DGL models in real-world scenarios. Meanwhile, there are increasingly more tasks and domain-specific applications that are low-resource, having a paucity of labeled training examples. When ground-truth labels are extremely scarce, DGL models may easily overfit and be hard to generalize, losing their efficacy in solving various downstream DGL tasks [92; 21]. (2) On the other hand, real-world graphs are usually extracted from complex interaction systems which inevitably contain redundant, erroneous, or missing features and connections. In addition, the noxious manipulations from adversaries as well as the inherent limitations of GNNs such as the oversmoothing issue [67] also bring additional challenges to the success of reliable DGL. Directly training GNN-based models on such inferior graphs that are not clean and consistent with the properties of GNNs might lead to serious performance degradation [18].

To improve the sufficiency and quality of training data, data augmentation is proposed as an effective tool to augment the given input data by either slightly modifying existing data instances or generating synthetic instances from existing ones. The importance of data augmentation has been well recognized in the computer vision [91] and natural language processing domains [149] in the past few years. More recently, data augmentation techniques have also been explored in the graph domain to push forward the performance boundary of DGL and demonstrated promising results [149; 85]. Apart from conventional image or text data, graph-structured data is known to be far more complicated with heterogeneous information modalities and complex graph properties, yielding a broader design space as well as the additional challenges for graph data augmentation (GraphDA). Though this line of research has been actively conducted lately, the problem of GraphDA has not been well formulated and researchers commonly adopt GraphDA techniques (e.g., edge perturbation, feature masking) arbitrarily without clear preference. Hence, it poses great challenge for researchers to grasp the design principles behind and further leverage them to solve specific DGL problems. Therefore, a timely and systematic review of GraphDA is of great benefit to be aware of existing research in this field and what the challenges are for conducting future research.

Contributions. In this work, we present a forward-looking

and up-to-date survey for GraphDA and its applications in solving data-centric DGL problems. In summary, our major contributions are as follows:

- To the best of our knowledge, this is the first survey for GraphDA. We provide a formal formulation for this emerging research area and review the related recent advances, which can facilitate the understanding of important issues to promote future research.
- We present a comprehensive taxonomy of GraphDA which categorizes the existing techniques in terms of the target augmentation modality (i.e., feature-oriented, structure-oriented, and label-oriented) and provides a clear design space for developing and customizing new GraphDA methods for different DGL problems.
- We discuss the applications of GraphDA in solving two major research problems in data-centric DGL, i.e., optimal graph learning and low-resource graph learning, and review the prevalent learning paradigms for solving specific sub-problems. We also outline the open issues and promising future directions in this area.

Connection to Existing Surveys. Although there are a few survey papers [73; 116; 123] have related content to GraphDA, those surveys predominately focus on a single DGL problem such as graph self-supervised learning or graph adversarial defense. Other data-centric DGL problems related to GraphDA are largely overlooked and the corresponding GraphDA techniques are rarely discussed. In contrast, our survey includes a detailed and systematic review of GraphDA techniques and their corresponding applications for solving two most representative data-centric DGL problems. Meanwhile, we also discuss a list of under-explored directions in GraphDA, which can shed great light on future DGL research.

Structure Overview. This survey is structured as follows. Section 2 first gives background on graph neural networks (GNNs) and deep graph learning. Then in Section 3 we provide a comprehensive taxonomy of GraphDA techniques based on the focused augmentation modality of the input graph(s). In the following two sections, we describe the applications of GraphDA techniques for solving two data-centric DGL problems, i.e., *Low-resource Graph Learning* (Section 4) and *Reliable Graph Learning* (Section 5). Specifically, Section 4 includes the techniques based on GraphDA for solving graph self-supervised learning and graph semi-supervised learning. Section 5 covers the content of improving the robustness, expressiveness, and scalability of DGL models from the data augmentation perspective. Within each subsection, we introduce the methods grouped by their related GraphDA techniques. Finally, Section 6 discusses challenges and future directions in GraphDA.

2. PRELIMINARIES

2.1 Notations and Definitions

We briefly introduce the main symbols and notations used throughout this paper. We use bold uppercase letters for matrices (e.g., $\mathbf{X}$), bold lowercase letters for vectors (e.g., $\mathbf{v}$), lowercase and uppercase letters for scalars (e.g., d, n), and calligraphic letters for sets (e.g., $\mathcal{N}$). We use $\mathbf{X}[i,j]$ to represent the entry of matrix $\mathbf{X}$ at the i-th row and the j-th column, $\mathbf{X}[i,:]$ to represent the i-th row of matrix $\mathbf{X}$, and $\mathbf{X}[:,j]$ to represent the j-th column of matrix $\mathbf{X}$. Similarly, $\mathbf{v}[i]$ denotes the i-th entry of vector $\mathbf{v}$.

For the general purpose, we focus on undirected attributed graphs in this survey. A graph can be represented as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ denotes the node set $\{v_i\}_{i=1}^n$ and $\mathcal{E}$ denotes the edge set $\{e_i\}_{i=1}^m$. In matrix form, it can also be represented as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ denotes the adjacency matrix and $\mathbf{X} \in \mathbb{R}^{n\times d}$ denotes the node feature matrix. Here n is the number of nodes, m is the number of edges, and d is the feature dimension. For supervised tasks, a part of the node labels $\mathbf{y} \in \mathbb{R}^n$, the edge labels $\mathbf{Y} \in \mathbb{R}^{n\times n}$, and the graph label y are provided. For tasks with multiple graphs (e.g., graph-level tasks), we appropriately use subscripts to describe graphs and corresponding components. For example, $\mathbf{A}_i$ denotes the adjacency matrix of the i-th graph $\mathcal{G}_i$.

2.2 Graph Neural Networks

Graph neural networks (GNNs) [120][143] are the extension of the neural network models [3] onto graph data. They show great flexibility and strong expressiveness to extract representations of various graph components and are becoming the core modules of broad graph learning tasks. In this subsection, we will briefly introduce the general message passing formulas [33] of the mainstream GNNs as they are widely adopted by the works remaining of this paper.

The message passing framework of GNNs can be mathematically presented as follows:

$$\mathbf{m}_i^{t+1} = \sum_{j\in N(i)} \texttt{Message}(\mathbf{h}_i^t, \mathbf{h}_j^t), \tag{1a}$$

$$\mathbf{h}_i^{t+1} = \texttt{Update}(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}), \tag{1b}$$

where $\mathbf{h}_i^t$ denotes the representation of the node i at t-th layer and $\mathbf{m}_i^{t+1}$ denotes the message aggregates on the node i at the $(t+1)$-th layer. The initial node representation is the node feature (i.e., $\mathbf{h}_i^0 = \mathbf{X}[i]$) and the node neighborhood can be represented by the adjacency matrix (i.e., $\mathbf{A}[i,j] = 1$ is equivalent to $j \in N(i)$). For graph-level tasks, graph representation $\mathbf{h}_G$ can be obtained through a `Readout` function as follows:

$$\mathbf{h}_G = \texttt{Readout}(\mathbf{H}), \tag{2}$$

where $\mathbf{H}$ can be the node representations from the final layer or any intermediate layer.

2.3 Deep Graph Learning Tasks

In this subsection, we introduce several mainstream DGL tasks on which GraphDA techniques are widely used. We categorize tasks according to their objective graph components (i.e., node, edge, and graph).

Node-level DGL Tasks. Node-level DGL tasks aim to find a mapping p_ϕ from the given graphs to node properties by minimizing a utility loss $L_{\texttt{util}}$ as follows:

$$\phi^* = \arg\min_\phi L_{\texttt{util}}\Big(p_\phi(\mathcal{G}), \mathbf{y}\Big),$$

whose typical example is *semi-supervised node classification*. Given the labels of partial nodes for training, the goal is to predict the labels of (a part of) unlabelled nodes. A classic implementation of a node classifier is a node encoder (e.g., GNNs) working with a multi-class classifier (e.g., an MLP).

Edge-level DGL Tasks. Edge-level DGL tasks focus on finding a mapping p_ϕ from the given graphs to edge proper-

ties which can be presented as below:

$$\phi^* = \arg\min_{\phi} L_{\texttt{util}}\Big(p_\phi(\mathcal{G}), \mathbf{Y}\Big).$$

Taking *link prediction* as an example, its goal is to discriminate if there is an edge between specified node pair. A common implementation is a binary GNN classifier whose input is the edge embeddings (e.g., the aggregation of node embeddings of the head and tail nodes).

Graph-level DGL Tasks. A graph-level task considers every graph as a data sample and infers the property of graph(s) by a mapping p_ϕ. Its mathematical formulation is as follows:

$$\phi^* = \arg\min_{\phi} L_{\texttt{util}}\Big(\{p_\phi(\mathcal{G}_i)\}, \{y_i\}\Big).$$

For instance, in the *graph classification* task, some labeled graphs are provided and the goal is to predict the labels of graphs of interest. A general solution is to aggregate the node embeddings into a graph embedding via a readout function and feed the graph embeddings into a classifier.

3. TECHNIQUES OF GRAPH DATA AUGMENTATION

The goal of graph data augmentation (GraphDA) is to find a transformation function $f_\theta(\cdot) : \mathcal{G} \to \tilde{\mathcal{G}}$ to generate augmented graph(s) $\{\tilde{\mathcal{G}}_i = (\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i)\}$ that can enrich or enhance the preserved information from the given graph(s). In terms of whether the parameter θ can be updated or not during the learning process, most, if not all, of the GraphDA methods can be classified to: *non-learnable* and *learnable* methods. If the augmentation method is *non-learnable*, we can simply omit the parameter θ for brevity.

