Data-Augmented Curriculum Graph Neural Architecture Search under Distribution Shifts

Yang Yao[1], **Xin Wang**[1,2*], **Yijian Qin**[1], **Ziwei Zhang**[1], **Wenwu Zhu**[1,2*], **Hong Mei**[3]

[1]Department of Computer Science and Technology, Tsinghua University
[2]Beijing National Research Center for Information Science and Technology, Tsinghua University
[3]MoE Key Lab of High Confidence Software Technologies, Peking University
{yaoyang21,qinyj19}@mails.tsinghua.edu.cn, {xin_wang,zwzhang,wwzhu}@tsinghua.edu.cn, meih@pku.edu.cn

Abstract

Graph neural architecture search (NAS) has achieved great success in designing architectures for graph data processing. However, distribution shifts pose great challenges for graph NAS, since the optimal searched architectures for the training graph data may fail to generalize to the unseen test graph data. The sole prior work tackles this problem by customizing architectures for each graph instance through learning graph structural information, but fails to consider data augmentation during training, which has been proven by existing works to be able to improve generalization. In this paper, we propose Data-augmented Curriculum Graph Neural Architecture Search (DCGAS), which learns an architecture customizer with good generalizability to data under distribution shifts. Specifically, we design an *embedding-guided data generator*, which can generate sufficient graphs for training to help the model better capture graph structural information. In addition, we design a *two-factor uncertainty-based curriculum weighting strategy*, which can evaluate the importance of data in enabling the model to learn key information in real-world distribution and reweight them during training. Experimental results on synthetic datasets and real datasets with distribution shifts demonstrate that our proposed method learns generalizable mappings and outperforms existing methods.

Introduction

Graph neural networks have achieved great success in processing non-Euclidean data such as social networks, traffic networks, molecular structures, etc. They use the information of adjacent nodes to update the features of nodes, and can capture the structure of graph data. In recent years, many excellent graph neural networks, such as GCN (Kipf and Welling 2017), GAT (Velickovic et al. 2018), and GIN (Xu et al. 2019), have been designed for various datasets and tasks. However, manually designing network architectures requires expert knowledge and significant amounts of effort.

With the rise of neural architecture search (NAS), graph neural architecture search methods have been used to customize graph neural network architectures for different datasets and tasks. Compared to manually designed networks, graph NAS methods (Gao et al. 2020a; Zhou et al. 2022a) have achieved better results. Existing graph NAS works (Wei et al. 2021; Gao et al. 2020b) mainly focused on searching for one architecture for a given dataset. This strategy works well for independently and identically distributed (I.I.D.) datasets, where the training data and testing data come from the same distribution. However, in graph classification datasets with distribution shifts between training and testing, the architectures found using the training dataset may generalize poorly to data under a different distribution. GRACES (Qin et al. 2022) is the sole work to consider distribution shifts and out-of-distribution generalization in graph NAS. They attempt to produce customized architectures by learning the feature of different graphs through decoupling graph encoding, and then using the feature information to customize the architecture for each graph. By producing different architectures for each graph, it is possible to achieve great performance in tasks with distribution shifts.

However, customizing the architecture for datasets with distribution shifts still faces several important challenges. The existing graph NAS methods designed for datasets with distribution shifts failed to consider data augmentation during training, which has been proven by existing works (Zhao et al. 2021; Wu et al. 2022a; You et al. 2020) to be able to improve generalization. Merely learning from the graphs present in the training dataset often fails to effectively capture the crucial graph structural information, thus impacting the performance of customized architectures in existing works. Additionally, since distribution shifts exist between the training dataset and the test distribution, the importance of different graphs in enabling the model to learn key information in real-world distribution can vary significantly. Existing methods have treated all data equally without considering their importance, which leads to inferior performance.

In order to solve these challenges, we propose a Data-augmented Curriculum Graph Neural Architecture Search (DCGAS) approach to produce customized architectures from graph data. To enable better generalization ability with limited training data, we design an *embedding-guided data generator* to generate new data for training. It adds an embedding guidance module to a discrete graph diffusion model, allowing for the generation of graphs with similar structures to a given graph. Moreover, to better facilitate important graphs in the training dataset for further enhancing OOD generalization ability, we use a *two-factor uncertainty-based curriculum weighting strategy* to schedule the training

*Corresponding authors.

data. Specifically, we design a two-factor uncertainty-based curriculum weighting strategy that measures the uncertainty of the architecture customizer's performance on data. Higher uncertainties indicate data that should be used more during training, and therefore are assigned higher weights. Experimental results on synthetic datasets and real datasets with distribution shifts demonstrate that our proposed method is able to learn generalizable architecture customizers that outperform existing methods.

Our contributions are summarized as follows:

- We propose a graph NAS method, DCGAS for graph classification, which learns the mapping from graph data to architectures and is able to customize architectures for graph data under distribution shifts.
- We design a data generator with embedding-guided discrete diffusion, which can generate new graph data with similar structures to a given graph for training.
- We design a data scheduler with two-factor uncertainty-based curriculum weighting, which can evaluate the importance of data and reweight them during training.

Preliminaries

Graph Neural Architecture Search

We denote the training dataset by $\mathcal{D}_{\text{train}} = (\mathcal{G}_{\text{train}}, \mathcal{Y}_{\text{train}})$, where $\mathcal{G}_{\text{train}} = \{G_i\}$ is the set of training graphs and $\mathcal{Y}_{\text{train}} = \{y_i\}$ is their corresponding labels. Similarly, let $\mathcal{D}_{\text{test}} = (\mathcal{G}_{\text{test}}, \mathcal{Y}_{\text{test}})$ be the testing dataset. Graph neural architecture search methods aim to choose architectures from a search space $\mathcal{A}$ such that the model after training optimizes the loss on the testing set:

$$\begin{aligned} A^* &= \operatorname{argmin}_{A\in\mathcal{A}} \mathcal{L}(A, \theta^*, \mathcal{D}_{\text{test}}) \\ s.t.\ \theta^* &= \operatorname{argmin}_{\theta} \mathcal{L}(A, \theta, \mathcal{D}_{\text{train}}) \end{aligned}, \tag{1}$$

where $\mathcal{L}$ is the loss function and θ is the network parameters.

