AutoCoG: A Unified Data-Model Co-Search Framework for Graph Neural Networks

Anonymous[1]

[1]Anonymous Institution

Abstract Neural architecture search (NAS) has demonstrated success in discovering promising architectures for vision or language modeling tasks, and it has recently been introduced to searching for graph neural networks (GNNs) as well. Despite the preliminary success, GNNs struggle in dealing with heterophily or low-homophily graphs where connected nodes may have different class labels and dissimilar features. To this end, we propose co-optimizing both the input graph topology and the model's architecture topology simultaneously. That yields **AutoCoG**, the first unified data-model co-search NAS framework for GNNs. By defining a highly flexible data-model co-search space, **AutoCoG** is gracefully formulated as a principled bi-level optimization that can be end-to-end solved by the differentiable search methods. Experiments show **AutoCoG** achieves an average performance gain across all datasets of 3.18% over the following best approach and ranks best against all other state-of-the-art methods with an average ranking of 2.5.

1 Introduction

Graph neural networks (GNNs) have emerged as promising tools to analyze networked data in various real-world scenarios, such as social media Grover and Leskovec (2016) and biochemical graph analytics Zitnik and Leskovec (2017). Specifically, GNNs apply recursive message passing to learn the embedding representation of each node via aggregating the representations of its neighbors and itself. Motivated by the significant success of node embedding learning, plenty of GNN variants have been explored for the diverse downstream graph analysis tasks, including GCN Kipf and Welling (2016a), GraphSAGE Hamilton et al. (2018), and GCNII Chen et al. (2020a).

However, training GNNs is notoriously challenging, more so when they are train under heterophily or disassociative graphs, not to mention deep GNNs Chen et al. (2020a). First, since graphs abstract diverse data sources and present tremendous heterogeneity, the success of GNNs is often accompanied by extensive tuning of model architectural hyperparameters to characterize specific graph data. For example, it was reported that graph attention networks GAT Veličković et al. (2018) are sensitive to the number of attention heads, which has to be carefully searched for the citation networks and the protein-protein interaction data, respectively. Second, in the real world graphs often opposites attract which inevitably lead to noisy setting where GNNs tend to suffer from overfitting and generalize poorly to the unseen testing data. Third, despite the potential of deep GNNs in learning the informative high-order neighborhood, the training of deep GNNs is widely known to be limited by the issues of over-smoothing, gradient vanishing, and over-squashing Chen et al. (2020a).

Recently, the automated graph neural architecture search (NAS), graph augmentation tricks, and deeper architectures have been independently proposed to tackle the above GNN training challenges partially. Expressly, most of the existing automated efforts are limited to neural architecture tuning, while graph augmentation is often overlooked and untouched despite often being effective to gain performance Li and King (2020); Zhou et al. (2019a). This is primarily because changes to the existing graph structure can have a cascading effect on the process of information aggregation, which adds a new layer of complexity above the already complex architecture tuning problem.

Additionally, existing GNN NAS works are known to scale poorly in deeper architectures. This is primarily due the exploding search space which makes training unstable. Previous efforts have limited themselves in searching the shallow GNNs with less than 3 layers. Finally, Figure(1) illustrates circumstances where the aggregation mechanism fails due to unfavorable graph topology thus, it remains a daunting task to optimize the design philosophy for GNNs comprehensively.

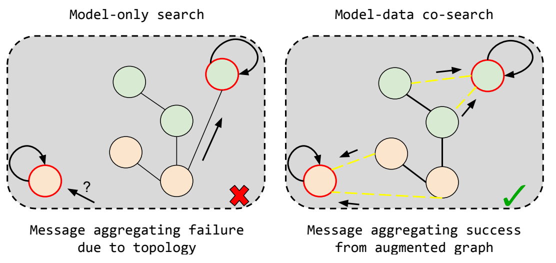
Model-only search
Model-data co-search
?
Message aggregating failure due to topology
Message aggregating success from augmented graph

Figure 1: In red outline are nodes with poor aggregation, black arrows, due to a graph topology under heterophily. This can be mitigated by learning to place meaningful edges, yellow lines, to facilitate proper message propagation. Motivate to solve this performance problem, we propose co-adapting both graph and model in a end-to-end manner.

To bridge the gaps, we propose **AutoCoG**, the first NAS framework towards **unified data-model co-search for GNNs** to specifically tame the problem training under heterophily condition. Besides automatically optimizing the GNN model architecture, we propose to simultaneously optimize the input graph topology, via progressively growing and pruning using a separate GNN model to learn to attune the graph to the proposed architecture. Additionally, by defining the highly flexible data-model co-search space, **AutoCoG** is formulated as a principled bi-level optimization that can be end-to-end solved by the differentiable search methods. To scale up our core framework to searching deep GNN architectures, we curb an explosive search space as the number of layers increased by performing multiple searching stages with increasing depth, as inspired by Chen et al. (2019c). Additionally, for each search stages, we evolve the graph by growing/pruning it at the same time. To stabilize the search landscape from the shifting topologies of graph and model, we further utilize Chen et al. (2020a) to combat the over-smoothing/over-squashing issues.

Together, our framework ensures a reliable way to discover powerful architectures, a stable model training environment, and state-of-the-art results to train graph different degree of homophily. **AutoCoG** searches for and trains on deep or shallow graph neural networks to successfully achieve state-of-the-art results in Web datasets, Actor, Coauthor, and Wikipedia benchmarks.

In summary, our three contributing novelties are:

- We propose **AutoCoG**, the first NAS framework towards unified data-model co-search for GNNs. Our novel bi-level optimization formulation uniquely enables the end-to-end discovery of state-of-the-art GNN model and graph altogether.
- We perform extensive analysis the resulting learned graph-structures for each benchmarks. To strengthen the co-search framework, we organically integrate several techniques to directly combat issues of searching unreliability, training instability, and scalability, that have previously plagued NAS approaches for searching deeper GNNs.
- Experiments show **AutoCoG** achieves an average performance gain across all datasets of 3.18% over the following best approach and ranks best against all other state-of-the-art methods with an average ranking of 2.5.