In general, as the ultimate goal of GraphDA is to improve the GNN model performance on downstream learning tasks, we need to consider them together during the learning process. We denote the augmentation loss as L_{aug} whose goal is to regularize the augmented graph(s) to be close to the given graph(s), and denote the utility loss that measures the GNN performance on specific downstream tasks as L_{utility}. In terms of training strategies, most, if not all, of the *learnable* GraphDA methods can be divided into three categories: (1) decoupled training, (2) joint training, and (3) bi-level optimization. The workflow of each scheme is shown in Figure 1 and the details can be found as follows:

Decoupled Training (DT). In this training scheme, the augmenter f_θ and the predictor p_ϕ are independently trained in a two-stage paradigm. Specifically, the augmenter is first learned with augmentation loss $L_{\texttt{aug}}$. After that, the prediction model p_θ is trained on the augmented graph under the supervision of specific downstream tasks (i.e., L_{util}). The learning process can be formulated as:

$$\begin{aligned}\theta^* &= \arg\min_{\theta}\ L_{\texttt{aug}}\Big(\{\mathcal{G}_i\}, \{f_\theta(\mathcal{G}_i)\}\Big),\\ \phi^* &= \arg\min_{\phi} L_{\texttt{util}}\Big(p_\phi, \{f_{\theta^*}(\mathcal{G}_i)\}\Big).\end{aligned} \tag{3}$$

Joint Training (JT). In the joint training scheme, the augmenter f_θ and the predictor p_ϕ are jointly trained with the augmentation loss L_{aug} and utility loss L_{util}. This learning process can be also considered as multi-task learning, which

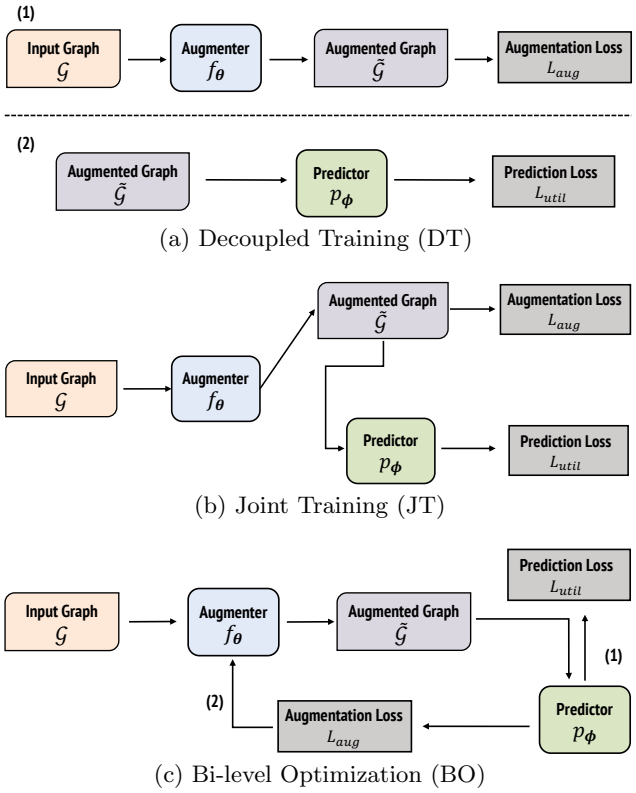


Figure 1: Learnable GraphDA training paradigms.

can be expressed as follows:

$$\begin{aligned}\theta^*, \phi^* = \arg\min_{\theta,\phi}\ & L_{\texttt{aug}}\Big(\{\mathcal{G}_i\}, \{f_\theta(\mathcal{G}_i)\}\Big)\\ &+ L_{\texttt{util}}\Big(p_\phi, \{f_\theta(\mathcal{G}_i)\}\Big).\end{aligned} \tag{4}$$

Bi-level Optimization (BO). Another training scheme of GraphDA for DGL is bi-level optimization. Different from joint training, the augmenter f_θ and the predictor p_ϕ are alternatively updated with the augmentation loss L_{aug} and utility loss L_{util}. As shown in Figure 1 (c), the update of the augmenter is based on the optimal updated predictor, which implies a bi-level optimization problem with θ as the upper-level variable and ϕ as the lower-level variable:

$$\begin{aligned}\theta^* &= \arg\min_{\theta}\ L_{\texttt{aug}}\Big(\{\mathcal{G}_i\}, \{f_\theta(\mathcal{G}_i)\}, p_{\phi^*(\theta)}\Big),\\ s.t.\quad \phi^*(\theta) &= \arg\min_{\phi} L_{\texttt{util}}\Big(p_\phi, \{f_\theta(\mathcal{G}_i)\}\Big).\end{aligned} \tag{5}$$

In the following subsections, we provide a systematic taxonomy to cover mainstream GraphDA techniques. Since graphs commonly consist of multiple information modalities, GraphDA techniques can be naturally divided into three categories based on the augmentation modality, including: *feature-oriented*, *structure-oriented*, and *label-oriented* techniques. Specifically, we summarize the commonly used techniques in terms of each augmentation modality and clearly illustrate their augmentation strategies. Figure 2 (left) summarizes our proposed taxonomy for GraphDA techniques.

3.1 Structure-oriented Augmentations

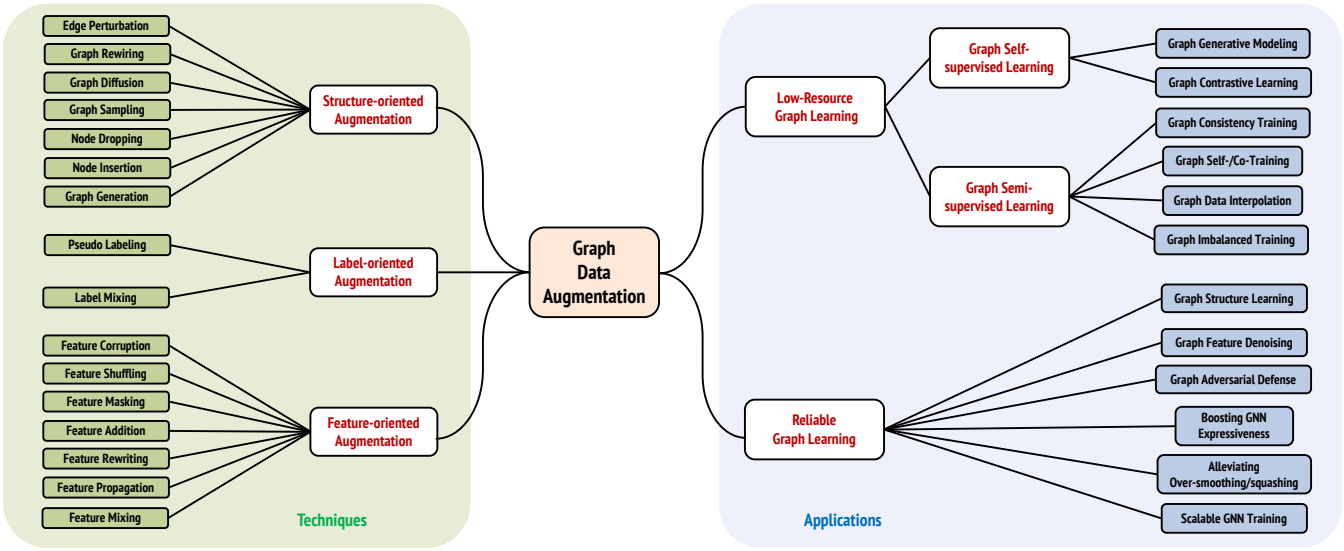


Figure 2: Proposed taxonomy of Graph Data Augmentation (GraphDA) techniques and applications.

Different from i.i.d. data, graphs are inherently relational where the connections (i.e., edges) between data instances (i.e., nodes) are unique and essential for understanding and analyzing graphs. Given an input graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, a structure-oriented GraphDA operation focuses on augmenting the adjacency matrix $\mathbf{A}$ of the input graph. We summarize the representative ones as follows.

Edge Perturbation. Perturbing the given graph structure, e.g., randomly adding or dropping edges, is a widely adopted GraphDA method in different DGL tasks [101; 132; 131; 153]. Mathematically *edge perturbation* keeps the original node order and rewrites a part of the entries in the given adjacency matrices, which can be defined as follows:

$$\tilde{\mathbf{A}} = \mathbf{A} \oplus \mathbf{C}, \tag{6}$$

where $\mathbf{C}$ is the corruption matrix and $\oplus$ denotes the XOR (exclusive OR) operation. Commonly, the corruption matrix $\mathbf{C}$ is obtained by sampling, i.i.d., from a prior distribution, and $\mathbf{C}_{ij}$ determines whether to corrupt the adjacency matrix at position (i, j). For example, assuming a given corruption rate ρ, we may define the corruption matrix as $\mathbf{C}_{ij} \sim Bernoulli(\rho)$, whose elements in $\mathbf{C}$ are set to 1 individually with a probability ρ and 0 with a probability $1-\rho$.

Graph Rewiring. Though sharing the same basic operation with *edge perturbation*, *graph rewiring* has an opposite augmentation objective, which is to improve the utility of the input graph by rewiring the edges. Instead of perturbing the input graph structure by randomly adding/dropping edges, *graph rewiring* is commonly guided by the learning objective of the downstream task, and the corruption matrix $\mathbf{C}$ is learned or predicted through a specific module.

Graph Diffusion. As another effective structure-wise augmentation strategy for improving the graph utility, *graph diffusion* generates an augmented graph by exploiting the global structure knowledge of the input graph. In certain cases, it is also considered as a *graph rewiring* method [100]. Specifically, graph diffusion injects the global topological information into the given graph structure by connecting nodes with their indirectly connected neighbors with calculated weights. A generalized graph diffusion operation can be formulated as:

$$\tilde{\mathbf{A}} = \sum_{k=0}^{\infty} \gamma_k \mathbf{T}^k, \tag{7}$$

where $\mathbf{T} \in \mathbb{R}^{N \times N}$ is the generalized transition matrix derived from the adjacency matrix $\mathbf{A}$ and γ_k is the weighting coefficient that determines the ratio of global-local information. Imposing $\sum_{k=0}^{\infty} \gamma_k = 1, \gamma_k \in [0,1]$ and $\lambda_i \in [0,1]$ where λ_i are eigenvalues of $\mathbf{T}$, guarantees convergence. Two popular examples of graph diffusion are personalized PageRank (PPR) [81] (i.e., $\gamma_k = \alpha(1-\alpha)^k$) and the heat kernel [62] (i.e., $\gamma_k = e^{-t}\frac{t^k}{k!}$). where α denotes teleport probability in a random walk and t is diffusion time. Closed-form solutions to heat kernel and PPR diffusion are formulated in Eq. (8) and (9), respectively:

$$\tilde{\mathbf{A}}^{\text{heat}} = e^{-(\mathbf{I}-\mathbf{T})t}, \tag{8}$$

$$\tilde{\mathbf{A}}^{\text{PPR}} = \alpha(\mathbf{I} - (1-\alpha)\mathbf{T})^{-1}. \tag{9}$$

Graph Sampling. Graph sampling or subgraph sampling is a commonly used data augmentation technique for graphs. It can be used for different purposes, such as scaling up GNNs [36], and creating augmented views [85], to name a few. The augmented graph is obtained via a sampler $\textsc{Sample}(\mathcal{G})$ which can be vertex-based sampling [50], edge-based sampling [151], traversal-based sampling [85], and other advanced methods such as Metropolis-Hastings sampling [82]. For all the above graph samplers, they commonly return a connected subgraph induced from the sampled nodes. Mathematically, graph sampling can be represented as:

$$\tilde{\mathcal{G}} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{X}}\} = \{\mathbf{A}[idx, idx], \mathbf{X}[idx, :]\}, \tag{10}$$

where idx is a list of index to select the given elements (i.e., rows and columns) from $\mathbf{A}$ and $\mathbf{X}$. In general, the goal of graph sampling is to find augmented graph instances from the input graphs that best preserve desired properties by keeping a portion of nodes and their underlying linkages. For example, in the *graph sparsification* [151; 76] problem,

researchers aim to sample the subgraph which preserves as much task-relevant information as possible.