Graph neural architecture search learns to produce an architecture that is optimal for the whole dataset. However, for graph classification tasks, since the training dataset and the testing data often have different distributions of graphs, the optimal architectures can vary for different graphs. Therefore, it can be advantageous to allow using different architectures based on the graph. GRACES (Qin et al. 2022) first studied the problem of customizing architectures based on input data, and achieved superior results on several graph classification datasets with distribution shifts. They generalized graph neural architecture search to learn a mapping from graphs to architectures, producing optimal architectures for each graph.

Graph Diffusion Model

Diffusion models are a class of generative models that recently grew popular due to their excellent performance in computer vision. Recently, there are works that successfully apply diffusion models to the problem of graph generation. Due to the discrete nature of graph data, diffusion models that build on discrete probabilistic distributions (Vignac et al. 2023) have better performance than traditional diffusion models based on Gaussian distribution. We briefly describe the framework of discrete graph diffusion as follows.

A discrete graph diffusion model consists of a forward process and a reverse process. Let $G = (X, E)$ be a graph, where $X = \{x_i\}$ is the node features and $E = \{e_{i,j}\}$ is the matrix of edge features, and the absence of edges is represented with the special value $e_{i,j} = 0$. The forward process is a Markov chain that starts with $G^0 = G$, gradually adds noise and ultimately transforms it into pure noise $G^T = (X^T, E^T)$. In the case of discrete diffusion, each step of the Markov chain $q(G^t \mid G^{t-1})$ transforms each element of X^{t-1} and E^{t-1} independently:

$$\begin{aligned} q(x^t = j \mid x^{t-1} = i) &= Q^t_{X,ij} \\ q(e^t = j \mid e^{t-1} = i) &= Q^t_{E,ij} \end{aligned}, \tag{2}$$

where Q^t_X and Q^t_E are predefined transition matrices. Given the initial value G^0, we can sample G^T with a closed-form expression, using the one-hot encoding for X_0 and E_0 as follows:

$$q(X^t \mid X^0) = X^0 \bar{Q}^t_X, \quad q(E^T \mid E^0) = E^0 \bar{Q}^t_E \tag{3}$$

$$\bar{Q}^t_X = Q^1_X \dots Q^t_X, \quad \bar{Q}^t_E = Q^1_E \dots Q^t_E. \tag{4}$$

The reverse process of the diffusion model is defined with a denoising neural network $p_\theta(G^0 \mid G^t)$ that learns to remove the added noise from G^t. By expressing $p(G^{t-1} \mid G^t)$ as $\sum_{G^0} p_\theta(G^0 \mid G^t) q(G^{t-1} \mid G^0, G^t)$, it can be used for gradually denoising the graph and generating new data. The denoising network is trained with the following loss:

$$\begin{aligned} \mathcal{L} = \mathbb{E}_{G\sim D_{\text{train}}, t\sim U(0,T), G^t\sim q(G^t\mid G^0)} \\ \left[\lambda_X \text{CE}(X^0, p_\theta(X^0 \mid G^t)) + \lambda_E \text{CE}(E^0, p_\theta(E^0 \mid G^t))\right], \end{aligned} \tag{5}$$

where $\text{CE}(\cdot)$ is the cross-entropy loss, and λ_X and λ_E are hyperparameters controlling the weighting of terms.

Data-Augmented Curriculum Graph Neural Architecture Search

In this section, we introduce our method. The framework of our method is shown in Figure 1. We describe the embedding-guided data generator and two-factor uncertainty-based curriculum weighting strategy in the following subsections.

Framework

Our method aims to learn the mapping of graph data to architecture, which can customize a good architecture for data with distribution shifts. It can be formalized as follows:

$$A^*(G) = \operatorname{argmin}_{A\in\mathcal{A}} \mathcal{L}(A, \theta^*, G), \tag{6}$$

where $\mathcal{A}$ is the space of available architectures, $A^*(G)$ is the learned mapping for graph data G, and θ^* represents the trained model parameters.

The framework of our method is shown in Figure 1. For a given graph data, it firstly computes the graph embedding using a graph encoder, and then produces the customized architecture from the graph embedding using an architecture customizer. To address the problem of limited training data,