2 Related works

Graph neural networks. Motivated by the state-of-the-art results of GNNs in graph analytics, there have been numerous GNN variants Bruna et al. (2013); Hamilton et al. (2017); Xu et al. (2019); Chen et al. (2020a); Wu et al. (2019a). Most of these existing approaches fit within the category of spatial GNNs. Namely, following the spatial message passing strategy, the core idea of GNNs is to learn the embedding representation of a node by aggregating the embeddings of its neighbors and node itself recursively. The previous empirical studies show that GNNs often achieve the best performance with less than 3 layers Kipf and Welling (2016a); Veličković et al. (2018). Key limitations of GNNs are their performances decrease significantly with the increasing of model depth and the degree of graph homophily they operate on. As the graph convolutional layer increases, the node representations will converge to indistinguishable vectors due to the recursive neighborhood aggregation and non-linear activation Li et al. (2018); Oono and Suzuki (2020), which is well recognized as over-smoothing issue NT and Maehara (2019); Chen et al. (2019a); Alon and Yahav (2020); Chien et al. (2021); Huang et al. (2020).

Graph augmentation. Data augmentation methods has been widely applied to improve the generalization performances of deep neural networks, such as convolutional and recurrent neural networks Shorten and Khoshgoftaar (2019); Antoniou et al. (2017); Feng et al. (2021). They aim to craft the out-of-distribution training data to avoid overfitting with the customized augmentation policies. In the graph analytics, GNNs are prone to overfit the naturally noisy training graphs, which may miss the ground-truth nodes/edges or contain the erroneous information Zügner et al. (2018). Different from the grid-like image data, the graph augmentation is often operated on the adjacency structure or node features. The existing graph augmentations could be catagorized into the following two classes. (i) The random augmentation either drops/adds edges to modify the graph, or masks parts of the node features Rong et al. (2020a); You et al. (2020b); Feng et al. (2020). (ii) The differentiable augmentation learns to optimize the adjacency affinity matrix by minimizing the concerned task loss. Based upon the computed affinity matrix, the differentiable augmentation either continuously combines it into the original adjacency matrix Zhao et al. (2020b); Chen et al. (2020b), or samples the discrete edges to formulate new graph Chen et al. (2019b).

Neural architecture search. Targeting at alleviating the laborious hyperparameter tuning, NAS automates the designing of good neural architectures for any a given application. It is shockingly reported that the searched neural architectures could outperform the human-designed ones in many real-world scenarios, such as image classification Zoph and Le (2016); Zoph et al. (2018) and generation Wang and Huan (2019); Gong et al. (2019). Most of NAS frameworks apply one of the following search algorithms: reinforcement learning (RL) Pham et al. (2018); Baker et al. (2016), evolution algorithm (EA) Liu et al. (2017); Miikkulainen et al. (2019); Xie and Yuille (2017), and one-shot differentiable search Liu et al. (2018); Zela et al. (2020). There are several recent efforts to conjoin the researches of GNNs and NAS Gao et al. (2019); Zhou et al. (2019b); You et al. (2020a); Ding et al. (2020); Zhao et al. (2020a). However, all of them are limited in exploring the shallow GNNs, and fail to denoise the underlying graph to further ameliorate the model performance. In this work, we aim to simultaneously search the deep GNN models and graph structure to optimize the downstream graph analytics.

Co-Adaptive Search between graph's structure and model's architecture. GASSO Qin et al. (2021) is a recent work that similarly proposes the idea of model-graph co-search. Yet two differentiation factors uniquely defined our two works. Firstly, GASSO is a technique that learns attention coefficients $G \in [0, 1]$ only for existing edges E, which is mathematically equivalent to graph attention neural networks (Qin et al., 2021). In contrast, **Auto-CoG** directly modifies the graph's structure via pruning poorly attended edges and adding new un-seen edges. Thus the derived graph structure is unique from the original underlying graph. Finally, GASSO employs a coarse macro-level search space with only eight operators and two layers. Its design decision space is shallow (small), consisting of only 256 unique combinations. Theoretically, by searching the optimal attention function in our **Auto-Cog**, we could approximate the "attentional structure learning" in GASSO.

3 Methodology

Preliminary. We briefly review the basic of a message-passing based graph convolution network (GCN) and the definition of homophily.

First, the homophily or edge-homophily ratio, of a graph measures the ratio between intra-node pairs (v, w) overall all edges E and is given as:

$$\frac{|\{(v,w) : (v,w) \in E \wedge y_v = y_w\}|}{E} \tag{1}$$

Second, given a GCN, its k-th layer could be written generally as:

$$\begin{aligned} h_i^{(k)} &= \mathrm{AGGR}(\{a_{ij}^{(k)} W^{(k)} x_j^{(k-1)} : j \in \mathcal{N}(i)\}) \\ x_i^{(k)} &= \sigma(\mathrm{COMB}(W^{(k)} x_i^{(k-1)}, h_i^{(k)})) \end{aligned} \tag{2}$$

$x_i^{(k)}$ denotes the node embedding of element i at k-th layer. $W^{(k)} \in \mathbb{R}^{D \times D}$ represents the learnable layer-wise weights for all $\{x_i : i \in |V|\}$, where $|V|$ is our total number of nodes and D our number of hidden features. $a_{ij}^{(k)}$ dictates the attention coefficient between i and j derived from some Attention function. $\mathcal{N}(i)$ denotes the neighboring nodes of node i from a graph G. $h_i^{(k)}$ is the resulting embeddings after applying an AGGR function to aggregate a set of neighboring embeddings from the previous $k-1$ layers. In addition, function COMB incorporates information from itself with its neighboring embeddings $h_i^{(k)}$, and σ provides the nonlinear activation.

