EFFICIENT GRAPH NEURAL ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently graph Neural Networks (GNN) have been demonstrated effective in various graph-based tasks. To obtain state-of-the-art (SOAT) data-specific GNN architectures, researchers turn to neural architecture search (NAS) methods. However, it remains to be a challenging problem to conduct efficient architecture search for GNN. In this work, we present a novel framework for Efficient GrAph Neural architecture search (EGAN). By designing a novel and expressive search space, an efficient one-shot NAS method based on stochastic relaxation and natural gradient is proposed. Further, to enable architecture search in large graphs, a transfer learning paradigm is designed. Extensive experiments, including node-level and graph-level classification tasks, are conducted, whose results show that the proposed EGAN can obtain SOTA architectures compared to human-designed ones, and be more efficient in orders than existing NAS baselines.

1 INTRODUCTION

Recent years have witnessed the success of graph neural networks (GNN) (Gori et al., 2005; Battaglia et al., 2018) in various graph-based tasks, e.g., recommendation (Ying et al., 2018; Xiao et al., 2019), fraud detection (Liu et al., 2019b), chemistry (Gilmer et al., 2017). To adapt to different graph-based tasks, various GNN models (Kipf & Welling, 2016; Hamilton et al., 2017; Gilmer et al., 2017; Veličković et al., 2018; Gao et al., 2018; Xu et al., 2019; 2018; Liu et al., 2019b; Pei et al., 2020; Rong et al., 2020; Zeng et al., 2020; Zhao & Akoglu, 2020) have been designed in the past five years. Most of existing GNN models follow a neighborhood aggregation (or *message passing*) schema (Gilmer et al., 2017), as shown in the left part of Figure 1, which is that the representation of a node in a graph is learnt by iteratively aggregating the features of its neighbors. By stacking GNN layers, multi-hop, the so-called *higher-order*, neighbors, can be utilized to enhance the performance.

Despite the broad applications of GNN models, researchers have to take tedious efforts to design proper GNN architectures given different tasks by imposing different relational biases (Battaglia et al., 2018). As pointed out by Battaglia et al. (2018), the GNN architectures can support one form of *combinatorial generalization* given different tasks, i.e., graphs. Then a natural and interesting question can be asked: *Can we automatically design state-of-the-art (SOTA) GNN architectures, thus provide better combinatorial generalization for graph-based tasks?* A straightforward solution is to adopt the neural architecture search (NAS) approaches, which have shown impressive results in automatically designing architectures for convolutional neural networks (CNN) and recurrent neural networks (RNN) (Zoph & Le, 2017; Pham et al., 2018; Cai et al., 2019; Bender et al., 2018; Liu et al., 2019a; Xie et al., 2019; Tan & Le, 2019; You et al., 2020).

However, it is nontrivial to adopt NAS to GNN architecture search. The first challenge is to define the search space. A *dummy* search space can include all parameters, e.g., aggregation functions, hidden embedding size, etc., related to the message passing framework (Eq. (1)), and obviously it leads to a huge space, which is difficult to obtain a SOTA architecture by any search algorithm. The second challenge is to design an efficient and effective search algorithm. In the literature, reinforcement learning (RL) based and evolutionary based algorithms have been explored for GNN architecture search (Gao et al., 2020; Zhou et al., 2019b; Lai et al., 2020; Nunes & Pappa, 2020). However, they are inherently computationally expensive due to the trial-and-error manner. In the NAS literature, by adopting the weight sharing strategy, one-shot methods NAS methods have been shown to be in orders more efficient than RL based ones(Pham et al., 2018; Liu et al., 2019a; Xie et al., 2019; Akimoto et al., 2019; Guo et al., 2019; Zela et al., 2020). However, the one-shot methods cannot

be directly to applied to the aforementioned dummy search space. Therefore, it remains to be a challenging problem to conduct efficient and effective architecture search for GNN.