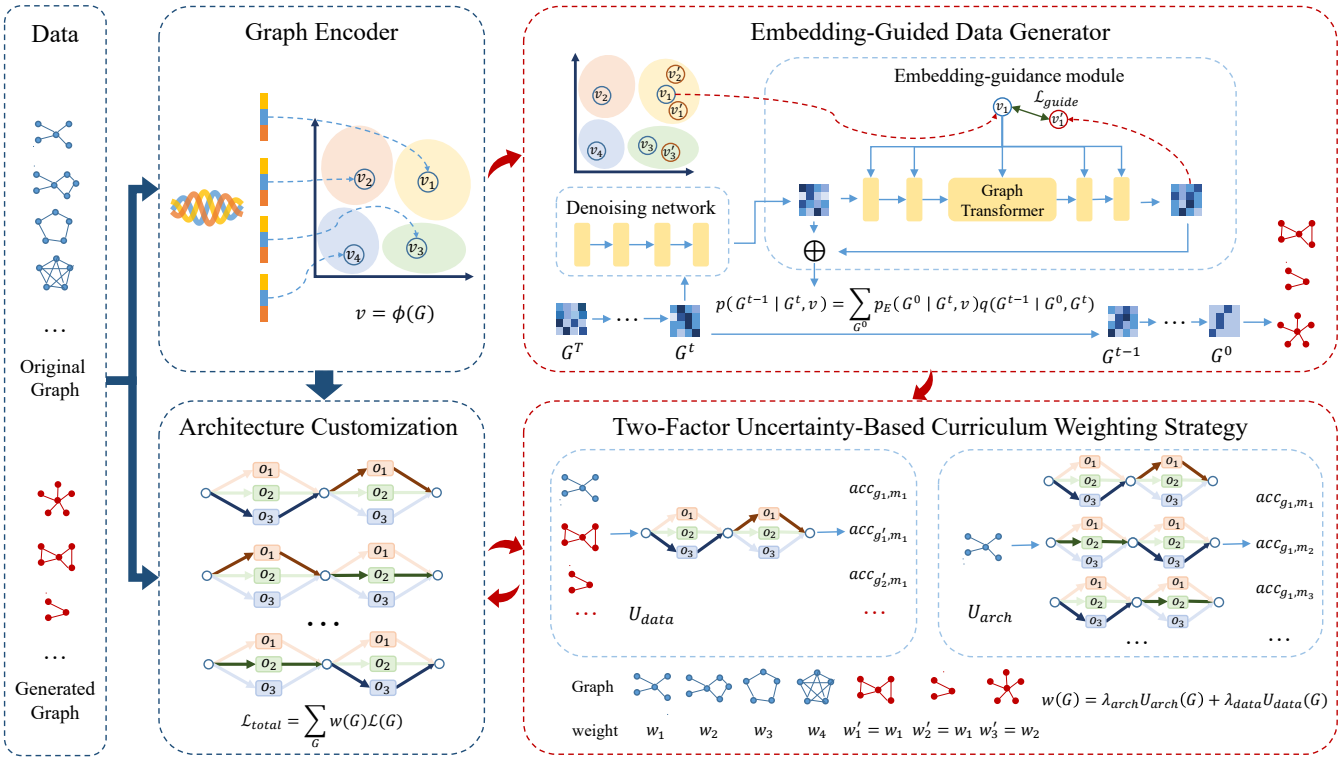

Figure 1: The framework of DCGAS. The architecture customizer uses the graph embedding obtained from graph encoder to customize the architecture for each graph. The embedding-guided data generator generates new graph data for training to enhance the generalization ability of architecture customizer. The two-factor uncertainty-based curriculum weighting strategy measures the uncertainty of the architecture customizer's performance on data to better facilitate important graphs in training.

we use a data generator with discrete embedding-guided diffusion to generate more data for training. In addition, we use a data scheduler with two-factor uncertainty-based curriculum weighting to calculate the weights of the original and generated data, improving the quality of the learned mappings. With the use of these two modules, our method can perform well on graph datasets with distribution shifts.

Embedding-Guided Data Generator

For many real-world graph scenarios, however, the amount of training data is limited, which makes it difficult to learn good mappings from graphs to architectures. Therefore, we design a data generator to generate additional graph data for training. Since it has been demonstrated (Cai, Zheng, and Chang 2018) that graph embedding can accurately represent the structure of graphs, we guide the generation of new graphs with the embeddings of existing graphs in the training dataset to improve the quality of graph generation.

Our embedding-guided data generator builds upon discrete graph diffusion generation model (Vignac et al. 2023), whose main component is a denoising network that can remove noise from noisy graph data. Generating new data involves starting with pure noise and iteratively denoising over many steps. However, the data generation of the diffusion model is unconditional and cannot be controlled with additional input. Since the space of all graphs is discrete and there is no gradient information of graph embedding regarding the graphs, we cannot use the gradient-based guidance methods (Vignac et al. 2023; Dhariwal and Nichol 2021) commonly used with diffusion models. We design an embedding guidance module to guide the generation of new graphs.

Let $p_\theta(G^0 \mid G^t)$ be the predictions of the denoising network. The data generation process starts with G^T sampled from a prior distribution and samples G^{t-1} from the following distribution in each step:

$$p(G^{t-1} \mid G^t) = \sum_{G^0} p_\theta(G^0 \mid G^t) q(G^{t-1} \mid G^0, G^t), \quad (7)$$

where $q(G^{t-1} \mid G^0, G^t)$ is defined in the forward process of the diffusion model. To make graph generation conditional on graph embedding v, we replace $p_\theta(G^0 \mid G^t)$ with $p_{\text{guide}}(G^0 \mid G^t, v)$ in graph generation, which can be represented as follows:

$$p_{\text{guide}}(G^0 \mid G^t, v) \propto p_\theta(G^0 \mid G^t) p(v \mid G^0), \quad (8)$$

where $p(v \mid G^0)$ is a guidance term defined as $p(v \mid G^0) \propto \exp(-\lambda \|v - v_G(G^0)\|^2)$ and $v_G(G^0)$ is the graph embeddings of G^0.

However, it is difficult to perform sampling using this definition as it is impossible to compute $p(v \mid G^0)$ for all values of G^0. Therefore, we approximate p_{guide} using an embedding

guidance module p_E that outputs a probability distribution over G^0 which is factorized over the nodes and edges of the graph.

$$p_E(G^0 \mid G^t, v) = \prod_{x_i \in X^0} p_E(x_i \mid G^t, v) \cdot \prod_{e_i \in E^0} p_E(e_i \mid G^t, v), \quad (9)$$

where X^0 and E^0 are the nodes and edges of G^0 respectively. The data generation process using the embedding guidance module is given in Algorithm 1.