3.1 Unified Data-Model Co-Search Space

3.1.1 Model Search Space: Attention, Activation, and Skip Connection.

Defining the Model Search Space. The design of model search space should achieve a balanced trade-off between the diversity and efficiency. Although a large search space subsumes the diverse GNN architectures to adapt to the different graph analysis tasks, it would be extremely time-consuming to explore the optimal design. In the existing search spaces of

GNNs (Gao et al., 2019; Zhou et al., 2019a; You et al., 2020a), they often contain the architecture components of hidden units, attention, aggregation, combination, and activation functions, as well as the skip connections. To efficiently search the outperforming shallow and deep GNNs, we compare the effectiveness of each component, and greatly shrink down the search space to focus on three key components: the Activation function, the Attention module, and the skip connections. They are generally believed to impact GNN's expressive capability and depth scalability (Chen et al., 2021b). We fix the Aggreation function and Combination function to be simple summation, and treat the hidden units as hyperparameter. Below we lay out our searchable design for them one-by-one:

Table 1: The set of attention functions, where $||$ denotes the concatenation operation, $\bar{a}, \bar{a}_i, \bar{a}_j$ denote learn-able vectors, W_G denotes the trainable matrix.

Attention Choice	Expression Form
GCN	$\frac{1}{\sqrt{\lvert\mathcal{N}(i)\rvert\lvert\mathcal{N}(j)\rvert}}$
COS	$\bar{a}(W^{(k)} x_i^{(k-1)} \Vert W^{(k)} x_j^{(k-1)})$
LINEAR	$\tanh(\bar{a}_l W^{(k)} x_i^{(k-1)} \Vert W^{(k)} x_j^{(k-1)})$
GERE-LINEAR	$W_G \tanh(W^{(k)} x_i^{(k-1)} + W^{(k)} x_i^{(k-1)})$
GAT	$\mathrm{LeakyReLU}(\bar{a}(W^{(k)} x_i^{(k-1)} \Vert W^{(k)} x_j^{(k-1)}))$
GAT-SUM	$a_{ij}^{(k)} + a_{ij}^{(k)}$ based on GAT
CONST	1

- *Attention Search Space*: Attention mechanism has been shown by (Veličković et al., 2018) to effectively stabilize training by placing proper neighborhood scaling with attention coefficient a_{ij}. We list our attention choices in Table 1.

- *Activation Search Space*: for basic activation functions, we search among these operations {ReLU, Sigmoid, Tanh, Linear, SoftPlus, LeakyReLU, ReLU6, ELU}.

- *Skip Connection Search Space*: For an L-Layer GCN, various skip connections can be applied to overcome the effect of over-smoothing. Previous deep GCN works (Chen et al., 2020a; Zhang et al., 2020; Chen et al., 2021b) illustrated a significant correlation between the type of skip-connections to the performance. We include three skip-connection types — i) Initial Connection, ii) Jumping-Knowledge aggregation.

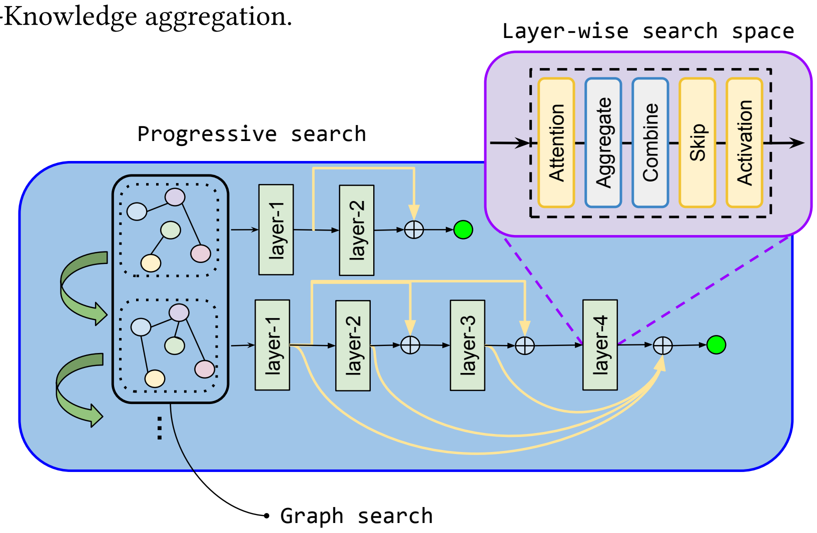

Figure 2: An illustration of **AutoCoG** framework. We marked the components we searched on as yellow; this notation also extends to the different skip connections illustrated in the progressive search box. Furthermore, we narrow down our search choice for each step within progressive search while extending the model's layers. We also perform graph augmentation for every step. Further details on the process can be found in Algorithm(1) in the appendix

3.1.2 Graph Augmentation.

We often expect a clustering of liked nodes when operating on graph data in a "like attracts like" world. However, in reality, when modeling complex relationships, we often observe the opposite, where node identities are often best described by contrasting with their neighbors in "different attract" relationships. Under such heterophily circumstance, GNNs' performances degrade (Pei et al., 2020a; Zhu et al., 2020a; Battaglia et al., 2018), which makes sense intuitively, since relating unrelated nodes can lead to class obscurity (Zhu et al., 2020a), i.e over-smoothing. Thus overcoming heterophily with model architecture alone is difficult and often requires complicated, and exotic works flow (Abu-El-Haija et al., 2019; Pei et al., 2020b; Lim et al., 2021). Lately, a number of works (Srivastava et al., 2014; Zou et al., 2019; Rong et al., 2020b; Chen et al., 2021a; Huang et al., 2021) have found that direct graph augmentations with stochastic policies — drop/add edges — can decelerate both the over-fitting and over-smoothing issues in training deep GNNs. By learning the graph's topology and the model's architecture, we naturally adapt our data structure around the model's strength and co-optimize data flow around the message-aggregation mechanism.