Node Dropping. In the literature, node dropping is also known as node masking. Specifically, a set of nodes $\hat{\mathcal{V}} = \{v_i \in \mathcal{V}\}$ will be dropped from the input graph, together with their associated edges $\hat{\mathcal{E}} = \{e_i \in \mathcal{E}\}$. This augmentation can be formulated as:

$$\tilde{\mathbf{A}} = \{\mathcal{V} \setminus \hat{\mathcal{V}}, \mathcal{E} \setminus \hat{\mathcal{E}}\}. \tag{11}$$

Note that for attributed graphs, the corresponding node features will also be dropped at the same time.

Node Insertion. Node insertion is commonly used to improve the message-passing or connectivity on the input graph by inserting virtual node(s) [33]. Specifically, node insertion adds an extra set of nodes $\hat{\mathcal{V}} = \{v_i\}$ and a set of edges $\hat{\mathcal{E}} = \{e_i\}$ between $\hat{\mathcal{V}}$ and $\mathcal{V}$ to the original node set $\mathcal{V}$ and the edge set $\mathcal{E}$, respectively:

$$\tilde{\mathbf{A}} = \{\mathcal{V} \cup \hat{\mathcal{V}}, \mathcal{E} \cup \hat{\mathcal{E}}\}. \tag{12}$$

Since node insertion also requires adding additional edges in the new graph, this GraphDA operation is highly related to graph rewiring. Note that for attributed graphs, the corresponding node features also need to be initialized, e.g., using the mean average of all the connected node features.

Graph Generation. Graph generation is commonly used as a GraphDA strategy for improving the scales of training graphs graph-level DGL tasks, e.g., graph classification. Most graph generation methods are expected to automatically learn from observed graphs. Generally, the graph generation process can be presented as follows:

$$\tilde{\mathcal{G}} \sim D_\theta(\mathcal{G}|\{\mathcal{G}_i\}), \tag{13}$$

where $\{\mathcal{G}_i\}$ is the set of observed graphs and here the augmentation function D_θ is a graph generation distribution parameterized by θ. Notice that some techniques such as *graph coarsening* [8] and *graph condensation* [55] whose goals are to generate a new graph from the initial large graph can also be categorized into this augmentation operation. We also consider edge mixing between two (or mode) graphs [35] as one instantiation of graph generation.

3.2 Feature-oriented Augmentations

In this subsection, we review the feature-oriented GraphDA techniques. Generally, given an input graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, a feature-oriented GraphDA operation focuses on performing transformation on the node feature matrix $\mathbf{X}$. Notably, we also consider those methods performing augmentations on the latent feature representations $\mathbf{H}$ as feature-oriented augmentation methods.

Feature Corruption. This GraphDA method aims at adding noises to either the original node features [27] or learned feature representations [127]. For simplicity, here we use $\mathbf{x}_i$ to represent the original node features or learned feature representations:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{r}_i, \tag{14}$$

where $\mathbf{r}_i$ denotes the added feature noise. Note that the feature noise could be either randomly added [101] or learned in an adversarial training fashion [27; 127].

Feature Shuffling. By randomly changing the contextual information through switching rows and columns in the feature matrix, The input feature matrix $\mathbf{X}$ is corrupted to yield augmentations. This method can be formulated as:

$$\tilde{\mathbf{X}} = \mathbf{P}_r\mathbf{X}\mathbf{P}_c, \tag{15}$$

where $\mathbf{P}_r$ and $\mathbf{P}_c$ are row-wise permutation matrix and column-wise permutation matrix, respectively. have exactly one entry of 1 in each row and each column and 0 elsewhere.

Feature Masking. The core operation of feature masking is to set a part of the entries in the node feature matrix $\mathbf{X}$ to 0, which can be formulated as:

$$\tilde{\mathbf{X}} = \mathbf{X} \odot \mathbf{M} \tag{16}$$

where $\mathbf{M}$ is the masking matrix that $\mathbf{M}_{i,j} = 0$ if the j-th element of vector i is masked/dropped, otherwise $\mathbf{M}_{i,j} = 1$. The masking matrix $\mathbf{M}$ is commonly generated by the Bernoulli distribution [132; 131; 98].

Feature Addition. Since the input graph usually lacks informative features in real-world scenarios, Feature Addition can be used to (1) initiate node features on plain graphs to smoothly incorporate into DGL models (e.g., GNNs) and (2) supplement additional graph knowledge that are hard to be captured by GNN models. A straightforward way is to encode proximate/topological information (e.g., node index or node properties) into a feature vector and concatenate with the original node features. In general, Feature Addition can be expressed as:

$$\tilde{\mathbf{x}}_i = [\hat{\mathbf{x}}_i || \mathbf{x}_i], \tag{17}$$

where $||$ denotes the concatenation operation and $\mathbf{x}_i$ could be an empty vector if the input graph is plain.

Feature Rewriting. Considering the fact that the given node features are commonly noisy and incomplete, recovering the clean and complete node features can directly improve the performance of the DGL models. Generally, feature rewriting can be expressed as:

$$\tilde{\mathbf{x}} = \alpha\mathbf{x}_i + \beta\mathbf{b}_i, \tag{18}$$

where α and β are two controlling parameters and $\mathbf{b}_i$ is a feature vector computed in a heuristic or learnable way. For example, Wang et al. propose *feature replacement* [110] that rewrites node features with its neighbors' features. While Xu et al. [125] apply gradient descent-based optimizers to rewrite the node features as parameters.

Feature Mixing. Based on the features of nodes in the input graph, feature mixing can be used to obtain the node features of a synthetic node:

$$\tilde{\mathbf{x}} = \lambda\mathbf{x}_i + (1-\lambda)\mathbf{x}_j, \tag{19}$$

where λ is the mixing coefficient that controls the proportion of information from $\mathbf{x}_i$ and $\mathbf{x}_j$. Notably, feature mixing can also be performed on the intermediate representations learned from two training samples (i.e., Manifold Mixup [102]).

Feature Propagation. Based on Graph Diffusion, Feature Propagation propagates the node features along the graph structure. It is an interpolation method that has also been widely used to augment the node features of the input graph. Mathematically:

$$\tilde{\mathbf{X}} = \tilde{\mathbf{A}}\mathbf{X}, \tag{20}$$

where $\tilde{\mathbf{A}}$ is the new adjacency matrix obtained through different graph diffusion methods.

3.3 Label-oriented Augmentations

Due to the expensive data labeling cost on graphs, label-oriented GraphDA is an important line of work to directly enrich the limited labeled training data. Commonly, there are two groups of strategies.

Pseudo-Labeling. Pseudo-labeling is a semi-supervised learning mechanism that aims to obtain one (or several) augmented labeled set(s), based on their most confident predictions on unlabeled data. Its learning process starts with a base teacher model trained on the labeled set $\mathcal{D}^L$, and then the teacher model is applied to the unlabeled data $\mathcal{D}^U$ to obtain pseudo labels (hard or soft) of unlabeled data. Finally, a subset of unlabeled data $\mathcal{D}^P$ will be used to augment the training data, and the combined data $\mathcal{D}^L \cup \mathcal{D}^P$ can be used to train a student model. In this sense, the label signals can be "propagated" to the unlabeled data samples via the learned teacher model. Note that this learning process could go through multiple rounds until converges by iteratively updating the teacher model with the current student model.

Label Mixing. In order to enlarge the scale of training samples, we can directly interpolate the training samples based on the labeled examples. Generally, Mixup [140] constructs virtual training samples via feature mixing and label mixing:

$$\begin{aligned} \tilde{\mathbf{x}} &= \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j, \\ \tilde{\mathbf{y}} &= \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j, \end{aligned} \tag{21}$$

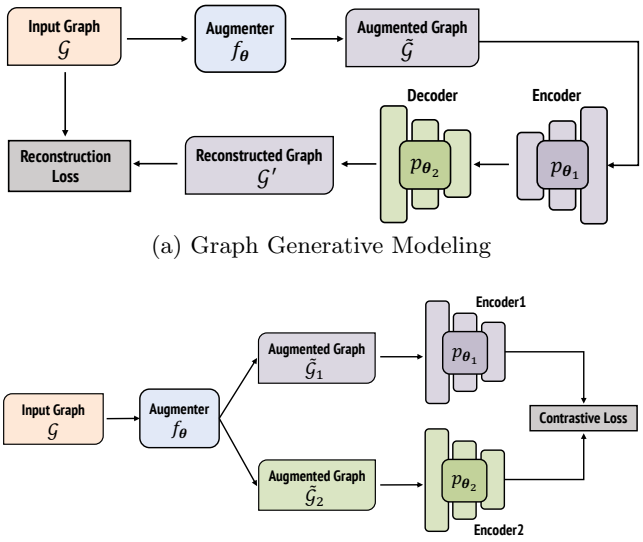
where $(\mathbf{x}_i, \mathbf{y}_i)$ and $(\mathbf{x}_j, \mathbf{x}_j)$ are two labeled samples randomly sampled from the training set, and $\lambda \in [0, 1]$. In this way, Mixup methods extend the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated target labels.

4. GRAPH DATA AUGMENTATION FOR LOW-RESOURCE GRAPH LEARNING

The shortage of ground-truth labels has been a longstanding and notorious problem for learning effective DGL. To advance the research on *Low-resource Graph Learning*, GraphDA has been actively investigated and shows promising results. In this section, we discuss the applications of GraphDA for solving both *Graph Self-Supervised Learning* and *Graph Semi-Supervised Learning*. We summarize the representative works in Table 1 and Table 2, respectively.

4.1 Graph Self-Supervised Learning

Graph Generative Modeling. Recently data augmentation has been widely used for graph self-supervised learning (SSL). Based on the idea of Graph AutoEncoders (GAE) [58; 20], graph generative modeling methods perform data augmentation on the input graphs through either ***edge perturbation***, ***feature masking***, or ***node dropping (masking)*** then learn the node representations by reconstructing feature or/and structure information from the augmented graphs (as shown in Fig. 3 (a)). For example, Denoising Link Reconstruction [45] randomly drops existing edges to obtain the perturbed graph and tries to recover the discarded connections with a pairwise similarity-based decoder



(a) Graph Generative Modeling

(b) Graph Contrastive Learning

Figure 3: Comparison between the workflows of Graph Generative Modeling and Graph Contrastive Learning.

trained by the cross-entropy loss. Given a graph with its nodes and edges randomly masked, GPT-GNN [44] generates one masked node and its related edges jointly and optimizes the likelihood of the node and edges generated in the current iteration. GraphBert [141] samples linkless subgraphs and pre-trains a graph transformer model via node feature reconstruction and graph structure recovery. You et al. [133] define the graph completion pretext task which aims to recover the masked feature of target nodes based on their neighbors' features and connections. MGM [78] tries to reconstruct the masked node/edge features for learning the GNNs for molecule generation. GMAE [10] and GraphMAE [43] first randomly mask the features of some nodes and then their decoders reconstruct the original features of the masked nodes, while MGAE [97] tries to reconstruct the masked edges based on the augmented graphs.