We use an architecture based on graph transformer (Dwivedi and Bresson 2020) for the embedding guidance module. It takes the logit predictions (before softmax) of the denoising network as input as well as the target graph embedding, and outputs the adjusted logits. To facilitate efficient training of the embedding guidance module, we add a residual connection from the input to the output, and zero-initialize the last layer so the input is passed unmodified in the beginning of training. The model is trained by matching p_E to p_{guide}:

$$\begin{aligned}\mathcal{L}_{\text{guide}} &= \mathrm{KL}(p_E \parallel p_{\text{guide}}) \\ &= \mathbb{E}_{G\sim p_E}\left[\log p_E(G \mid G^t, v) \right. \\ &\qquad \left. - \log p_\theta(G \mid G^t) - \log p(v \mid G)\right] + C \\ &= \mathrm{KL}(p_E \parallel p_\theta) + \mathbb{E}_{G\sim p_E}\left[-\log p(v \mid G)\right] + C \\ &= \mathrm{KL}(p_E \parallel p_\theta) + \mathbb{E}_{G\sim p_E}\left[\lambda \|v - v_G(G)\|^2\right] + C,\end{aligned} \quad (10)$$

where C denotes terms independent of the model parameters. Since the second term in the loss cannot be differentiated directly, we compute $\mathcal{L}_{\text{guide}}$ in the following way, which has an unbiased estimate of the gradient over p_E:

$$\mathcal{L}_{\text{guide}} = \mathrm{KL}(p_E \parallel p_\theta) + \log p_E(G)\lambda\|v - v_G(G)\|^2, \quad (11)$$

where G is sampled from p_E.

Two-Factor Uncertainty-Based Curriculum Weighting Strategy

It is known that data have different importance in various stages of training, and scheduling data according to their importance using curriculum learning techniques can improve the efficiency of model training (Bengio et al. 2009; Wang, Chen, and Zhu 2022; Zhou et al. 2022b; Li, Wang, and Zhu 2023). For weight-sharing NAS methods, it has been demonstrated (Zhou et al. 2022c) that the importance of data can be reflected by its uncertainty over architectures. Training on data with high uncertainty can help the under-trained architectures to catch on, thus facilitating more accurate comparison between architectures and improving the quality of final architectures. Similarly, for graph NAS, the scheduling of training data is also crucial for search performance. Therefore, we designed a two-factor uncertainty-based curriculum weighting for graph data.

The two-factor uncertainty-based curriculum weighting consists of two uncertainty measurements. The first is the uncertainty over architectures, which is defined as the variance of the losses of the same graph data on multiple sampled architectures:

$$U_{\text{arch}}(G) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}(A_i, \theta, G) - \frac{1}{N}\sum_{j=1}^{N}\mathcal{L}(A_j, \theta, G)\right)^2, \quad (12)$$

where $\mathcal{L}$ is the loss function, θ is the current parameters of the supernet, and A_i is the i-th sampled architecture. It measures how well the NAS method can evaluate different architectures using this graph data.

The second measurement is the uncertainty over data, defined as the difference among the losses for graph data G and structurally similar graphs G'_i using the same architecture:

$$U_{\text{data}}(G) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}(A(G), \theta, G) - \mathcal{L}(A(G), \theta, G'_i)\right)^2, \quad (13)$$

where $A(G)$ is the customized architecture for G. Since the graph embedding can be used to represent the structure of graphs, we use the embedding-guided graph generator to generate G'_i. This uncertainty reflects the generalization capability of architectures when given similar input data.

Inspired by curriculum learning methods (Wang, Chen, and Zhu 2022; Shrivastava, Gupta, and Girshick 2016) where difficult and informative data are used early on during training, we consider data with higher uncertainty to have greater importance. Therefore, we assign weights to each data based on their uncertainties:

$$w(G) = \frac{\lambda_{\text{arch}}U_{\text{arch}}(G) + \lambda_{\text{data}}U_{\text{data}}(G)}{\sum_{G\in\mathcal{G}}\left(\lambda_{\text{arch}}U_{\text{arch}}(G) + \lambda_{\text{data}}U_{\text{data}}(G)\right)}, \quad (14)$$

where λ_{arch} and λ_{data} are hyperparameters. During training, the losses are multiplied by the weights, effectively scaling up or down the gradients depending on the data.

Architecture Customization

We build our architecture customization module based on differentiable NAS frameworks. The parameters of all possible architectures in the search space are shared using a supernet which contains all candidate operators. For each choice of operators, its output in the supernet is given as a mixture of operator outputs:

$$o^*(x) = \sum_{i=1}^{M} q_i o_i(x), \quad (15)$$

where o^* is the mixed operator in the supernet, o_i is an individual candidate operator, M is the number of candidates, x is the input to this operator, and q_i is the corresponding mixture coefficients.

In classical NAS methods (Liu, Simonyan, and Yang 2019), q_i is directly represented as a trainable value. However, our method needs to choose customized architectures, and q_i should be computed from the graph data. Since the structure of graph data is well represented by the graph embedding, we match the graph embeddings to a set of prototype vectors to obtain q_i:

$$q_i = \frac{\exp(u_i^T v)}{\sum_{j=1}^{M}\exp(u_j^T v)}, \quad (16)$$

Algorithm 1: Generate new graphs with embedding guidance

Input: Target graph embedding v

Sample G^T from the initial distribution
for t = T to 1 **do**
  Compute predictions $p_\theta(G^0 \mid G^t)$ using the denoising network
  Compute guidance p_E using the embedding guidance module via Eq. (9)
  Sample G^{t-1} from ..., using p_E as an approximation of p_{guide}
end for

where u_i is a trainable prototype vector. After computing q_i, the customized architecture can be used to produce predictions for the graph data.

Optimization Procedure

Our method is optimized using the overall loss $\mathcal{L}_{\text{all}}$:

$$\begin{aligned} \mathcal{L}_{\text{all}} &= \sum_G w(G)\mathcal{L}(G) \\ \mathcal{L}(G) &= (\mathcal{L}_{\text{CE}}(G) + \lambda_{\text{embed}}\mathcal{L}_{\text{embed}}(G)) \end{aligned}, \tag{17}$$

where $w(G)$ is the data weight assigned to G, $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss for graph classification, and $\mathcal{L}_{\text{embed}}$ is the auxiliary loss used to train the graph encoder. Additionally, the embedding guidance module for data generation is trained using $\mathcal{L}_{\text{guide}}$.