The scoring function. Given graph $G(V, E)$ can be expressed in the form of an adjacent matrix $A \in \mathbb{R}^{|V|\times|V|}$, where V is the set of vertices and E is the set of edges. We learn an edge score matrix $S \in \mathbb{R}^{|V|\times|V|}$ such that we rewrite the aggregation step in Eqn (2) as:

$$H^{(k)} = (S \odot A)X^{(k-1)}W^{(k)} \tag{3}$$

where $H^{(k)} = \{h_i^{(k)} : 0 \leq i \leq |V|\}$ and $X^{(k-1)} = \{x_i^{(k-1)} : 0 \leq i \leq |V|\}$. We formally denote $S \in [0, 1]$ as:

$$\begin{aligned} S &= \sigma(\mathrm{MLP}(Z_{V_{\mathrm{src}}} || Z_{V_{\mathrm{tgt}}})) \\ Z &= f(X, G(V, E); W_s) \end{aligned} \tag{4}$$

Where $f(.;W_s)$ is simply the classic VGAE model by Kipf and Welling (2016b), $Z_{V_{src}}/Z_{V_{tgt}}$ are the source and target nodes available from $G(V,E)$, $||$ denotes the concatenation function, and σ is the sigmoid function. S is therefore a sparse matrix with only $|E|$ number of scores.

Edge growing and pruning. Taking advantage of the Progressive NAS workflow Chen et al. (2019c), at each searching stage, we prune the bottom p-percentile from S, and at the same-time we grow our graph by appending k new edges for each nodes via embedding similarity. This embedding similarity function is best defined as:

$$\text{Sim}(v_i, v_j) = \frac{1 + \text{Cosine}(z_i, z_j)}{\log D_{ij}} \tag{5}$$

Where $Cosine(.)$ is the cosine-similarity between two nodes' scoring embeddings, while D is the shortest distance between them. We illustrate visually in Figure(2) and in code via Algorithm(1) in the our appendix.

3.2 Optimization Formulation and Algorithm

3.2.1 **A Principled Bi-Level Optimization Formulation**. For the sake of conciseness, we use α as the *model space architecture parameters*, and denote $\mathbf{L}_{\text{obj}}$ as the objective loss function given α. With α defined, we further denote $\hat{W} = W \bigodot m_\alpha$ as the pruned sub-model from the supernet derived from α description, where $\hat{W}, m_\alpha \in \mathbf{R}^{L\times D\times D}$, D denotes the size of hidden embeddings. Additionally, we can write our augmented graph $\hat{G}$ as $\hat{A} = A \bigodot S$, where S is defined as our learned scoring matrix. Then, let Z represents our output vector for a hypothetical 2-layer **AutoCoG**:

$$Z = Softmax((\hat{A}\sigma(\hat{A}X\hat{W}^{(0)})\hat{W}^{(1)})) \tag{6}$$

Thus the objective loss function $\mathbf{L}_{\text{obj}}$ for a transductive semi-supervised node classification tasks is formally denoted as:

$$\mathbf{L}_{\text{obj}}(\hat{G}, \hat{W}, X, Y) = -\frac{1}{|Y|}\sum_{y_i \in Y} y_i \log(z_i) \tag{7}$$

Extending from (Dong and Yang, 2019), we formulate our data-model co-search as a **joint bi-level optimization**, to solve α, S concurrently with the weights W and data space parameters:

$$\begin{aligned} &\min_{\alpha} \mathbf{L}_{\text{obj}}^{\text{valid}}(\hat{W}(W,\alpha), \hat{G}(S), X_{valid}, Y_{valid}) \\ &\text{s.t.} \quad \hat{W}, \hat{G} = \arg\min_{W,S} \mathbf{L}_{\text{obj}}^{\text{train}}(W, G, \alpha, S, X_{train}, Y_{train}) \end{aligned} \tag{8}$$

Note that α are optimized using the objective loss function on the validation set, while W, S are optimized under training set. Additionally, $\hat{G}$ also consists of modified edges $\bar{E}$, not-shown explicitly in Equation (8), but is illustrated in our Algorithm(1). We adopt the same hard-*Gumbel-softmax* trick (Jang et al., 2017) to differentially optimize architectural variables during search.

3.2.2 **Scaling and Stabilizing the Search**. Thus, the bi-level optimization (8) can be solved by differential search methods, and we adopt the GDAS approach in (Dong and Yang, 2019) by default. However, when exploring GCN deep architectures and larger graphs, the data/model search spaces grow exponentially with the layer depth/graph size, and they can be entangled to cause even more serious scalability challenge. That is further amplified by the training difficulty and instability of deep GNNs (Chen et al., 2021b). Indeed, we observe that naively applying GDAS is prone to over-smoothening and search collapse, only yielding very poor architectures when searching for more than three layers. Besides, it is not uncommon for the derived graph and model to have considerable performance variations across repeated experiments, due the stochastic initialization and training.

Progressive search space. We follow the idea proposed by (Chen et al., 2019c) (also illustrated in Algorithm 1), to divide search into N progressive stages, with each consecutive stage having a larger or equal number of layers than those previously. At each stage, we greedily remove the least selected options (by taking the mean of Soft-Max across L layers and removing the option with the smallest value) from the data's p parameters or model space, and pass on the shrunk co-search space to the next stage. Note that we do not shrink the number of augmentation policies.

4 Experiments

4.1 Experimental settings.

We will list our default training hype-parameters common across all datasets, then we will note each dataset's specific particularity, if any. By default, we employ the Adam-optimizer (Kingma and Ba, 2017) to learn edges' scores, model's architecture and model's weights with equal learning rate of 0.005, and a L_2 regularization of 0.0005. We set the hidden-dimension D to be 256 with a dropout rate of 0.6. As for P-DARTS, for every stage we prune the bottom 10% of edges, and add one new edge per node, and the number of stages are set to be 4, starting from 2 layers, and with a 2 layers increment. Furthermore, for Identity-Mapping, γ is set to be 0.5. We search/train for 1000 epochs, while setting our rate of patient to be 400 and 200 respectively. To get our final results, we train the network 10 times to get the average and standard deviation.