In this work, we propose a novel framework, called EGAN (Efficient GrAph Neural architecture search), to automatically design SOTA GNN architectures. Motivated by two well-established works (Xu et al., 2019; Garg et al., 2020) that the expressive capabilities of GNN models are highly relying on the properties of the aggregation functions, a novel search space consisting of node and layer aggregators is designed, which can emulate many popular GNN models. Then by representing the search space as a directed acyclic graph (DAG) (Figure 1(c)), we design a one-shot framework by using the stochastic relaxation and natural gradient method, which can optimize the architecture selection and model parameters in an end-to-end manner. To enable architecture search in large graphs, we further design a transfer learning paradigm, which constructs a proxy graph out of the large graph by keeping the properties, and then searched SOTA GNN architectures in the proxy graph, finally the searched architecture is transferred to the large graph. To demonstrate the efficiency and effectiveness of the proposed framework, we apply EGAN to various tasks, from node-level to graph-level ones. The experimental results on eleven different datasets shows that EGAN can obtain SOTA architectures for different tasks, and the transfer learning paradigms make it possible to conduct architecture search in large graphs. [1] [2] [3]

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple graph with node features $\mathbf{X} \in \mathbb{R}^{N \times d}$, where $\mathcal{V}$ and $\mathcal{E}$ represent the node and edge sets, respectively. N represents the number of nodes and d is the dimension of node features. We use $N(v)$ to represent the first-order neighbors of a node v in $\mathcal{G}$, i.e., $N(v) = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$. In the literature, we also create a new set $\widetilde{N}(v)$ is the neighbor set including itself, i.e., $\widetilde{N}(v) = \{v\} \cup \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$.

2 Related Works

GNN was first proposed in Gori et al. (2005) and in the past five years many different variants (Kipf & Welling, 2016; Hamilton et al., 2017; Veličković et al., 2018; Gao et al., 2018; Battaglia et al., 2018; Xu et al., 2019; 2018; Liu et al., 2019b; Abu-El-Haija et al., 2019; Wu et al., 2019; Zeng et al., 2020; Zhao & Akoglu, 2020; Rong et al., 2020) have been designed, all of which are relying on a neighborhood aggregation (or *message passing*) schema (Gilmer et al., 2017), which learns the representation of a given node by iteratively aggregating the hidden features ("message") of its neighbors. Besides, in Xu et al. (2018); Chen et al. (2020), the design of residual networks are incorporated in existing message passing GNN models.

Neural architecture search (NAS) (Baker et al., 2017; Zoph & Le, 2017; Elsken et al., 2018; Yao & Wang, 2018) aims to automatically find better and smaller architectures comparing to expert-designed ones, which have shown promising results in architecture design for CNN and Recurrent Neural Network (RNN) (Liu et al., 2019a; Zoph et al., 2018; Tan & Le, 2019). In the literature, one of the representative NAS approaches are reinforcement learning (RL) (Baker et al., 2017; Zoph & Le, 2017; Pham et al., 2018), which trains an RNN controller in the loop: the controller firstly generates an candidate architecture by sampling a list of actions (operations) from a pre-defined search space, and then trains it to convergence to obtain the performance of the given task. The controller then uses the performance as the guiding signal to update the RNN parameters, and the whole process is repeated for many iterations to find more promising architectures. To address the efficiency problem of RL based methods, a series of methods adopt weight sharing strategy to reduce the computational cost. To be specific, instead of training one by one thousands of separate models from scratch, one can train a single large network (*supernet*) capable of emulating any architecture in the search space. Then each architecture can inherit the weights from the supernet, and the best architecture can be obtained more efficiently. This paradigm is also referred to as "one-shot NAS", and representative methods are (Pham et al., 2018; Bender et al., 2018; Liu et al., 2019a; Cai et al., 2019; Yao et al., 2020; Xie et al., 2019; Zhou et al., 2019a; Akimoto et al., 2019).

[1] +qm+ not good to say this. it is very easy for reviewers to ask you other architectures that you do not include. a better way is to say layer agg, which is important to extract the whole graph information is not considered. This is a "structural limitation".✓✓

[2] +qm+ you can use whole graph task to high-light such a limitation.

[3] +qm+ better use list form.✓✓In this version, the list form can break the consistence of reading experience, let us keep it like this currently.