The optimization procedure for our method is shown in Algorithm 2. In each round of training, we first use a graph encoder to compute graph embeddings, then use the embedding-guided data generator to generate graphs with similar embeddings, customize their architecture, and calculate data weights through a data scheduler for parameter updates.

Experiments

To demonstrate the effectiveness of our proposed method, we conducted sufficient experiments on simulated and real datasets with distribution differences. And we demonstrate the effectiveness of each module in the method through ablation experiments.

Experiment Settings

Datasets We conducted experiments on a simulated dataset and three real datasets. There are distribution shifts between the training and testing sets of these datasets.

- Spurious-Motif (Qin et al. 2022; Wu et al. 2022b; Ying et al. 2019) is a synthetic dataset. The distribution shifts between its training and testing sets are constructed. It contains 18,000 graphs. The graph inside it consists of base subgraph (Tree, Ladder, Wheel denoted by S = 0, 1, 2) and motif subgraph (Cycle, House, Crane denoted by C = 0, 1, 2). The label of the graph is only determined by the motif subgraph. The test dataset consists of randomly combined graphs of subgraphs and motif subgraphs. The training data consists of graphs composed of base subgraphs and motif subgraphs combined according to probability distribution $P(S) = b \times \mathbb{I}(S = C) + \frac{1-b}{2} \times \mathbb{I}(S \neq C)$. The value of b determines the amount of distribution shifts between the training set and the test set. We selected three values (0.7, 0.8, and 0.9) of b to generate three datasets with significant data distribution shifts for experiments to demonstrate the effectiveness of our method. We use accuracy as the evaluation metric for the dataset.
- Ogbg-molhiv, Ogbg-molbace, Ogbg-molsider (Hu et al. 2020): they are molecular property prediction datasets consisting of 41,127, 1,513, 1,427 molecule graphs, respectively. We use ROC-AUC as the evaluation for these datasets.

Algorithm 2: The overall searching algorithm of DCGAS

Input: Training dataset $\mathcal{D}_{\text{train}}$, discrete graph diffusion model trained on $\mathcal{D}_{\text{train}}$

Initialize all trainable parameters
while not convergent **do**
  for graph data G in $\mathcal{D}_{\text{train}}$ **do**
    Compute embedding v of G using graph encoder
    Generate new graphs G' using graph generator with guidance from v via Alg. 1
    Compute embedding v' of G' using graph encoder
    Get customized architectures for G and G' via Eq. (16)
    Compute $\mathcal{L}_{\text{total}}$ using G and G' via Eq. (17)
    Store the value of losses for G and G'
    Update loss weighting for G via Eq. (14)
    Update parameters of graph encoder and architecture customizer using $\mathcal{L}_{\text{all}}$
    Update parameters of the supernet using $\mathcal{L}_{\text{all}}$
    Compute $\mathcal{L}_{\text{guide}}$ using G via Eq. (10)
    Update parameters of data generator using $\mathcal{L}_{\text{guide}}$
  end for
end while

Baselines We compared our method with the following baselines:

- Manually design GNNs: GCN (Kipf and Welling 2017), GAT (Velickovic et al. 2018), GIN (Xu et al. 2019), SAGE (Hamilton, Ying, and Leskovec 2017), and GraphConv (Morris et al. 2019) are GNN architecture. ASAP (Ranjan, Sanyal, and Talukdar 2020) and DIR (Wu et al. 2022b) are recent methods, which achieved good performance on several graph datasets. In particular, DIR is specially designed for out-of-generalization performance.
- Neural architecture search: DARTS (Liu, Simonyan, and Yang 2019) is a differentiable architecture search method. In addition, we also compare our method with random search.
- Graph neural architecture search: GNAS (Gao et al. 2020b) and PAS (Wei et al. 2021) are graph neural architecture search methods for i.i.d. graph datasets.

Method	$b = 0.7$	$b = 0.8$	$b = 0.9$
GCN	$48.39_{\pm 1.69}$	$41.55_{\pm 3.88}$	$39.13_{\pm 1.76}$
GAT	$50.75_{\pm 4.89}$	$42.48_{\pm 2.46}$	$40.10_{\pm 5.19}$
GIN	$36.83_{\pm 5.49}$	$34.83_{\pm 3.10}$	$37.45_{\pm 3.59}$
SAGE	$46.66_{\pm 2.51}$	$44.50_{\pm 5.79}$	$44.79_{\pm 4.83}$
GraphConv	$47.29_{\pm 1.95}$	$44.67_{\pm 5.88}$	$44.82_{\pm 4.84}$
MLP	$48.27_{\pm 1.27}$	$46.73_{\pm 3.48}$	$46.41_{\pm 2.34}$
ASAP	$54.07_{\pm 13.85}$	$48.32_{\pm 12.72}$	$43.52_{\pm 8.41}$
DIR	$50.08_{\pm 3.46}$	$48.22_{\pm 6.27}$	$43.11_{\pm 5.43}$
Random	$45.92_{\pm 4.29}$	$51.72_{\pm 5.38}$	$45.89_{\pm 5.09}$
DARTS	$50.63_{\pm 8.90}$	$45.41_{\pm 7.71}$	$44.44_{\pm 4.42}$
GNAS	$55.18_{\pm 18.62}$	$51.64_{\pm 19.22}$	$37.56_{\pm 5.43}$
PAS	$52.15_{\pm 4.35}$	$43.12_{\pm 5.95}$	$39.84_{\pm 1.67}$
GRACES	$\underline{65.72_{\pm 17.47}}$	$\underline{59.57_{\pm 17.37}}$	$\underline{50.94_{\pm 8.14}}$
DCGAS	$\mathbf{87.68}_{\pm 6.12}$	$\mathbf{75.45}_{\pm 17.40}$	$\mathbf{61.42}_{\pm 16.26}$

Table 1: Test accuracy of different methods on Spurious-Motif datasets. Values after $\pm$ denote the standard deviations. The best results are in bold and the second best results are underlined.