The only notable exception to the default settings are the Co-author datasets, where we set the dropout rate to be 0.8 and 0 for CS and Physics respectively. We typically only search between two and eight layers. Finally, for all datasets, we average their accuracy over 10 runs, with random seed between 0 to 9.

4.2 Ablation studies

Table 2: Ablation results comparing the test results between different searching modes at increasing degree of homophily with fixed depth of eight. Best results are bold.

Experiments	Actor	Texas	Wisconsin	Cornell	CS	Photo
$\mathcal{H}$	0.375	0.411	0.488	0.567	0.827	0.833
Co-search	**38.039±0.16**	**78.378±2.21**	**80.392±0.00**	**64.864±2.97**	**91.840±0.60**	**83.204±2.42**
Data-only	23.924±1.86	64.324±1.71	46.666±2.41	46.486±1.13	80.225±2.12	82.255±2.72
Model-only	36.394±0.07	72.070±0.85	70.588±1.60	56.216±1.14	88.599±0.86	62.798±5.51

Model-Graph codependancy. We justify the need for model-graph co-search by performing three experiments, namely — co-search, data-search, and model-search — to illustrate the respective effectiveness of the individual components which constitutes our framework. We collected these results from several datasets at a fixed depth of 8 while maintaining identical searching settings for all experiments. Note that for data-search, we substitute our model with the vanilla GCN (Kipf and Welling, 2016a). The results are collected in Table(2). Herein our results speak for themselves; we observe a significant improvement in performance, especially for graphs under heterophily, utilizing co-search over model-only and data-only search.

Correlation between depth and performance. We observe that an increasing depth does not always positively correlate to performance gain. Homophily negatively correlates to our performance at depth. To explain this phenomenon, we offer this hypothesis: since the number of layers in a model correlates to the number of k-hop neighbors observed, graphs under heterophily need to observe a much larger sub-graph to aggregate meaning information against the inherent noisy neighbors.

In contrast, with an increasing degree of homophily, more layers may induce over-smoothing sooner, contributing to an overall degradation in performance. We investigate the relationship between the model's depth and performance. As depth is a hyper-parameter, we perform a search on 2, 4, 8, and 16 layers configurations — while maintaining identical searching parameters — on several datasets at an increasing rate of homophily. We then normalize our final accuracy results for each graph against the result of our 2-layer configuration to obtain relative performance gain in percentage. We illustrate our results in Figure(3).

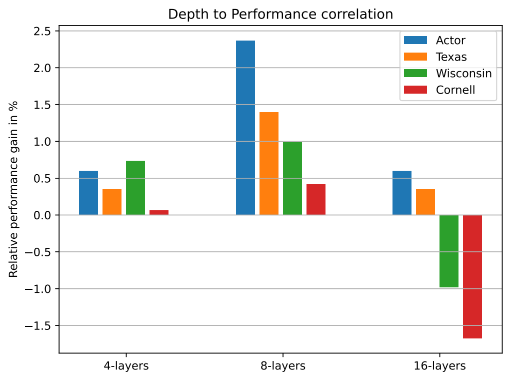

Figure 3: Illustration of relative performance against 2-layer configuration (not shown).

Analyzing the augmented graphs. We characterize the newly augmented graphs to understand better how they affect the overall performance by analyzing those searched under the eight-layers configuration. We first calculate the difference of the homophily rate between the new and old graphs. Next, we count the difference of total informative edges, i.e., edges between nodes of similar classes. Finally, we calculate the Intersection between edges of the original to the augmented graph. Table(3) summarizes our findings. Herein we observe that our improved graphs do not exhibit a stronger rate of homophily — in comparison to the original graphs — as initially assumed. However, we also observe that, despite the increasing heterophily, the searched graphs include more informative edges while retaining most of the original edges, indicating we are learning new and relevant unseen relationships. Nevertheless, this observation challenges the current research assumption on the correlation between heterophily and performance. We show that performance can still be achieved under low homophily given that enough informative edges are added to the graph and a deeper architecture.

Table 3: We characterize the new graph structured found after search. From left-to-right, ∇H represents the change of homophily rate, $\nabla|\hat{E}|$ denotes the change in informative edges, and IoU describes the overlap between the original edges with the current edges

dataset	$\nabla\mathcal{H}$	$\nabla\vert\hat{E}\vert$	Intersection
Actor	↓ 0.056	↑ 4654	99.72%
Texas	↑ 0.018	↑ 260	94.31%
Wisconsin	↓ 0.059	↑ 263	96.93%
Cornell	↓ 0.162	↑ 161	89.54%

Analyzing the method's effectiveness and efficiency. To evaluate the efficiency of our design, we compare its memory usage and total run time to other NAS-based approaches such as GraphNAS (Gao et al., 2019) and SANE (Zhao et al., 2021). Table(5) summarizes our findings. We could observe: that AutoCoG maintains relatively low memory utilization for each of the datasets tested, and AutoCoG is also the fastest model to complete both its full-search and training stages.

Table 4: We characterize the efficiency and effectiveness of our search method by measuring the memory usage and total run time to search and fully train a model for various datasets.