Graph Contrastive Learning. Motivated by the recent breakthroughs in contrastive visual feature learning, data augmentation has also been widely used for Graph Contrastive Learning (GCL). As illustrated in Fig. 3 (b), GCL methods try to generate augmented examples from the input and view two augmented examples from the same original sample as a positive pair, while those from different original samples are negative pairs. By applying contrastive learning loss, the positive pairs will be pulled together and the negative pairs will be pushed away in the latent space. Therefore, GraphDA plays an essential role in GCL. As a pioneering work, DGI [101] applies ***feature shuffling*** and ***edge perturbation*** to obtain the negative pairs of graphs, then a contrastive objective is proposed to maximize the mutual information between node embeddings and a global summary embedding. Another popular GraphDA method for GCL is ***graph sampling***. As an example, GCC [85] proposes to sample subgraphs as contrastive instances to pre-train the graph encoder, which can be used for different downstream tasks with either freezing or full fine-tuning strategy. SUBGCON [50] also utilizes a subgraph sampler

Table 1: Summary of representative GraphDA works for graph *self-supervised* learning. DT, JT, BO stand for decoupled training, joint training, and bi-level optimization, respectively

Topic	Name	Ref.	Year	Venue	Task Level	Augmented Data Modality			Learnable	Augmentation Technique
						Structure	Feature	Label		
Graph Generative Modeling	GraphMAE	[43]	2022	KDD	node & graph	✓				feature masking
	GMAE	[10]	2022	arXiv	noded & graph	✓				node dropping
	MGAE	[97]	2022	arXiv	node & edge	✓				edge perturbation (dropping)
	MGM	[78]	2021	Nature Communications	graph		✓			feature corruption
	GPT-GNN	[44]	2020	KDD	node	✓	✓			edge perturbation (dropping) feature masking
	MTL	[133]	2020	ICML	node		✓	✓	✓(JT)	pseudo-labeling/feature corruption
	GraphBert	[141]	2020	arXiv	node	✓	✓			graph sampling
	Pre-train	[45]	2019	arXiv	node & edge & graph	✓		✓		edge perturbation/pseudo-labeling
Graph Contrastive Learning	S^3-CL	[24]	2023	AAAI	node		✓	✓	✓(JT)	feature propagation/pseudo labeling
	SUGRL	[79]	2022	AAAI	node	✓	✓			feature shuffling/graph sampling
	LG2AR	[39]	2022	arXiv	node & graph	✓	✓		✓(BO)	node dropping/edge perturbation graph sampling/feature corruption
	BGRL	[98]	2022	ICLR	node	✓	✓			edge perturbation (dropping) feature masking
	ARIEL	[28]	2022	TheWebConf	node	✓	✓		✓(JT)	feature corruption/edge perturbation
	AD-GCL	[95]	2021	NeurIPS	graph	✓			✓(JT)	edge perturbation (dropping)
	JOAO	[131]	2021	ICML	graph	✓	✓		✓(BO)	node dropping/edge perturbation graph sampling/feature corruption
	GCA	[153]	2021	TheWebConf	node	✓	✓			edge perturbation (dropping) feature masking
	MERIT	[52]	2021	IJCAI	graph	✓	✓			graph diffusion/edge perturbation graph sampling/feature masking
	CSSL	[137]	2021	AAAI	graph	✓				node dropping/node insertion edge perturbation
	GraphCL	[132]	2020	NeurIPS	graph	✓	✓			node dropping/edge perturbation graph sampling/feature corruption
	GCC	[85]	2020	KDD	graph	✓				graph sampling
	SUBGCON	[50]	2020	ICDM	node	✓				graph sampling
	MVGRL	[38]	2020	ICML	node & graph	✓				graph diffusion/graph sampling
	GRACE	[152]	2020	arXiv	node	✓	✓			edge perturbation (dropping) feature masking
	DGI	[101]	2019	ICLR	node		✓			feature corruption

based on Personalized PageRank to sample the augmented subgraph and perform contrastive learning between the representations of the central node and the sampled subgraph. It is noteworthy that many GCL methods usually leverage combinations of augmentation strategies. For instance, MVGRL [38] first augments the input graph via ***graph diffusion***. Afterward, two graph views are generated by ***subgraph sampling*** and the model learns to contrast node representations to global representations across the two views. GRACE [152] adopts two augmentation strategies, i.e., ***edge perturbation (dropping)*** and ***feature masking***, to generate augmented views of graph data. It jointly considers both intra-view and inter-view negative pairs for the contrastive learning objectives. gCooL [69] uses the same augmentations and considers the community information in GCL. GraphCL [132] considers four GraphDA operations: ***node dropping***, ***edge perturbation***, ***feature masking***, and ***subgraph sampling***, while MERIT [52] leverages ***graph diffusion***, ***edge perturbation (dropping)***, ***subgraph sampling***, and ***feature masking*** to generate augmented graphs. CSSL [137] augments graphs with the ***edge perturbation (dropping)*** and ***node dropping***. Recent work SUGRL [79] leverages ***feature shuffling*** and ***graph sampling***, S^3-CL uses ***feature propagation*** and ***pseudo labeling*** to efficiently perform GCL.

Different from the aforementioned pre-defined GraphDA strategies, another line of research proposes to perform data augmentation on graphs in a learnable manner. Specifically, Zhu et al. [153] propose a joint, adaptive data augmentation scheme based on ***edge perturbation (dropping)*** and ***feature masking*** to provide diverse contexts for nodes in different views, so as to boost optimization of the contrastive objective. GASSL [127] is an adversarial self-supervised learning framework for learning unsupervised representations of graph data without any handcrafted views. It learns to perform ***feature corruption*** on either the input or latent space. AD-GCL [95] enables GNNs to avoid capturing redundant information during the training by optimizing ***edge perturbation (dropping)*** strategy used in GCL in an adversarial fashion. To automatically select optimal augmentation combinations for the given graph dataset, JOAO [131] and LG2AR [39] are proposed to automatically select augmentations from a given pool of augmentation types: {***node dropping***, ***subgraph sampling***, ***edge perturbation***, ***feature masking***}. It is important to note that an expanding body of literature, such as [90], has emerged in the field of GCL. However, due to space constraints, we are unable to provide an exhaustive coverage of all these works.

4.2 Graph Semi-Supervised Learning

Graph Consistency Training. Similar to the idea of contrastive learning, Consistency Training [122] leverages

Table 2: Summary of representative GraphDA works for graph *semi-supervised* learning. DT, JT, BO stand for decoupled training, joint training, and bi-level optimization, respectively

Topic	Name	Ref.	Year	Venue	Task Level	Augmented Data Modality			Learnable	Augmentation Technique
						Structure	Feature	Label		
Graph Consistency Training	D^2-PT	[72]	2023	KDD	node	✓	✓	✓	✓(JT)	feature propagation/pseudo labeling
	NASA	[5]	2022	AAAI	node	✓		✓		edge perturbation (dropping)
	MH-Aug	[82]	2021	NeurIPS	node	✓		✓		graph sampling
	SCR	[139]	2021	arXiv	node	✓				graph rewiring
	NodeAug	[110]	2020	KDD	node	✓	✓	✓		feature rewriting/graph rewiring pseudo labeling
	GRAND	[30]	2020	NeurIPS	node		✓	✓		node dropping/feature propagation pseudo labeling
Graph Self-/Co-Training	Meta-PN	[21]	2022	AAAI	node			✓	✓(BO)	pseudo-labeling
	NRGNN	[18]	2021	KDD	node	✓		✓	✓(DT)	graph rewiring/pseudo-labeling
	PTA	[25]	2021	TheWebConf	node			✓		pseudo-labeling
	CGCN	[47]	2020	AAAI	node			✓	✓(JT)	pseudo-labeling
	M3S	[92]	2020	AAAI	node			✓	✓(JT)	pseudo-labeling
	Co-GCN	[68]	2020	AAAI	node	✓			✓(DT)	feature masking/pseudo labeling
	ST-GCNs	[67]	2018	AAAI	node			✓	✓(DT)	pseudo-labeling
Graph Data Interpolation	G-Transplant	[83]	2022	AAAI	graph	✓	✓	✓	✓(DT)	label mixing/graph rewiring
	G-Mixup	[109]	2021	TheWebConf	graph		✓	✓	✓(DT)	graph generation/label mixing
	GraphMix	[103]	2021	AAAI	node		✓	✓		feature mixing/label mixing/pseudo labeling
	ifMixup	[35]	2021	arXiv	graph	✓	✓	✓		feature mixing/label mixing/graph generation
Graph Imbalanced Training	GATSMOTE	[74]	2022	Mathematics	node	✓	✓	✓	✓(JT)	feature rewriting/node insertion/graph rewiring
	GNN-CL	[69]	2022	SDM	node	✓	✓	✓	✓(JT)	feature mixing/node insertion/graph rewiring
	GraphMixup	[117]	2022	ECML-PKDD	node	✓	✓	✓	✓(BO)	feature mixing/label mixing/graph rewiring
	GraphSMOTE	[150]	2021	WSDM	node	✓	✓	✓	✓(JT)	feature mixing/node insertion/graph rewiring
	DPGNN	[108]	2021	arXiv	node			✓		pseudo-labelling
	D-GCN	[106]	2021	ICCSAE	node	✓				graph rewiring

unlabeled data to improve model performance by enforcing the consistency across predictions learned from different stochastic augmentations of the same input. As one effective semi-supervised learning paradigm, consistency training has also been explored in learning graph neural networks under low-resource settings. For example, NodeAug [110] uses local structure-wise augmentation operations (i.e., ***feature corruption***, and ***edge perturbation***), and minimizes the KL-divergence between the node representations learned from the original graph and augmented graph. GRAND [30] creates multiple different augmented graphs with ***node dropping*** and ***feature masking***, followed by ***feature propagation***. Then the consistency loss is applied to minimize the distances of the representations learned from the augmented graphs. Following GRAND, Zhang et al. [139] propose to use two ***edge perturbation*** methods – DropEdge [86] or DropNode [30] as the augmentation strategies and leverage a mean-teacher consistency regularization to guide the training of the GNN model by calculating a consistency loss between the student and teacher models. To enable consistency training on large-scale graphs, Hawkins et al. [40] propose to use ***graph sampling*** to generate different neighborhood subgraph expansions and ensemble the predictions to provide pseudo labels. To avoid the detrimental effects of arbitrary augmentations, Park et al. [82] propose a ***graph sampling*** method MH-Aug that uses the Metropolis-Hastings algorithm to obtain the augmented samples of the input graph, and adopt consistency training to better utilize the unlabeled data. Following the idea of ***graph rewiring***, NASA [5] generates graph augmentations with high consistency and diversity by replacing immediate neighbors with remote neighbors and enforcing the predictions of augmented neighbors to be consistent.