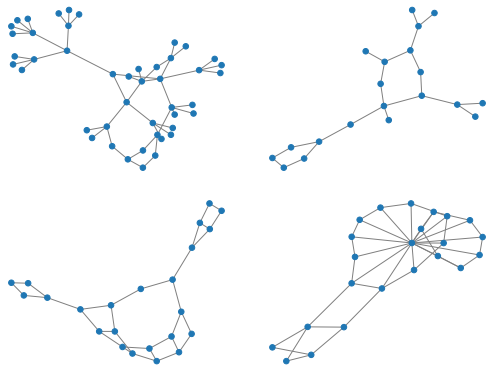

Figure 2: Examples of generated graphs for Spurious-Motif datasets.

GRACES (Qin et al. 2022) is a state-of-the-art graph neural architecture search method for non-i.i.d. graph datasets.

For our experiments, the reported mean and standard deviation of metrics are computed from 10 runs with random seeds.

Results on Synthetic Datasets

As seen in Table 1, the methods generally have worse performance for datasets with larger b, which matches the intuition as larger b indicates more significant distribution shifts between training and testing. Overall, NAS works better than GNNs on the three simulated datasets with distribution shifts, suggesting that it is necessary to customize architectures for graph classification tasks. GNNs designed for non-I.I.D. datasets (DIR) perform better than GNNs designed for I.I.D. datasets, and graph NAS methods designed for non-I.I.D. datasets (GRACES and DCGAS) work better than graph NAS methods designed for I.I.D. datasets.

Method	HIV	SIDER	BACE
GCN	$75.99_{\pm 1.19}$	$59.84_{\pm 1.54}$	$68.93_{\pm 6.95}$
GAT	$76.80_{\pm 0.58}$	$57.40_{\pm 2.01}$	$75.34_{\pm 2.36}$
GIN	$77.07_{\pm 1.49}$	$57.57_{\pm 1.56}$	$73.46_{\pm 5.24}$
SAGE	$75.58_{\pm 1.40}$	$56.36_{\pm 1.32}$	$74.85_{\pm 2.74}$
GraphConv	$74.46_{\pm 0.86}$	$56.09_{\pm 1.06}$	$78.87_{\pm 1.74}$
MLP	$70.88_{\pm 0.83}$	$58.16_{\pm 1.41}$	$71.60_{\pm 2.30}$
ASAP	$73.81_{\pm 1.17}$	$55.77_{\pm 1.18}$	$71.55_{\pm 2.74}$
DIR	$77.05_{\pm 0.57}$	$57.34_{\pm 0.36}$	$76.03_{\pm 2.20}$
DARTS	$74.04_{\pm 1.75}$	$60.64_{\pm 1.37}$	$76.71_{\pm 1.83}$
PAS	$71.19_{\pm 2.28}$	$59.31_{\pm 1.48}$	$76.59_{\pm 1.87}$
GRACES	$\underline{77.31_{\pm 1.00}}$	$\underline{61.85_{\pm 2.58}}$	$\underline{79.46_{\pm 3.04}}$
DCGAS	$\mathbf{78.04}_{\pm 0.71}$	$\mathbf{63.46}_{\pm 1.42}$	$\mathbf{81.31}_{\pm 1.94}$

Table 2: Test ROC-AUC of different methods on real-world molecular property prediction datasets. Values after $\pm$ denote the standard deviations. The best results are in bold and the second best results are underlined.

Our method achieves significant improvement on all three datasets. This is due to the fact that our method learns a good mapping from data to architecture. Specifically, we design an embedding-guided data generator to generate graphs with similar structures, which provide more information to facilitate the model's understanding of graph structures. The generator enables the model to better adapt to graphs with different structures, and improves the generalization of architecture customization. As demonstrated in Figure 2, the data generator can generate diverse graphs that capture the graph structures in the dataset.

Additionally, we design two-factor uncertainty-based curriculum weighting strategy to evaluate the learning importance of the data, allowing our method to stably customize good architectures for similarly structured graph data. Thus, it can be said that our method understands the structure of graphs well and learns the mapping of graph data to architectures, which allows us to customize suitable architectures for data with large differences in data distribution.

Results on Real-World Datasets

The results of the experiments are shown in Table 2 and it can be seen that the NAS methods are still generally better. The methods designed for non-I.I.D. datasets also still achieved better results. Our method is better compared to all baselines. The experimental results prove that our method is still effective on real-world datasets.

In addition, we can find that the performance improvement of our method over baselines is more significant for datasets like Ogbg-molsider and Ogbg-molbace, which have much smaller data sizes compared to Ogbg-molhiv. A possible explanation is that learning information about graph structures is more difficult with limited data, and our method improves the performance using the data generator. We also compare our method with GRACES using subsets of train-

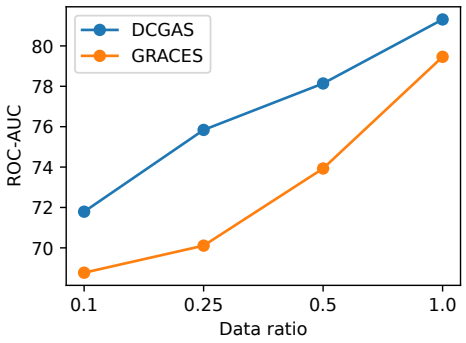

Figure 3: The performance of DCNAS and GRACES trained using different ratios of data from Ogbg-molbace.

Method	$b = 0.7$	$b = 0.8$	$b = 0.9$
None	$72.44_{\pm 17.13}$	$61.73_{\pm 20.23}$	$54.89_{\pm 15.68}$
Data gen	$82.05_{\pm 5.67}$	$68.25_{\pm 11.32}$	$60.07_{\pm 12.62}$
Curriculum	$80.38_{\pm 8.29}$	$63.99_{\pm 19.60}$	$56.08_{\pm 13.80}$
DCGAS	$\mathbf{87.68}_{\pm 6.12}$	$\mathbf{75.45}_{\pm 17.40}$	$\mathbf{61.42}_{\pm 16.26}$

Table 3: Ablation study on Spurious-Motif datasets.

ing data from Ogbg-molbace. As demonstrated in Figure 3, our method remains competitive when the amount of training data is limited.