	Actor		Texas		Wisconsin		Cornell	
Model	GPU(MiB)	Run-time(s)	GPU(MiB)	Run-time(s)	GPU(MiB)	Run-time(s)	GPU(MiB)	Run-time(s)
SANE	3890	3788	1070	2686	998	2194	994	3024
GraphNAS	1088	4320	1268	5161	1326	5262	972	4642
Auto-CoG	2634	960	992	427	994	450	984	360

Analyzing the effectiveness of progressive search. To test the benefit of progressive search, we perform three ablation studies on Texas, Wisconsin, and Cornell at various depths. As shown in the following tables, note that, with progressive search, accuracy is positively correlated to a model's depth, while the opposite is true when searching without it. This is due to the search instability from the resulting search space size, and a deeper network only further exacerbates the problem. Indeed, we further observe that simply applying GDAS, as in the case of searching without progressive search, only yields poor architectures. The results align with our reasoning in section 3.2.2.

Table 5: We study the effectiveness of progressive search (PS) by comparing **Auto-CoG**'s performance at various depth search with and without it.

	Texas			Wisconsin			Cornell		
Mode	2	4	8	2	4	8	2	4	8
With PS	77.30	77.57	80.27	79.96	80.20	80.39	64.86	64.32	64.59
Without PS	76.76	73.24	64.86	77.84	63.92	68.23	60.27	59.2	61.08

Analyzing graph-model co-search improvement on model's robustness. First, taking the original graphs, we added random noisy edges to the percentage amount, with respect to the total edges, specified in each column. Next, we performed two searches from these noisy graphs, one without graph co-search and one with. Table (6) summarizes our finding. Here we make the following observations: First, noisy graph data leads to poor model performance from lack of robustness. Second Graph-search can rectify and improve the model's robustness by removing artificial noise, correspondingly leading to better model performance.

Table 6: We analyze the improved robustness provided by graph-model co-search. First taking the original graphs, we added random noisy edges to the percentage amount, with respect to the total edges, specified in each column.

	Texas			Wisconsin			Cornell		
Mode	20%	40%	80%	20%	40%	80%	20%	40%	80%
Model Only	74.59 ± 1.39	73.78 ± 1.30	71.35 ± 2.28	65.88 ± 2.11	72.35 ± 0.62	71.35 ± 2.28	54.59 ± 1.13	61.24 ± 2.61	62.70 ± 1.13
Co-Search	80.81 ± 0.85	79.73 ± 1.91	77.29 ± 1.39	80.0 ± 1.24	81.56 ± 1.01	81.96 ± 0.83	66.22 ± 2.29	63.24 ± 1.88	65.40 ± 1.70

4.3 Results

Table 7: Test Accuracy (%) comparison with other previous state-of-the-art frameworks. Experiments are conducted on the WebKB, Coauthor, Amazon, and Actor datasets. To highlight only the model's performance, we select the best accuracy from each model among different depths between two to eight layers for each dataset. (*) best result. (**) second best result.

Model	Actor	Texas	Wisconsin	Cornell	Computer	CS	Photos	Physics	Avg improv.	Avg Rank
SGC	26.17±1.15	56.41±4.25	51.29±6.44	58.57±3.44	37.53±0.20	70.52±3.96	26.60±4.64	91.46±0.48	↑ 24.30	9.87
GCN	28.82±0.13	65.95±2.76	57.84±1.81	54.05±0.00	81.62±2.11**	91.83±0.50**	79.76±3.14	93.68±0.22	↑ 7.43	5.50
GAT	28.24±0.36	62.16±1.21	52.55±1.92	53.78±1.46	77.74±2.02	89.27±0.46	74.56±3.02	93.19±0.36	↑ 10.18	8.13
GCNII	34.28±1.12**	69.19±6.56	70.31±4.75	61.08±2.76	37.56±0.43	71.67±2.68	62.95±9.41	93.15±0.92	↑ 14.10	5.75
JKNet	28.80±0.97	61.08±6.23	52.76±5.69	57.30±4.95	67.99±5.07	81.82±3.32	78.42±6.95	90.92±1.61	↑ 11.73	8.00
APPNP	28.65±1.28	60.68±4.50	54.24±5.94	58.43±3.74	43.02±10.16	91.61±0.49	59.62±23.27	93.75±0.61**	↑ 15.74	7.13
Geom-GCN	31.63±0.02	65.94±1.39	68.63±0.00	59.75±1.80	—	—	—	—	↑ 9.40	5.5
H2GCN	33.13±0.10	82.41±0.07*	79.61±1.01**	80.4 ±0.05*	37.48±0.08	28.83±7.95	46.56±0.17	93.90 ±0.05*	↑ 16.33	4.75
GraphNAS	26.87±2.09	78.11±3.91	63.14±5.13	59.73±4.49	84.66±0.22*	90.11±0.31	91.11±0.18*	93.75±0.60	↑ 3.18	4.25
SANE	32.05±1.49	71.89±7.77	60.39±10.57	54.59±11.02	78.99±4.3	88.51±0.65	87.72±1.50	OOM	↑ 6.50	5.20
GASSO	27.02±0.05	64.86±0.00	78.43±0.00	64.70±0.00	OOM	OOM	89.32±0.05**	OOM	↑ 4.88	5.00
Auto-CoG	38.04±0.16*	80.27±2.21**	80.39±0.00*	64.86±2.97**	78.91±2.57	92.05±0.40*	85.16±1.12	93.28±0.58	–	2.50

We compare **Auto-CoG** to several notable state-of-the-methods inferencing on graphs with increasing degrees of associativity from 0.3 and 0.9. Additionally, we also include *average improve-*

ment and *average rank* for quick performance comparison at a glance. *Average improvement* is the average accuracy difference between *Auto-CoG* and another model across all datasets, so a higher score indicates a better result. *Average rank* is a model's average performance rank for all datasets, so lower is better. For comparison, we include:

- NAS based graph models: for this category, we include GraphNAS(Gao et al., 2019), SANE(Zhao et al., 2021) and GASSO(Qin et al., 2021).
- Handcrafted graph models: we compare against traditional designs such as GCN (Kipf and Welling, 2016a), SGC (Wu et al., 2019b), GAT (Veličković et al., 2018), GCNII Chen et al. (2020a), JKNet (Xu et al., 2018) and APPNP (Klicpera et al., 2018). Additionally, we also compare against designs that are crafted specifically for disassortative graphs such as Geom-GCN (Pei et al., 2020c) and H2GCN (Zhu et al., 2020b).