Graph Self-/Co-Training. To address the data scarcity issue, one effective solution is to leverage the unlabeled data to augment the limited labeled data. Following the idea of ***pseudo labeling***, self-training [128] imputes the labels on unlabeled data based on a teacher model trained with limited labeled data, and it has become a prevailing paradigm to solve the problem of semi-supervised node classification when training data is limited. Among those methods, Li et al. [67] first combine GCNs with self-training to expand supervision signals. CGCN [47] generates pseudo labels by combining variational graph auto-encoder with Gaussian mixture models. Furthermore, M3S [92] propose the multi-stage self-training and utilize a clustering method to eliminate the pseudo labels that may be incorrect. Similar ideas can also be found in [18]. In addition, recent research [25; 21] adopt label propagation as the teacher model to generate pseudo labels that encode valuable global structure knowledge.

Similar to Self-training, Co-training [4] has also been investigated for augmenting the training set with unlabeled data. It learns two classifiers with initial labeled data on the two views respectively and lets them label unlabeled data for each other to augment the training data. Li et al. [68] develop a novel multi-view semi-supervised learning method Co-GCN based on ***feature masking***, which unifies GCN and co-training into one framework.

Graph Data Interpolation. Another way of obtaining extra training examples is to use interpolation-based data augmentation strategy, such as Mixup [140] to generate synthetic training examples (i.e., ***node insertion***) based on ***feature mixing*** and ***label mixing***. While unlike images or natural sentences, graphs have arbitrary structure, it re-

mains a non-trivial task to identify the meaningful connections between the original nodes and synthetic nodes. Also, due to the cascading effect of graph data, even simply adding an edge into a graph can dramatically change the graph semantic meanings. To circumvent those challenges, Manifold Mixup [102] has been applied to graph data interpolation. Specifically, GraphMix [103] trains a fully connected network (FCN) jointly with the GNN via parameter sharing, and the FCN is learned based on Manifold Mixup and ***pseudo-labeling***, which can effectively train GNNs for semi-supervised node classification. Similarly, Wang et al. [109] also follow the idea of Manifold Mixup and interpolate the input features of both nodes and graphs in the embedding space. Those methods leverage a simple way to avoid dealing with the arbitrary structure in the input space for mixing a node or graph pair, through mixing the graph representation learned from GNNs.

For the input-level graph data interpolation, ifMixup [35] targets the Manifold Intrusion issue (mixing graph pairs may naturally create graphs with identical structure but with conflict labels) and first interpolates both the node features and the edges of the input pair based on ***feature mixing*** and ***graph generation***. Graph Transplant [83] is another input-level graph interpolation method that leverages ***graph rewiring*** to mix two dissimilar-structured graphs by replacing the destination subgraph with the source subgraph while preserving the local structure. *G*-Mixup [37] first estimates the graphon of each class and then mixup between the graphons and perform ***graph generation*** to generate interpolated graphs, which improves the generalizability and robustness of GNNs for semi-supervised graph classification.

Graph Imbalanced Training. The class distribution of graph data is inherently imbalanced which follows the power-law distribution. As an example, on the benchmark Pubmed dataset, nodes are labeled into three classes while the minority class only contains 5.25% of the total nodes. Such highly imbalanced data will lead to the suboptimal performance of downstream tasks especially classification tasks and one of the effective solutions is to augment the minority to alleviate the imbalance. To counter this problem, ***node insertion*** has been proven as an effective solution to augment the minority class. Meanwhile, ***feature mixing*** and ***graph rewiring*** are also needed for enable graph imbalanced training. For instance, GraphSMOTE [150] augments the minority class by mixing up the minority nodes and leverages an edge generator to predict neighbor information for those synthetic nodes. GraphMixup [117] first performs interpolation on a node from one target minority class with its nearest neighbors, then adopts an edge prediction module to predict the connections between generated nodes and existing nodes. Following this idea, GATSMOTE [74] and GNN-CL [69] adopts an attention mechanism to generate the edges between the synthetic nodes and the original nodes. On the label level, DPGNN [108] conducts ***pseudo labeling*** via label propagation to enrich the training samples from the minority class. However, many challenges in this topic are still under-explored. For example, if the amount of labeled minority nodes is extremely small, such as few-shot or even one-shot per class, how to transfer knowledge from the majority classes to augment the minority classes is worth studying.

5. GRAPH DATA AUGMENTATION FOR RELIABLE GRAPH LEARNING

One main goal of GraphDA is to achieve *Reliable Graph Learning* in the real-world scenarios by augmenting the input graph(s). Specifically, in this work we focus on improving the robustness, expressiveness, and scalability of DGL models via GraphDA for different challenging learning scenarios. We summarize the representative works in Table 3.

Graph Structure Learning. Due to various reasons such as fake connections [42], over-personalized users [13] and construction heuristics, the given graph structure is not optimal for downstream graph learning tasks. Graph structure learning is proposed as a solution for the above challenge. From a general sense, the core technique used for structure learning is ***graph rewiring***.

A series of methods rewire the given graphs following various node similarity metrics. In most cases, such metrics are learned from the given graph topology. For example, GAUG [149] and IDGL [16] train edge predictors based on learned node embeddings. Besides, from the optimization perspective, it is feasible to directly incorporate the graph data (e.g., adjacency matrix) itself as a part of the optimization variables. Based on that, the ***graph rewiring*** process is essentially guided by the optimization objective. TO-GNN [126] is a representative work whose loss functions include smoothness-related regularizations and the update of graphs is gradient descent-based. In addition, instead of optimizing the graph itself, an interesting idea is to optimize the graph-related distributions (e.g., graph generation distributions and edge dropping distributions). After that, ***graph rewiring*** can be conducted by sampling from those distributions. A representative work is LDS [31] which assumes that every edge is sampled from an independent Bernoulli distribution. Other representative works in this line include Bayesian-GCNN [145], GEN [105], NeuralSparse [151], and PTDNet [76].

Graph Feature Denoising. Compared to structure denoising, the research on graph feature denoising has received less attention. In general, most of the work is developed based on ***feature rewriting***. For instance, AirGNN [71] regularizes the l_{21} norm between the input node features and convoluted node features such that the model is more tolerant against abnormal features. To handle missing node features, a special case of suboptimal initial node features, ***feature propagation*** [88] diffuses the features from observed nodes to neighbors whose features are missing based on the heat diffusion equation; in other words, it imputes the missing node features with aggregated features from the neighborhood of the target nodes. GCNMF [96] explicitly formulate the missing node features by Gaussian mixture models whose parameters are inferred from the downstream tasks. An effort named SAT [15] reconstructs the missing features through the feature distribution, which is inferred from the topology distribution. To handle missing features on heterogeneous information networks, HGNN-AC [51] imputes missing features from neighbor nodes' topology-based node embedding, while HGCA [41] designs a feature augmenter which is trained by maximizing the agreement between the augmented node embedding and the actual node embedding.

Graph Adversarial Defense. Aside from the noise introduced in the data collection phase, GNNs are fragile against adversarial attacks on graph structure and features [93].

Table 3: Summary of representative GraphDA works for *reliable* graph learning. DT, JT, BO stand for decoupled training, joint training, and bi-level optimization, respectively.

Topic	Name	Ref.	Year	Venue	Task Level	Augmented Data Modality			Learnable	Augmentation Technique
						Structure	Feature	Label		
Graph Structure Learning	D^2-PT	[72]	2023	KDD	node	✓	✓	✓	✓(JT)	feature propagation/pseudo labeling
	HES-GSL	[115]	2023	TNNLS	node	✓		✓	✓(JT)	graph rewiring
	KDGA	[114]	2022	NeurIPS	node	✓			✓(JT)	graph rewiring
	DGM	[57]	2022	TPAMI	node	✓	✓		✓(JT)	feature rewriting/graph rewiring
	GEN	[105]	2021	TheWebConf	node	✓			✓(JT)	graph rewiring
	PTDNet	[76]	2021	WSDM	node & edge	✓			✓(JT)	graph sampling
	GAUG	[149]	2021	AAAI	node	✓			✓(DT)	graph rewiring
	HGSL	[147]	2021	AAAI	node	✓			✓(JT)	graph rewiring
	IDGL	[16]	2020	NeurIPS	node & graph	✓			✓(JT)	graph rewiring
	NeuralSparse	[151]	2020	ICML	node	✓			✓(JT)	graph sampling
	TO-GNN	[126]	2019	IJCAI	node	✓			✓(JT)	graph rewiring
	LDS	[31]	2019	ICML	node	✓			✓(BO)	graph rewiring
	PG-LEARN	[119]	2018	CIKM	node	✓			✓(JT)	graph rewiring
Graph Feature Denoising	D^2-PT	[72]	2023	KDD	node	✓	✓	✓	✓(JT)	feature propagation/pseudo labeling
	HGCA	[41]	2022	TNNLS	node		✓		✓(JT)	feature addition
	AirGNN	[71]	2021	NeurIPS	node		✓		✓(JT)	feature rewriting
	GCNMF	[96]	2021	FGCS	node & edge		✓		✓(JT)	feature addition
	HGNN-AC	[51]	2021	TheWebConf	node		✓		✓(JT)	feature addition
	SAT	[15]	2020	TPAMI	node & edge		✓		✓(JT)	feature addition
Graph Adversarial Defense	Gasoline	[125]	2022	TheWebConf	node	✓	✓		✓(BO)	feature rewriting/graph rewiring
	Pro-GNN	[53]	2020	KDD	node	✓			✓(JT)	graph rewiring
	GIB-N	[118]	2020	NeurIPS	node	✓			✓(JT)	graph rewiring
	G-SVD	[26]	2020	WSDM	node	✓			✓(DT)	graph rewiring
	RoGNN	[111]	2020	WCSP	node	✓			✓(JT)	graph rewiring
	GNNGuard	[144]	2020	NeurIPS	node	✓			✓(JT)	graph rewiring
	Flag	[63]	2020	arXiv	node & edge & graph		✓		✓(JT)	feature rewriting
	G-Jaccard	[113]	2019	IJCAI	node	✓				graph rewiring
	GraphAT	[27]	2019	TKDE	node		✓		✓(JT)	feature rewriting
Boosting GNN Expressiveness	LAGNN	[70]	2022	ICML	node & edge & graph		✓		✓(JT)	feature addition
	rGINs	[89]	2021	SDM	graph		✓			feature addition
	GSN	[7]	2021	TPAMI	graph		✓			feature addition
	NGNN	[142]	2021	NeurIPS	graph	✓				graph sampling
	ID-GNN	[130]	2021	AAAI	node & edge & graph	✓	✓			graph sampling/feature addition
	Distance Encoding	[66]	2020	NeurIPS	node & edge & graph		✓			feature addition
	Master Node	[33]	2017	ICML	graph	✓				node insertion
Alleviating Over-Smoothing/ Squashing	SDRF	[100]	2022	ICLR	node	✓				graph rewiring
	ADC	[146]	2021	NeurIPS	node	✓			✓(BO)	graph diffusion
	SHADOW-GNN	[135]	2021	NeurIPS	node & edge	✓				graph sampling
	DropEdge	[86]	2020	ICLR	node	✓				edge perturbation (dropping)
	AdaEdge	[9]	2020	AAAI	node	✓			✓(JT)	graph rewiring
	GDC	[61]	2019	NeurIPS	node	✓				graph diffusion
Scalable GNN Training	GCOND	[55]	2022	ICLR	node	✓	✓	✓	✓(BO)	graph generation
	DosCond	[54]	2022	KDD	node & graph	✓	✓	✓	✓(BO)	graph generation
	GOREN	[8]	2021	ICLR	node	✓	✓	✓	✓(DT)	graph generation
	SpectralGC	[56]	2020	AISTATS	graph	✓			✓(DT)	graph generation
	SIGN	[87]	2020	arXiv	node	✓				graph diffusion
	PPRGO	[6]	2020	KDD	node	✓				graph diffusion
	GBP	[14]	2020	NeurIPS	node	✓				graph diffusion
	GraphSAINT	[136]	2020	ICLR	node	✓				graph sampling
	LADIES	[154]	2019	NeurIPS	node	✓				graph sampling
	Cluster-GCN	[17]	2019	KDD	node	✓			✓(DT)	graph rewiring
	Fast-GCN	[11]	2018	ICLR	node	✓				graph sampling
	GraphSAGE	[36]	2017	NIPS	node	✓				graph sampling