Ablation Study

We conducted ablation experiments to demonstrate the effectiveness of each module in our method. The following methods are tested in the experiments:

- Curriculum: only the two-factor uncertainty-based curriculum weighting strategy module is used.
- Data gen: only the embedding-guided data generator module is used.
- None: neither of these modules is used.

The experimental results are shown in Table 3. It can be seen that both the data generator module and the curriculum weighting module lead to performance improvements, and the best results are achieved when both modules are used. This demonstrates that both components in our method are indispensable to achieving satisfactory performance on tasks with distribution shifts.

Related Work

Graph Generation

Graph generation is a widely studied field, which has research value in many fields. Influenced by the emergence of generation models in computer vision, recent graph generation has focused more on using deep learning methods for generation, such as auto-regressive model (Goyal, Jain, and Ranu 2020; Bacciu and Podda 2021), variational autoencoder (Du et al. 2022a,b), normalizing flow (Zang and Wang 2020; Luo, Yan, and Ji 2021), generative adversarial networks (Gamage et al. 2020; Pölsterl and Wachinger 2020), and diffusion model (Jo, Lee, and Hwang 2022; Vignac et al. 2023). GDSS (Jo, Lee, and Hwang 2022) uses a diffusion model for generation based on a system of SDEs. DiGress (Vignac et al. 2023) is a discrete graph diffusion model generation method, which can preserve sparsity in the noisy graphs and improve generation quality.

Graph Neural Architecture Search

Neural architecture search is a research direction in automatic machine learning that has emerged in recent years. Traditional manual model design methods require professional knowledge, while neural architecture search can automatically search for the optimal architecture in a given search space. Recent research has shown that neural architecture search can customize the optimal architecture for many tasks compared to manually designed models. Various search strategies have been proposed, such as reinforcement learning (RL) based NAS (Zoph and Le 2017; Jaâfra et al. 2019), evolutionary algorithms based NAS (Real et al. 2017; Liu et al. 2023), and gradient based NAS (Liu, Simonyan, and Yang 2019; Ye et al. 2022).

Graph neural architecture search has also received widespread attention. GraphNAS (Gao et al. 2020a) is the first to use reinforcement learning methods to search for graph neural network architectures, which integrate excellent architectures in previous GNN fields as search spaces, such as GCN, GAT, etc. After this work, many works (Gao et al. 2020b; Wei et al. 2021; Qin et al. 2022; Cai et al. 2021; Li et al. 2021; Qin et al. 2021; Guan, Wang, and Zhu 2021; Zhang et al. 2023b,a; Guan et al. 2022) on graph NAS emerged and excellent architectures were found. Recently, graph NAS, which focuses on graph classification tasks, has also been widely studied. Its characteristic is that the dataset contains graphs, such as protein molecule datasets. Some works (Wei et al. 2021) studied the problem of graph classification on independent identically distributed datasets. GRACES (Qin et al. 2022) studied the problem of graph classification on non-independent identically distributed datasets.

Conclusion

We propose Data-Augmented Curriculum Graph Neural Architecture Search (DCGAS) method for graph classification under distribution shifts, which customizes architectures for each graph data through learning the mapping from graph data to architectures. We design an embedding-guided data generator to generate more graph data with similar structures to a given graph for training. Moreover, we design a two-factor uncertainty-based curriculum weighting strategy which measures the uncertainty of the architecture customizer's performance on data to schedule data in training. They greatly enhance the generalization ability of architecture customization. The effectiveness of our method has been demonstrated through experiments on both simulated and real datasets, which show our method achieves state-of-the-art performance for the graph classification task under distribution shifts.

Acknowledgments

This work was supported by the National Key Research and Development Program of China No. 2020AAA0106300, National Natural Science Foundation of China (No. 62250008, 62222209, 62102222), Beijing National Research Center for Information Science and Technology under Grant No. BNR2023RC01003, BNR2023TD03006, and Beijing Key Lab of Networked Multimedia.

References

Bacciu, D.; and Podda, M. 2021. Graphgen-redux: a Fast and Lightweight Recurrent Model for labeled Graph Generation. In *International Joint Conference on Neural Networks, IJCNN 2021*, 1–8. IEEE.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, volume 382 of *ACM International Conference Proceeding Series*, 41–48. ACM.

Cai, H.; Zheng, V. W.; and Chang, K. C. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans. Knowl. Data Eng.*, 30(9): 1616–1637.

Cai, S.; Li, L.; Deng, J.; Zhang, B.; Zha, Z.; Su, L.; and Huang, Q. 2021. Rethinking Graph Neural Architecture Search From Message-Passing. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021*, 6657–6666. Computer Vision Foundation / IEEE.

Dhariwal, P.; and Nichol, A. Q. 2021. Diffusion Models Beat GANs on Image Synthesis. In *Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, 8780–8794.

Du, Y.; Guo, X.; Shehu, A.; and Zhao, L. 2022a. Interpretable Molecular Graph Generation via Monotonic Constraints. In *Proceedings of the 2022 SIAM International Conference on Data Mining, SDM 2022*, 73–81. SIAM.

Du, Y.; Guo, X.; Shehu, A.; and Zhao, L. 2022b. Interpretable Molecular Graph Generation via Monotonic Constraints. In *Proceedings of the 2022 SIAM International Conference on Data Mining, SDM 2022*, 73–81. SIAM.

Dwivedi, V. P.; and Bresson, X. 2020. A Generalization of Transformer Networks to Graphs. arXiv:2012.09699.

Gamage, A.; Chien, E.; Peng, J.; and Milenkovic, O. 2020. Multi-MotifGAN (MMGAN): Motif-Targeted Graph Generation And Prediction. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020*, 4182–4186. IEEE.

Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; and Hu, Y. 2020a. Graph Neural Architecture Search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 1403–1409. ijcai.org.

Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; and Hu, Y. 2020b. Graph Neural Architecture Search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 1403–1409. ijcai.org.

Goyal, N.; Jain, H. V.; and Ranu, S. 2020. GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation. In *WWW '20: The Web Conference 2020*, 1253–1263. ACM / IW3C2.

Guan, C.; Wang, X.; Chen, H.; Zhang, Z.; and Zhu, W. 2022. Large-Scale Graph Neural Architecture Search. In *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, 7968–7981. PMLR.

Guan, C.; Wang, X.; and Zhu, W. 2021. AutoAttend: Automated Attention Representation Search. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, 3864–3874. PMLR.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *Annual Conference on Neural Information Processing Systems 2017*, 1024–1034.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Jaâfra, Y.; Laurent, J. L.; Deruyver, A.; and Naceur, M. S. 2019. Reinforcement learning for neural architecture search: A review. *Image Vis. Comput.*, 89: 57–66.

Jo, J.; Lee, S.; and Hwang, S. J. 2022. Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations. In *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, 10362–10383. PMLR.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Li, H.; Wang, X.; and Zhu, W. 2023. Curriculum Graph Machine Learning: A Survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023*, 6674–6682. ijcai.org.

Li, Y.; Wen, Z.; Wang, Y.; and Xu, C. 2021. One-shot Graph Neural Architecture Search with Dynamic Search Space. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, 8510–8517. AAAI Press.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net.

Liu, Y.; Sun, Y.; Xue, B.; Zhang, M.; Yen, G. G.; and Tan, K. C. 2023. A Survey on Evolutionary Neural Architecture Search. *IEEE Trans. Neural Networks Learn. Syst.*, 34(2): 550–570.

Luo, Y.; Yan, K.; and Ji, S. 2021. GraphDF: A Discrete Flow Model for Molecular Graph Generation. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, 7192–7203. PMLR.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 4602–4609. AAAI Press.

Pölsterl, S.; and Wachinger, C. 2020. Adversarial Learned Molecular Graph Inference and Generation. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020*, volume 12458 of *Lecture Notes in Computer Science*, 173–189. Springer.

Qin, Y.; Wang, X.; Zhang, Z.; Xie, P.; and Zhu, W. 2022. Graph Neural Architecture Search Under Distribution Shifts. In *International Conference on Machine Learning, ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, 18083–18095. PMLR.

Qin, Y.; Wang, X.; Zhang, Z.; and Zhu, W. 2021. Graph Differentiable Architecture Search with Structure Learning. In *Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, 16860–16872.

Ranjan, E.; Sanyal, S.; and Talukdar, P. P. 2020. ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, 5470–5477. AAAI Press.

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Leon-Suematsu, Y. I.; Tan, J.; Le, Q. V.; and Kurakin, A. 2017. Large-Scale Evolution of Image Classifiers. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, 2902–2911. PMLR.

Shrivastava, A.; Gupta, A.; and Girshick, R. B. 2016. Training Region-Based Object Detectors with Online Hard Example Mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, 761–769. IEEE Computer Society.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.

Vignac, C.; Krawczuk, I.; Siraudin, A.; Wang, B.; Cevher, V.; and Frossard, P. 2023. DiGress: Discrete Denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.

Wang, X.; Chen, Y.; and Zhu, W. 2022. A Survey on Curriculum Learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(9): 4555–4576.

Wei, L.; Zhao, H.; Yao, Q.; and He, Z. 2021. Pooling Architecture Search for Graph Classification. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management*, 2091–2100. ACM.

Wu, L.; Lin, H.; Huang, Y.; and Li, S. Z. 2022a. Knowledge Distillation Improves Graph Structure Augmentation for Graph Neural Networks. In *Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*.

Wu, Y.; Wang, X.; Zhang, A.; He, X.; and Chua, T. 2022b. Discovering Invariant Rationales for Graph Neural Networks. In *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net.

Ye, P.; Li, B.; Li, Y.; Chen, T.; Fan, J.; and Ouyang, W. 2022. β-DARTS: Beta-Decay Regularization for Differentiable Architecture Search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022*, 10864–10873. IEEE.

Ying, Z.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 9240–9251.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph Contrastive Learning with Augmentations. In *Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Zang, C.; and Wang, F. 2020. MoFlow: An Invertible Flow Model for Generating Molecular Graphs. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 617–626. ACM.

Zhang, Z.; Wang, X.; Guan, C.; Zhang, Z.; Li, H.; and Zhu, W. 2023a. AutoGT: Automated Graph Transformer Architecture Search. In *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.

Zhang, Z.; Zhang, Z.; Wang, X.; Qin, Y.; Qin, Z.; and Zhu, W. 2023b. Dynamic Heterogeneous Graph Attention Neural Architecture Search. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, 11307–11315. AAAI Press.

Zhao, T.; Liu, Y.; Neves, L.; Woodford, O. J.; Jiang, M.; and Shah, N. 2021. Data Augmentation for Graph Neural Networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, 11015–11023. AAAI Press.

Zhou, K.; Huang, X.; Song, Q.; Chen, R.; and Hu, X. 2022a. Auto-GNN: Neural architecture search of graph neural networks. *Frontiers Big Data*, 5.

Zhou, Y.; Chen, H.; Pan, Z.; Yan, C.; Lin, F.; Wang, X.; and Zhu, W. 2022b. CurML: A Curriculum Machine Learning Library. In *MM '22: The 30th ACM International Conference on Multimedia*, 7359–7363. ACM.

Zhou, Y.; Wang, X.; Chen, H.; Duan, X.; Guan, C.; and Zhu, W. 2022c. Curriculum-NAS: Curriculum Weight-Sharing Neural Architecture Search. In *MM '22: The 30th ACM International Conference on Multimedia*, 6792–6801. ACM.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.