We summarizes our finding in Table(7). From the results, we make the following observations:

- Highlighting the challenge of heterophily, we observe the lack of a dominant approach that can outperform all datasets. However, when we compare their average ranking overall, we do find **Auto-CoG** ranks highly at 2.5 and able to improve against all other approaches on average. This showcase our method's robustness in dealing with graphs under different homophily settings
- In comparison to other NAS approaches, **Auto-CoG** reliably outperforms all of them when it comes to disassociative datasets since typical GNNs tend to over-smooth on noisy graph data — an inherent problem for message-passing. **Auto-CoG** directly modifies its graph data and network's architecture to overcome this weakness. GASSO (Qin et al., 2021) also performs graph structure search, but it is limited to only learning existing edges attention coefficients and therefore is still susceptible to some degree of over-smoothing.
- In comparison to handcrafted baselines, **Auto-CoG** comfortably outperforms Geom-GCN (Pei et al., 2020c) on Actor and WebKB datasets. Our graph-structured learning process provides a similar function as the "structural neighborhood" concept, which Geom-GCN utilizes for bi-level aggregation. On the other hand, H2GCN (Pei et al., 2020c) shows impressive performance on small Webkb datasets, outperforming *Auto-CoG* in both Texas and Cornell. However, the model's 'ego-embeddings' concept does not scale well on larger datasets such as CoAuthor and Amazon, where it repeatedly fails to produce competitive results.

5 Conclusion and Limitations

In this paper, we present **AutoCoG** the first NAS framework towards **unified data-model co-search for GNNs**. Our results convincingly demonstrate the benefit of data-graph co-search for both deep and shallow graph neural networks. Our ablation study shows that controlled variances in graph heterophily can result in a better, more generalized model and the necessity for graph-augmentation to be model-aware. We confidently demonstrate **AutoCoG** to be a reliable way to discover robust architectures, a stable model training environment, and state-of-the-art results. Additionally, we show that the localized disturbance of graph structure motivates node position learning, allowing for greater generalizability of the model.

However, there are still limitations that need to be addressed: large graph scalability and understanding heterophily's relationship to performance. To address this, we first plan to follow up by learning meaningful model/graph using **Auto-CoG** via graph-batching. Secondly, we want to conduct a study to understand better the phenomenon between heterophily and performance observed in our ablation. There is no negative societal impact to our best knowledge, except that the NAS search process is resource consuming - but even that excessive cost can be amortized by the re-usability of the searched model, which can achieve superior accuracy-resource trade-off.

6 Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We claim the benefit of model-graph co-search and prove it with our superior performance in most datasets tested.

   (b) Did you describe the limitations of your work? [Yes] We describe our two limitations in section 5.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We briefly discuss it in section 5.

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] We provide pseudo-code, experimental settings, and model's architecture in our supplementary. Full code will be release upon acceptance.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes]

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] We detailed our hyperparameters in section 4.1

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] see section 4.2

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [No]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] See section 4.1

   (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All our results are averaged over 10 times for seed between 0 and 9

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] This is not within-the-scope of our research.

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] we went with default hyper-parameters.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cited datasets used in section 4.1

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] the benchmarks we used contained no personally identifiable information or offensive content

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. (2019). Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing.

Alon, U. and Yahav, E. (2020). On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*.

Antoniou, A., Storkey, A., and Edwards, H. (2017). Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*.

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv*.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv*.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2019a). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*.

Chen, D., Liu, X., Lin, Y., Li, P., Zhou, J., Su, Q., and Sun, X. (2019b). Highwaygraph: Modelling long-distance node relations for improving general graph neural network.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020a). Simple and deep graph convolutional networks.

Chen, T., Sui, Y., Chen, X., Zhang, A., and Wang, Z. (2021a). A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pages 1695–1706. PMLR.

Chen, T., Zhou, K., Duan, K., Zheng, W., Wang, P., Hu, X., and Wang, Z. (2021b). Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *arXiv preprint arXiv:2108.10521*.

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019c). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303.

Chen, Y., Wu, L., and Zaki, M. (2020b). Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33.

Chien, E., Peng, J., Li, P., and Milenkovic, O. (2021). Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations. https://openreview.net/forum*.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (1998). Learning to extract symbolic knowledge from the world wide web.

Ding, Y., Yao, Q., and Zhang, T. (2020). Propagation model search for graph neural networks. *arXiv preprint arXiv:2010.03250*.

Dong, X. and Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours.

Feng, S. Y., Gangal, V., Wei, J., Chandar, S., Vosoughi, S., Mitamura, T., and Hovy, E. (2021). A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*.

Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., and Tang, J. (2020). Graph random neural networks for semi-supervised learning on graphs. *Advances in Neural Information Processing Systems*, 33.

Gao, Y., Yang, H., Zhang, P., Zhou, C., and Hu, Y. (2019). Graphnas: Graph neural architecture search with reinforcement learning.

Gong, X., Chang, S., Jiang, Y., and Wang, Z. (2019). Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3224–3234.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *NeuIPS*, pages 1024–1034.

Hamilton, W. L., Ying, R., and Leskovec, J. (2018). Inductive representation learning on large graphs.

Huang, W., Rong, Y., Xu, T., Sun, F., and Huang, J. (2020). Tackling over-smoothing for general graph convolutional networks. *arXiv e-prints*, pages arXiv–2008.

Huang, W., Rong, Y., Xu, T., Sun, F., and Huang, J. (2021). Tackling over-smoothing for general graph convolutional networks.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907.*

Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders.

Klicpera, J., Bojchevski, A., and Günnemann, S. (2018). Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997.

Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

Li, Y. and King, I. (2020). Autograph: Automated graph neural network. In *International Conference on Neural Information Processing*, pages 189–201. Springer.