Naturally, conducting GraphDA to recover and enhance (a part of) the poisoned graphs is effective to alleviate the performance degradation. In this section, we discuss recent advances in defending graph adversarial attacks via GraphDA. Using prior knowledge about benign graphs (e.g., feature smoothness) to guide ***graph rewiring*** is effective to defend graph adversarial attacks. For instance, works such as G-SVD [26] and DefenseVGAE [138] restore poisoned graphs into their reconstructed low-rank graphs which shows great empirical effectiveness again adversarial attacks. In

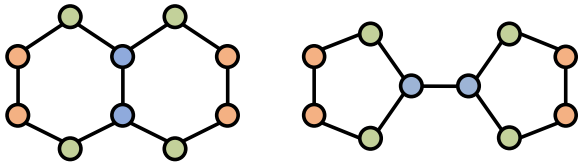


Figure 4: 1-WL test fails to distinguish the decalin graph (left) and the bicyclopentyl graph (right).

addition,G-Jaccard [113] and GNNGuard [144] prune links whose head and tail nodes' feature similarity (or embedding similarity) is lower than a predefined threshold to eliminate potentially malicious edges. Alternatively, the graph prior knowledge can be realized by setting explicit regularization terms. Pro-GNN [53] includes the topology sparsity and feature smoothness regularization terms into the optimization objective which can guide the ***graph rewiring***. Besides, the supervision signals from downstream tasks can implicitly reflect the uncontaminated topology and feature distribution. For instance, Gasoline [125] calibrates the given graph based on the evaluation performance (e.g., classification loss) of the validation nodes. GIB-N [118] adopts the information bottleneck objective [99] which maximizes the mutual information between node labels and node embeddings and uses this objective function to guide the ***graph rewiring***.

It is worth noting that an established defense strategy named adversarial training [34] is grafted onto the graph data. Examples include a family of graph (virtual) adversarial training [63; 27; 19; 107]. Their core idea is to adversarially conduct ***edge perturbation*** and ***feature corruption*** to generate various challenging graph data samples which maximize the classification loss. Then those samples are included in the training of downstream GNN models which can improve the robustness of GNNs against adversarial attacks.

Boosting GNN Expressive Power. Studies about the expressiveness of GNNs show that various mainstream GNNs cannot be more powerful than the 1-Weisfeiler-Lehman (WL) isomorphism test to distinguish graphs and subgraphs [80; 124]. E.g., Figure 4 shows a pair of common examples that cannot be distinguished by the 1-WL test due to the common rooted trees. In addition, due to the power-law distribution of node degree, many nodes with few neighbors are hard to be represented properly. To break such an inherent limitation of GNNs, augmenting the given graphs is a feasible solution. One line of research adopts the ***feature addition*** technique to augment the original node/edge features. For example, Distance Encoding [66] augments the given node features with distance measures between node pairs (or aggregated distance measures) to yield stronger expressiveness than 1-WL test-based GNNs. A recent study from Sato et al. [89] reveals that adding random features into the existing node features can boost the expressiveness of off-the-shelf GNNs (e.g., discriminate triangle structure). GSN [7] and a variant of ID-GNN [130] take a step further and augment the node features with the count of various motifs (e.g., cycles). Recently, LAGNN [70] adopts ***feature addition*** by a trained feature generator from which nodes with few neighbors can benefit greatly.

Another line of work argues that the limitation of the 1-WL test (and 1-WL test-based GNNs) is related to the structure of rooted trees. Based on that, ***graph sampling*** technique can enhance the GNN expressiveness. Specifically, NGNN [142] claims that node representations from sampled rooted subgraphs are more expressive than those from rooted trees. Interestingly, ID-GNN [130] shares a similar insight with NGNN [142] and also samples an ego network for every node for computing the node embedding.

As the node representations are obtained through the aggregation of node features within the receptive fields, to enhance the propagation of messages for long distance, a straightforward way is to apply ***node insertion*** by inserting virtual nodes (i.e., super nodes or master nodes) [33; 65; 48; 84] into the graphs which are connected with all the existing nodes.

Alleviating Over-Smoothing/Squashing of GNNs. Due to the inherent design limit of GNNs, as the depth of the model increases, the representations of different nodes in the graph eventually become indistinguishable with iterative message passing. The so-called over-smoothing phenomenon generally leads to failure in the graph learning tasks [67]. To counter the over-smoothing issue, GraphDA, especially ***graph rewiring*** methods have been shown as an effective solution. For example, DropEdge [86] randomly removes graph edges during message passing to alleviate over-smoothing. TADropEdge [32] exploits graph structural information to compute edge weights for edge dropping, such that the augmented subgraphs can avoid the arbitrary data augmentation issue in DropEdge. AdaEdge [9] iteratively adds or removes edges to the graph topology based on the classification results (from a GNNs-based classifier) and trains GNN classifiers on the updated graphs to overcome the over-smoothing issue. SHADOW-GNN [135] samples informative subgraphs centered on each node and then builds a deep GNN operating on subgraphs instead of the whole graph to decouple the depth and scope of GNNs.

Note that over-smoothing is mostly demonstrated in tasks that depend mostly on short-range information, while GNNs are also ineffective for capturing long-range node interactions. A recent study [1] points out that the distortion of information flowing from distant nodes (i.e., over-squashing) is the main factor limiting the efficiency of message passing for tasks relying on long-distance interactions. As the layer depth increases, the number of nodes in each node's receptive field grows exponentially and leads to the over-squashing issue. To overcome over-squashing, ***graph diffusion*** and ***graph rewiring*** are two effective choices. Graph Diffusion Convolution (GDC) [61] constructs a new graph based on a generalized form of graph diffusion, which can be further used to enlarge a larger neighborhood for message-passing. Adaptive Diffusion Convolution (ADC) [146] supports learning the optimal neighborhood from the data automatically to eliminate the manual search process of the optimal propagation neighborhood in GDC. Alon & Yahav [1] propose to rewire the graph to build a fully-adjacent GNN layer for reducing the bottleneck. Similarly, Stochastic Discrete Ricci Flow (SDRF) [100] is a curvature-based method for graph rewiring to mitigate the over-squashing issue.

Scalable Graph Training. Scalability is always a crucial challenge for GNNs which hinders the applicability of a broad class of GNN models over large real-world graphs. GraphDA techniques, such as ***graph sampling***, ***graph diffusion***, and ***graph generation*** play an important role to speed up the training and inferring of GNNs.

Based on the idea of ***graph sampling***, GraphSAGE [36]

uniformly samples the neighbors of the target nodes to enable GNN inductive learning on large graphs. FastGCN [11] shares a similar idea with GraphSAGE but follows the idea of importance sampling to sample vertices in every layer. To overcome the redundant sampling and sparse connection problem brought by the GraphSAGE [36] and FastGCN [11] respectively, Zou et al. [154] propose LADIES which is a layer-dependent sampling strategy. In addition, GraphSAINT [136] is another sampling-based method that applies various sampling methods (e.g., based on random walk) to improve the scalability of training rather than modifying the vanilla GCN model [59]. Similar to sampling methods, graph partition is also effective to lower the memory requirement. For example, Cluster-GCN [17] partitions the given graph into clusters and conducts convolution operation within every cluster to avoid heavy neighborhood search and sampling.

Another line of GraphDA research on scaling the training of GNNs adopts a decoupled design that trains MLP on the augmented graphs via ***graph diffusion*** [12; 112; 60; 87]. Given the above model design, the graph diffusion matrix can be pre-computed, which makes the training of MLP more efficient. A set of representative works adopt PageRank [81] and its variant (e.g., personalized PageRank [49]) to infer the diffusion matrix such as SGC [112] and APPNP [60]. Similarly, PPRGo [6] adopts the Forward Push algorithm [2] to approximate the personalized PageRank matrix for efficiency. GBP [14] revisits the core idea of PPRGo and further speeds up the computation of the generalized PageRank matrix by the Bidirectional Propagation algorithm. In addition, a very recent work named GRAND+ [29] also approximates the graph diffusion matrix to scale the training of graph random neural networks [30] in the consistency training context.

In addition, works on "graph condensation" and "graph coarsening" try to shrink the given graph $\mathcal{G}$ by ***graph generation*** so that the generated graph $\tilde{\mathcal{G}}$ can be handled by GNNs [46]. Their core idea is to minimize a 'quantity of interest' between the input graph $\mathcal{G}$ and augmented graph $\tilde{\mathcal{G}}$. For example, Jin et al. [56] propose to minimize the spectral distance between $\mathcal{G}$ and $\tilde{\mathcal{G}}$. GOREN [8] aims to keep Laplace operators from $\mathcal{G}$ and $\tilde{\mathcal{G}}$ comparable. GCOND [55] and DosCond [54] minimize the difference of training gradients on $\mathcal{G}$ and $\tilde{\mathcal{G}}$.