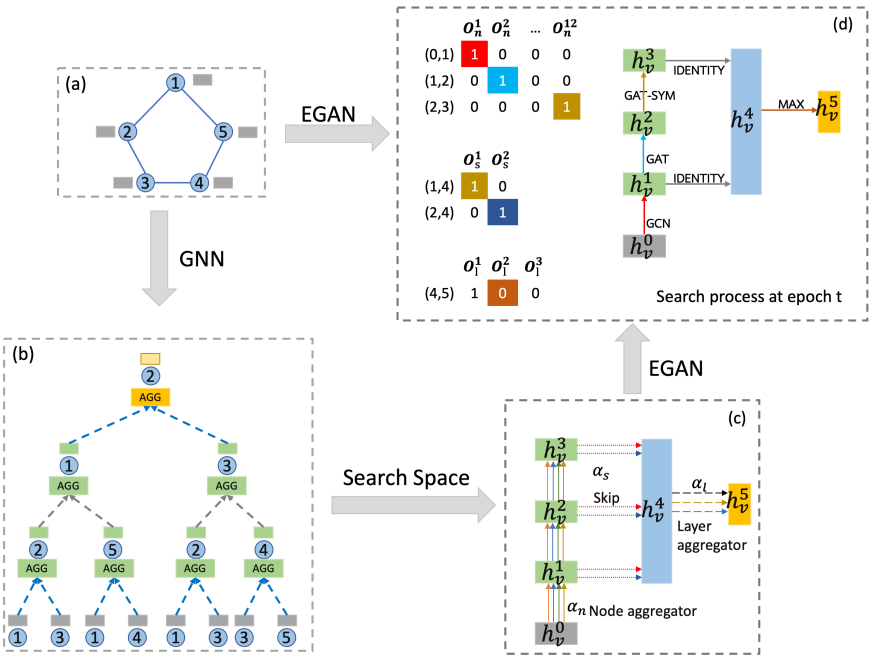


Figure 1: An illustrative example of a GNN model and the search space of the propsed EGAN (Best view in color). (a) An example graph with five nodes. The gray rectangle represents the input features of each node; (b) A typical 3-layer GNN model following the message passing neighborhood aggregation schema, which computes the embeddings of node "2"; (c) The DAG represents the search space for a 3-layer GNN, and $\boldsymbol{\alpha}_n, \boldsymbol{\alpha}_l, \boldsymbol{\alpha}_s$ represent, respectively, weight vectors for node aggregators, layer aggregators, and skip-connections in the corresponding edges. The rectangles denote the representations, out of which three green ones represent the hidden embeddings, gray ($\mathbf{h}_v^0$) and yellow ($\mathbf{h}_v^5$) ones represent the input and output embeddings, respectively, and blue one ($\mathbf{h}_v^4$) represent the set of output embeddings of three node aggregators for the layer aggregator. (d) At the t-th epoch, an architecture is sampled from $p(\mathbf{Z}_n), p(\mathbf{Z}_s), p(\mathbf{Z}_l)$, whose rows $\mathbf{Z}_{i,j}$ are one-hot random variable vector indicating masks multipled to edges (i, j) in the DAG. Columns of these matrices represent the operations from $\mathcal{O}_n, \mathcal{O}_s, \mathcal{O}_l$.

Recently, there are several works on architecture search for GNN, GraphNAS (Gao et al., 2020), Auto-GNN (Zhou et al., 2019b), and Policy-GNN (Lai et al., 2020), which are all RL-based methods. While GraphNAS and Auto-GNN search for different GNN architectures, Policy-GNN searches for different number of layers for each node in the graph. Here, we focus on searching for the aggregation functions, and propose a one-shot method, which simplifies and accelerates the architecture search for GNN. **Huan:** Add Nunes & Pappa (2020); Jiang & Balaprakash (2020)

3 The Proposed Framework

3.1 The design of search space

As introduced in 2, most existing GNN architectures are relying on a message passing framework (Gilmer et al., 2017), which constitutes the backbone of the designed search space in this work. To be specific, a K-layer GNN can be written as follows: the l-th layer ($l = 1, \cdots, K$) updates $\mathbf{h}_v$ for each node v by aggregating its neighborhood as

$$\mathbf{h}_v^l = \sigma(\mathbf{W}^{(l)} \cdot \Phi_n(\{\mathbf{h}_u^{(l-1)}, \forall u \in \widetilde{N}(v)\})), \tag{1}$$

where $\mathbf{h}_v^{(l)} \in \mathbb{R}^{d_l}$ represents the hidden features of a node v learned by the l-th layer, and d_l is the corresponding dimension. $\mathbf{W}^{(l)}$ is a trainable weight matrix shared by all nodes in the graph,

Table 1: The operations we use as node and layer aggregators for the search space of EGAN. Details of node aggregators are given in Appendix A.1.