Lim, D., Hohne, F., Li, X., Huang, S. L., Gupta, V., Bhalerao, O., and Lim, S.-N. (2021). Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv.*

Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv.*

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2019). Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier.

NT, H. and Maehara, T. (2019). Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550.*

Oono, K. and Suzuki, T. (2020). Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations.*

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020a). Geom-gcn: Geometric graph convolutional networks.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020b). Geom-gcn: Geometric graph convolutional networks. In *ICLR.*

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020c). Geom-gcn: Geometric graph convolutional networks.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. *arXiv*.

Qin, Y., Wang, X., Zhang, Z., and Zhu, W. (2021). Graph differentiable architecture search with structure learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16860–16872. Curran Associates, Inc.

Rong, Y., Huang, W., Xu, T., and Huang, J. (2020a). Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations. https://openreview. net/forum.*

Rong, Y., Huang, W., Xu, T., and Huang, J. (2020b). Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations.*

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2019). Pitfalls of graph neural network evaluation.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Tang, J., Sun, J., Wang, C., and Yang, Z. (2009). Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 807–816, New York, NY, USA. Association for Computing Machinery.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations.*

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks.

Wang, H. and Huan, J. (2019). Agan: Towards automated design of generative adversarial networks. *arXiv.*

Wu, F., Zhang, T., de Souza Jr. au2, A. H., Fifty, C., Yu, T., and Weinberger, K. Q. (2019a). Simplifying graph convolutional networks.

Wu, F., Zhang, T., Jr., A. H. S., Fifty, C., Yu, T., and Weinberger, K. Q. (2019b). Simplifying graph convolutional networks. *CoRR*, abs/1902.07153.

Xie, L. and Yuille, A. (2017). Genetic cnn. In *ICCV*, pages 1379–1388.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations.*

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. (2018). Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536.

You, J., Ying, Z., and Leskovec, J. (2020a). Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. (2020b). Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33.

Zela, A., Siems, J., and Hutter, F. (2020). Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422.*

Zhang, H., Yan, T., Xie, Z., Xia, Y., and Zhang, Y. (2020). Revisiting graph convolutional network on semi-supervised node classification from an optimization perspective.

Zhao, H., Wei, L., and Yao, Q. (2020a). Simplifying architecture search for graph neural network. In *International Conference on Information and Knowledge Management.*

Zhao, H., Yao, Q., and Tu, W. (2021). Search to aggregate neighborhood for graph neural network.

Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., and Shah, N. (2020b). Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830.*

Zhou, K., Song, Q., Huang, X., and Hu, X. (2019a). Auto-gnn: Neural architecture search of graph neural networks.

Zhou, K., Song, Q., Huang, X., and Hu, X. (2019b). Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184.*

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. (2020a). Beyond homophily in graph neural networks: Current limitations and effective designs.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. (2020b). Beyond homophily in graph neural networks: Current limitations and effective designs.

Zitnik, M. and Leskovec, J. (2017). Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. (2019). Layer-dependent importance sampling for training deep and large graph convolutional networks.

Zügner, D., Akbarnejad, A., and Günnemann, S. (2018). Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856.

A Appendix

A.1 Datasets

We evaluate **AutoCoG** on several popular semi-node classification datasets including (i) the WebKB datasets —Cornell, Texas, Wisconsin— (Craven et al., 1998), (ii) Actor dataset (Tang et al., 2009), (iii) Co-author datasets —CS, Physics— (Shchur et al., 2019), and (iv) Amazon datasets — Photo, Computers — (Shchur et al., 2019). We select these datasets to represent a wide range of graphs under different degrees of homophily, which will serve to demonstrate **Auto-CoG** robustness in comparison to the different SOTA methods. The specifics on each datasets, in sorted order under homophily, is recorded in Table(8).

h!

Table 8: The Statistics of each dataset. From left to right: unique classes, nodes, edges and embedding dimension and edge-homophily degree.

dataset	$\|Y\|$	$\|V\|$	$\|E\|$	$\|D\|$	$\mathcal{H}$
Actor	5	7,600	33,544	931	0.375
Texas	5	183	295	1,703	0.411
Wisconsin	5	251	499	1703	0.488
Cornell	5	183	295	1,703	0.567
Computers	10	13,752	491,722	767	0.783
CS	40	18,333	163,788	6,805	0.827
Photos	10	7,650	238,162	745	0.833
Physics	5	34,493	495,924	8,415	0.936

A.2 Algorithm

Algorithm 1: AutoCoG Searching Algorithm

```
Input: $W_s, X, G(V,E)$, searchSpace, epochs, startNumLayer, endNumLayer, stages ;
Output: $\alpha, G(V,\bar{E}), S$;
$\bar{E} \leftarrow E$ ;
for $s$ = 0 to stages-1 do
    #Initialize new model and architecture parameters
    $\alpha \leftarrow$ OnesInitParameters(searchSpace);
    $W \leftarrow$ initModel(min(startNumLayer+s, endNumLayer));
    for e=0 to epochs-1 do
        $S \leftarrow \sigma(\mathrm{MLP}(f(X, G(V,\bar{E}); W_s))$;
        $\bar{a} \leftarrow$ Sample($\alpha$);
        BackPropgate $L_{obj}(\bar{\alpha}, W, S, X_{train}, G(V,\bar{E})) \longrightarrow W, W_s$;
        BackPropgate $L_{obj}(\bar{\alpha}, W, S, X_{valid}, G(V,\bar{E})) \longrightarrow \alpha$;
    end
    #Reduce search space and augment edges
    $\bar{E} \leftarrow$ PruneAndGrow($f(.; W_s)$);
    searchSpace $\leftarrow$ ReduceSearchSpace(searchSpace, $\alpha$);
end
$S \leftarrow \sigma(\mathrm{MLP}(f(X, G(V,\bar{E}); W_s))$;
```