6. FUTURE DIRECTIONS

GraphDA is an emerging and fast-developing field. Although substantial progress has been achieved, many challenges remain under-explored. In this section, we discuss some promising research directions as follows.

Data Augmentation beyond Simple Graphs. Most of the aforementioned works develop augmentation strategies on homophilic (i.e., assortative) graphs where edges tend to connect nodes with the same properties (e.g., labels, features). However, heterophily (i.e., disassortativity) also exists commonly in networks such as heterosexual dating networks. Many existing augmentation approaches [94] on heterophilic graphs focus on improving the assortativity of the given graphs or dropping/deweighting the existing disassortative edges [129]. The augmentation of node features, labels, and non-existing edges of heterophilic graphs remains understudied. Besides, existing GraphDA efforts are mainly developed for either plain or attributed graphs, while principled augmentation approaches for other types of graphs (e.g., heterogeneous graphs, hypergraphs, multiplex graphs, dynamic graphs) remain largely unexplored. Those complex graphs provide broader design space for augmentation but also challenge the effectiveness of existing GraphDA methods greatly, which is vital to explore in the future.

Automated and Generalizable Graph Data Augmentation. In general, the effectiveness of DGL problems hinges on adhoc graph data augmentations, which have to be manually picked per dataset, by either rules of thumb or trial-and-errors. For example, researchers observe that different data augmentations affect downstream tasks differently across datasets, which suggests that searching over augmentation functions is crucial for graph self-supervised learning [131]. Nevertheless, evaluating representations derived from multiple augmentation functions without direct access to ground truth labels makes this problem challenging. Hence, it is necessary to develop automated data augmentation solutions to adaptively customize augmentation strategies for each graph dataset [77]. Meanwhile, considering that different graphs usually have distinct graph properties, developing generalizable data augmentation methods without learning from scratch for each domain is also a promising direction to improve the practical usage of GraphDA methods.

Semantic-Preserving Graph Data Augmentation. Designing effective data augmentation for graphs is challenging due to their non-Euclidean nature and the dependencies between data samples. Most graph data augmentation methods adopt arbitrary augmentations on the input graph, which may unexpectedly change both structural and semantic patterns of the graph [82]. For example, dropping a carbon atom from the phenyl ring of aspirin breaks the aromatic system and results in a alkene chain, which is an entirely different chemical compound. Hence, proposing a label-consistent/semantic-preserving GraphDA method is of necessity. To this end, recent studies [64; 121; 134] perform data augmentation on the latent space to avoid the perturbation on the semantics.

Graph Data Augmentation for Trustworthy DGL. Despite the success of DGL, how to ensure various DGL algorithms behave in a socially responsible manner and meet regulatory compliance requirements becomes an emerging problem, especially in risk-sensitive applications. In fact, GraphDA can be an effective tool to achieve Trustworthy GML, especially on fairness, causality, explainability of DGL algorithms. For example, counterfactural graph data augmentation [75; 148] has been used to explain the behavior of GNNs in the literature. Moreover, data augmentation itself is typically performed in an adhoc manner with little understanding of the underlying theoretical principles. Existing work on GraphDA is mainly surface-level, and rarely investigates the theoretical underpinnings and principles. Overall, there indeed appears to be a lack of research on interpret why exactly GraphDA works.

7. CONCLUSION

In this paper, we present a forward-looking and structured survey of graph data augmentation (GraphDA). To inspect the nature of GraphDA, we give a formal formulation and a taxonomy to facilitate the understanding of this emerging research problem. Specifically, we frame GraphDA meth-

ods into three categories according to the target augmentation modalities, i.e., feature-wise, structure-wise, and label-wise augmentations. We further review the application of GraphDA methods to address two data-centric DGL problems (i.e., low-resource graph learning and reliable graph learning) and discuss the prevailing GraphDA-based algorithms. Finally, we outline current challenges as well as opportunities for future research in this field.

Acknowledgments

This work is supported by NSF (No. 1947135, 2106825, 2229461, and 2134079), the NSF Program on Fairness in AI in collaboration with Amazon (No. 1939725), DARPA (No. HR001121C0165), ARO (No. W911NF2110088), ONR (No. N00014-21-1-4002), NIFA (No. 2020-67021-32799, and W911NF2110030), and ARL (No. W911NF2020124).

8. REFERENCES

[1] U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, 2006.

[3] C. M. Bishop. Neural networks and their applications. *Review of scientific instruments*, 1994.

[4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, 1998.

[5] D. Bo, B. Hu, X. Wang, Z. Zhang, C. Shi, and J. Zhou. Regularizing graph neural networks via consistency-diversity graph augmentations. In *AAAI*, 2022.

[6] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann. Scaling graph neural networks with approximate pagerank. In *KDD*, 2020.

[7] G. Bouritsas, F. Frasca, S. P. Zafeiriou, and M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *TPAMI*, 2022.

[8] C. Cai, D. Wang, and Y. Wang. Graph coarsening with neural networks. In *ICLR*, 2021.

[9] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, 2020.

[10] H. Chen, S. Zhang, and G. Xu. Graph masked autoencoder. *arXiv preprint arXiv:2202.08391*, 2022.

[11] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[12] L. Chen, Z. Chen, and J. Bruna. On graph neural networks versus graph-augmented mlps. In *ICLR*, 2020.

[13] L. Chen, Y. Yao, F. Xu, M. Xu, and H. Tong. Trading personalization for accuracy: Data debugging in collaborative filtering. *NeurIPS*, 2020.

[14] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J.-R. Wen. Scalable graph neural networks via bidirectional propagation. *NeurIPS*, 2020.

[15] X. Chen, S. Chen, J. Yao, H. Zheng, Y. Zhang, and I. W. Tsang. Learning on attribute-missing graphs. *TPAMI*, 2020.

[16] Y. Chen, L. Wu, and M. Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *NeurIPS*, 2020.

[17] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

[18] E. Dai, C. Aggarwal, and S. Wang. Nrgnn: Learning a label noise-resistant graph neural network on sparsely and noisily labeled graphs. In *KDD*, 2021.

[19] Z. Deng, Y. Dong, and J. Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv preprint arXiv:1902.09192*, 2019.

[20] K. Ding, J. Li, R. Bhanushali, and H. Liu. Deep anomaly detection on attributed networks. In *SDM*, 2019.

[21] K. Ding, J. Wang, J. Caverlee, and H. Liu. Meta propagation networks for graph few-shot semi-supervised learning. In *AAAI*, 2022.

[22] K. Ding, J. Wang, J. Li, D. Li, and H. Liu. Be more with less: Hypergraph attention networks for inductive text classification. In *EMNLP*, 2020.

[23] K. Ding, J. Wang, J. Li, K. Shu, C. Liu, and H. Liu. Graph prototypical networks for few-shot learning on attributed networks. In *CIKM*, 2020.

[24] K. Ding, Y. Wang, Y. Yang, and H. Liu. Eliciting structural and semantic global knowledge in unsupervised graph contrastive learning. In *AAAI*, 2023.

[25] H. Dong, J. Chen, F. Feng, X. He, S. Bi, Z. Ding, and P. Cui. On the equivalence of decoupled graph convolution network and label propagation. In *TheWebConf*, 2021.

[26] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *WSDM*, 2020.

[27] F. Feng, X. He, J. Tang, and T.-S. Chua. Graph adversarial training: Dynamically regularizing based on graph structure. *TKDE*, 2019.

[28] S. Feng, B. Jing, Y. Zhu, and H. Tong. Adversarial graph contrastive learning with information regularization. In *TheWebConf*, 2022.

[29] W. Feng, Y. Dong, T. Huang, Z. Yin, X. Cheng, E. Kharlamov, and J. Tang. Grand+: Scalable graph random neural networks. In *TheWebConf*, 2022.

[30] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph random neural networks for semi-supervised learning on graphs. In *NeurIPS*, 2020.

[31] L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In *ICML*, 2019.

[32] Z. Gao, S. Bhattacharya, L. Zhang, R. S. Blum, A. Ribeiro, and B. M. Sadler. Training robust graph neural networks with topology adaptive edge dropping. *arXiv preprint arXiv:2106.02892*, 2021.

[33] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[34] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[35] H. Guo and Y. Mao. ifmixup: Towards intrusion-free graph mixup for graph classification. *arXiv preprint arXiv:2110.09344*, 2021.

[36] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[37] X. Han, Z. Jiang, N. Liu, and X. Hu. G-mixup: Graph data augmentation for graph classification. In *ICML*, 2022.

[38] K. Hassani and A. H. Khasahmadi. Contrastive multi-view representation learning on graphs. In *ICML*, 2020.

[39] K. Hassani and A. H. Khasahmadi. Learning graph augmentations to learn graph representations. *arXiv preprint arXiv:2201.09830*, 2022.

[40] C. Hawkins, V. N. Ioannidis, S. Adeshina, and G. Karypis. Scalable consistency training for graph neural networks via self-ensemble self-distillation. *arXiv preprint arXiv:2110.06290*, 2021.

[41] D. He, C. Liang, C. Huo, Z. Feng, D. Jin, L. Yang, and W. Zhang. Analyzing heterogeneous networks with missing attributes by unsupervised contrastive learning. *TNNLS*, 2022.

[42] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*, 2016.

[43] Z. Hou, X. Liu, Y. Dong, C. Wang, J. Tang, et al. Graphmae: Self-supervised masked graph autoencoders. In *KDD*, 2022.

[44] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *KDD*, 2020.

[45] Z. Hu, C. Fan, T. Chen, K.-W. Chang, and Y. Sun. Pre-training graph neural networks for generic structural feature extraction. *arXiv preprint arXiv:1905.13728*, 2019.

[46] Z. Huang, S. Zhang, C. Xi, T. Liu, and M. Zhou. Scaling up graph neural networks via graph coarsening. In *KDD*, 2021.

[47] B. Hui, P. Zhu, and Q. Hu. Collaborative graph convolutional networks: Unsupervised learning meets semi-supervised learning. In *AAAI*, 2020.

[48] K. Ishiguro, S.-i. Maeda, and M. Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. *arXiv preprint arXiv:1902.01020*, 2019.

[49] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[50] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu. Sub-graph contrast for scalable self-supervised graph representation learning. In *ICDM*, 2020.

[51] D. Jin, C. Huo, C. Liang, and L. Yang. Heterogeneous graph neural network via attribute completion. In *TheWebConf*, 2021.

[52] M. Jin, Y. Zheng, Y.-F. Li, C. Gong, C. Zhou, and S. Pan. Multi-scale contrastive siamese networks for self-supervised graph representation learning. In *IJCAI*, 2021.

[53] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. In *KDD*, 2020.

[54] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin. Condensing graphs via one-step gradient matching. In *KDD*, 2022.

[55] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah. Graph condensation for graph neural networks. In *ICLR*, 2022.

[56] Y. Jin, A. Loukas, and J. JaJa. Graph coarsening with preserved spectral properties. In *AISTATS*, 2020.

[57] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. Bronstein. Differentiable graph module (dgm) for graph convolutional networks. *TPAMI*, 2022.

[58] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[59] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[60] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2018.

[61] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

[62] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002.

[63] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

[64] N. Lee, J. Lee, and C. Park. Augmentation-free self-supervised learning on graphs. In *AAAI*, 2022.

[65] J. Li, D. Cai, and X. He. Learning graph-level representation for drug discovery. *arXiv preprint arXiv:1709.03741*, 2017.

[66] P. Li, Y. Wang, H. Wang, and J. Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *NeurIPS*, 2020.

[67] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[68] S. Li, W.-T. Li, and W. Wang. Co-gcn for multi-view semi-supervised learning. In *AAAI*, 2020.

[69] X. Li, L. Wen, Y. Deng, F. Feng, X. Hu, L. Wang, and Z. Fan. Graph neural network with curriculum learning for imbalanced node classification. *arXiv preprint arXiv:2202.02529*, 2022.

[70] S. Liu, H. Dong, L. Li, T. Xu, Y. Rong, P. Zhao, J. Huang, and D. Wu. Local augmentation for graph neural networks. *ICML*, 2022.

[71] X. Liu, J. Ding, W. Jin, H. Xu, Y. Ma, Z. Liu, and J. Tang. Graph neural networks with adaptive residual. *NeurIPS*, 2021.

[72] Y. Liu, K. Ding, J. Wang, V. Lee, H. Liu, and S. Pan. Learning strong graph neural networks with weak information. In *KDD*, 2023.

[73] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. Yu. Graph self-supervised learning: A survey. *TKDE*, 2022.

[74] Y. Liu, Z. Zhang, Y. Liu, and Y. Zhu. Gatsmote: Improving imbalanced node classification on graphs via attention and homophily. *Mathematics*, 2022.

[75] A. Lucic, M. A. Ter Hoeve, G. Tolomei, M. De Rijke, and F. Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *AISTATS*, 2022.

[76] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang. Learning to drop: Robust graph neural network via topological denoising. In *WSDM*, 2021.

[77] Y. Luo, M. McThrow, W. Y. Au, T. Komikado, K. Uchino, K. Maruhash, and S. Ji. Automated data augmentations for graph classification. *arXiv preprint arXiv:2202.13248*, 2022.

[78] O. Mahmood, E. Mansimov, R. Bonneau, and K. Cho. Masked graph modeling for molecule generation. *Nature communications*, 2021.

[79] Y. Mo, L. Peng, J. Xu, X. Shi, and X. Zhu. Simple unsupervised graph representation learning. In *AAAI*, 2022.

[80] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[81] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[82] H. Park, S. Lee, S. Kim, J. Park, J. Jeong, K.-M. Kim, J.-W. Ha, and H. J. Kim. Metropolis-hastings data augmentation for graph neural networks. *NeurIPS*, 2021.

[83] J. Park, H. Shim, and E. Yang. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *AAAI*, 2022.

[84] T. Pham, T. Tran, H. Dam, and S. Venkatesh. Graph classification via deep learning with virtual nodes. *arXiv preprint arXiv:1708.04357*, 2017.

[85] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *KDD*, 2020.

[86] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2019.

[87] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[88] E. Rossi, H. Kenlay, M. I. Gorinova, B. P. Chamberlain, X. Dong, and M. Bronstein. On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features. *arXiv preprint arXiv:2111.12128*, 2021.

[89] R. Sato, M. Yamada, and H. Kashima. Random features strengthen graph neural networks. In *SDM*, 2021.

[90] X. Shen, D. Sun, S. Pan, X. Zhou, and L. T. Yang. Neighbor contrastive learning on learnable graph augmentation. In *AAAI*, 2023.

[91] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 2019.

[92] K. Sun, Z. Lin, and Z. Zhu. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *AAAI*, 2020.

[93] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, L. He, and B. Li. Adversarial attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*, 2018.

[94] S. Suresh, V. Budde, J. Neville, P. Li, and J. Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *KDD*, 2021.

[95] S. Suresh, P. Li, C. Hao, and J. Neville. Adversarial graph augmentation to improve graph contrastive learning. In *NeurIPS*, 2021.

[96] H. Taguchi, X. Liu, and T. Murata. Graph convolutional networks for graphs containing missing features. *FGCS*, 2021.

[97] Q. Tan, N. Liu, X. Huang, R. Chen, S.-H. Choi, and X. Hu. Mgae: Masked autoencoders for self-supervised learning on graphs. *arXiv preprint arXiv:2201.02534*, 2022.

[98] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko. Large-scale representation learning on graphs via bootstrapping. In *ICLR*, 2021.

[99] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[100] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

[101] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. *ICLR*, 2019.

[102] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, 2019.

[103] V. Verma, M. Qu, K. Kawaguchi, A. Lamb, Y. Bengio, J. Kannala, and J. Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *AAAI*, 2021.

[104] J. Wang, K. Ding, L. Hong, H. Liu, and J. Caverlee. Next-item recommendation with sequential hypergraphs. In *SIGIR*, 2020.

[105] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie. Graph structure estimation neural networks. In *TheWebConf*, 2021.

[106] X. Wang and J. Chen. A dual-branch graph convolutional network on imbalanced node classification. In *CSAE*, 2021.

[107] X. Wang, X. Liu, and C.-J. Hsieh. Graphdefense: Towards robust graph convolutional networks. *arXiv preprint arXiv:1911.04429*, 2019.

[108] Y. Wang, C. Aggarwal, and T. Derr. Distance-wise prototypical graph neural network in node imbalance classification. *arXiv preprint arXiv:2110.12035*, 2021.

[109] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Mixup for node and graph classification. In *TheWebConf*, 2021.

[110] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi. Nodeaug: Semi-supervised node classification with data augmentation. In *KDD*, 2020.

[111] X. Wei, Y. Li, X. Qin, X. Xu, X. Li, and M. Liu. From decoupling to reconstruction: A robust graph neural network against topology attacks. In *WCSP*, 2020.

[112] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[113] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu. Adversarial examples for graph data: deep insights into attack and defense. In *IJCAI*, 2019.

[114] L. Wu, H. Lin, Y. Huang, and S. Z. Li. Knowledge distillation improves graph structure augmentation for graph neural networks. In *NeurIPS*, 2022.

[115] L. Wu, H. Lin, Z. Liu, Z. Liu, Y. Huang, and S. Z. Li. Homophily-enhanced self-supervision for graph structure learning: Insights and directions. *TNNLS*, 2023.

[116] L. Wu, H. Lin, C. Tan, Z. Gao, and S. Z. Li. Self-supervised learning on graphs: Contrastive, generative, or predictive. *TKDE*, 2021.

[117] L. Wu, J. Xia, Z. Gao, H. Lin, C. Tan, and S. Z. Li. Graphmixup: Improving class-imbalanced node classification on graphs by self-supervised context prediction. In *ECML-PKDD*, 2022.

[118] T. Wu, H. Ren, P. Li, and J. Leskovec. Graph information bottleneck. In *NeurIPS*, 2020.

[119] X. Wu, L. Zhao, and L. Akoglu. A quest for structure: jointly learning the graph structure and semi-supervised classification. In *CIKM*, 2018.

[120] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *TNNLS*, 2020.

[121] J. Xia, L. Wu, J. Chen, B. Hu, and S. Z. Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *TheWebConf*, 2022.

[122] Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le. Unsupervised data augmentation for consistency training. In *NeurIPS*, 2020.

[123] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji. Self-supervised learning of graph neural networks: A unified review. *TPAMI*, 2022.

[124] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2018.

[125] Z. Xu, B. Du, and H. Tong. Graph sanitation with application to node classification. *arXiv preprint arXiv:2105.09384*, 2021.

[126] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo. Topology optimization based graph convolutional network. In *IJCAI*, 2019.

[127] L. Yang, L. Zhang, and W. Yang. Graph adversarial self-supervised learning. In *NeurIPS*, 2021.

[128] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *ACL*, 1995.

[129] Y. Ye and S. Ji. Sparse graph attention networks. *TKDE*, 2021.

[130] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec. Identity-aware graph neural networks. In *AAAI*, 2021.

[131] Y. You, T. Chen, Y. Shen, and Z. Wang. Graph contrastive learning automated. In *ICML*, 2021.

[132] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[133] Y. You, T. Chen, Z. Wang, and Y. Shen. When does self-supervision help graph convolutional networks? In *ICML*, 2020.

[134] H. Yue, C. Zhang, C. Zhang, and H. Liu. Label-invariant augmentation for semi-supervised graph classification. In *NeurIPS*, 2022.

[135] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen. Decoupling the depth and scope of graph neural networks. *NeurIPS*, 2021.

[136] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2019.

[137] J. Zeng and P. Xie. Contrastive self-supervised learning for graph classification. In *AAAI*, 2021.

[138] A. Zhang and J. Ma. Defensevgae: Defending against adversarial attacks on graph data via a variational graph autoencoder. *arXiv preprint arXiv:2006.08900*, 2020.

[139] C. Zhang, Y. He, Y. Cen, Z. Hou, and J. Tang. Improving the training of graph neural networks with consistency regularization. *arXiv preprint arXiv:2112.04319*, 2021.

[140] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[141] J. Zhang, H. Zhang, C. Xia, and L. Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[142] M. Zhang and P. Li. Nested graph neural networks. *NeurIPS*, 2021.

[143] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 2019.

[144] X. Zhang and M. Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *NeurIPS*, 2020.

[145] Y. Zhang, S. Pal, M. Coates, and D. Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In *AAAI*, 2019.

[146] J. Zhao, Y. Dong, M. Ding, E. Kharlamov, and J. Tang. Adaptive diffusion in graph neural networks. In *NeurIPS*, 2021.

[147] J. Zhao, X. Wang, C. Shi, B. Hu, G. Song, and Y. Ye. Heterogeneous graph structure learning for graph neural networks. In *AAAI*, 2021.

[148] T. Zhao, G. Liu, D. Wang, W. Yu, and M. Jiang. Learning from counterfactual links for link prediction. In *ICML*, 2022.

[149] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah. Data augmentation for graph neural networks. In *AAAI*, 2021.

[150] T. Zhao, X. Zhang, and S. Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *WSDM*, 2021.

[151] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang. Robust graph representation learning via neural sparsification. In *ICML*, 2020.

[152] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

[153] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Graph contrastive learning with adaptive augmentation. In *TheWebConf*, 2021.

[154] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *NeurIPS*, 2019.