	Operations
$\mathcal{O}_n$	SAGE−SUM, SAGE−MEAN, SAGE−MAX, SAGE−LSTM, GCN, GAT,GAT−SYM, GAT−COS, GAT−LINEAR, GAT−GEN−LINEAR, GIN, GeniePath
$\mathcal{O}_l$	CONCAT, MAX, LSTM
$\mathcal{O}_s$	IDENTITY, ZERO

and σ is a non-linear activation function, e.g., a sigmoid or ReLU. Φ_n is the key component, i.e., a pre-defined aggregation function, which varies across on different GNN models.

Thus a dummy search space is to include all relevant parameters in Eq. (1). However, it leads to a too large search space, making the search process more expensive. In this work, motivated by two well-established works (Xu et al., 2019; Garg et al., 2020), which show that the expressive capabilities of GNN models are highly relying on the properties of aggregation functions, we propose to search for different aggregation functions by simplifying the search space. For other hyper-parameters, we do simple tuning in the re-training stage, which is also a standard practice in NAS literature (Liu et al., 2019a; Xie et al., 2019). Then the first component of the proposed search space is the *node aggregators*, which consists of existing message passing models. To further improve the expressive capability, we add the other component, *layer aggregators*, to combine the outputs of all node aggregator layers, which have been demonstrated effectiveness in JK-Network (Xu et al., 2018). Then we introduce the backbone of the proposed search space, as shown in the bottom right part of Figure 1, which consists of two key components:

- **Node aggregators**: We choose 12 node aggregators based on popular GNN models, and they are presented in Table 1. The node aggregator set is denoted by $\mathcal{O}_n$.
- **Layer aggregators**: We choose 3 layer aggregators as shown in Table 1. Besides, we have two more operations, IDENTITY and ZERO, related to skip-connections. Instead of requiring skip-connections between all intermediate layers and the final layer in JK-Network, in this work, we generalize this option by proposing to search for the existence of skip-connection between each intermediate layer and the last layer. To connect, we choose IDENTITY, and ZERO otherwise. The layer aggregator set is denoted by $\mathcal{O}_l$ and the skip operation set by $\mathcal{O}_s$.

From Table 1, we can see that the designed search space can emulate extensive human-designed GNN architectures. The node aggregators focus on how to aggregate the neighborhood features, while the latter one focus on the range of neighborhood to use. To further inject relational bias from existing GNN architectures, when searching for the skip-connections for each GNN layer, we add one more constraint that the last layer should always be used as the final output, thus for a K-layer GNN architecture, we need to search $K-1$ IDENTITY or ZERO for the skip-connection options. [4]

3.2 Differentiable architecture search

Following the NAS literature (Liu et al., 2019a; Xie et al., 2019), we represent the search space by a directed acyclic graph (DAG), as shown in Figure 1(c), where nodes represent embeddings, and edges represent operations between the two end nodes. Then the intermediate nodes are

$$x_j = \sum\nolimits_{i<j} \tilde{\mathbf{O}}_{i,j}(x_i), \tag{2}$$

where $\tilde{\mathbf{O}}_{i,j}$ is the selected operation at edge (i,j). In our work, each edge corresponds to an operation in the search space, and we represent it with a distribution $p_{\boldsymbol{\alpha}}(\mathbf{Z})$. By multiplying each one-hot random variable $\mathbf{Z}_{i,j}$ to each edge (i,j) in the DAG, the intermediate nodes in each child graph are represented by

$$x_j = \sum\nolimits_{i<j} \tilde{\mathbf{O}}_{i,j}(x_i) = \sum\nolimits_{i<j} \mathbf{Z}_{i,j}^T \mathbf{O}_{i,j}(x_i). \tag{3}$$

[4] +qm+ you can merge Figure **??** into (c) of Figure 1.✓✓

Following the setting in Zoph & Le (2017); Gao et al. (2020), the objective of the framework is [5]

$$\mathbb{E}_{\mathbf{Z}\sim p_\alpha(\mathbf{Z})}\big[R(\mathbf{Z})\big] = \mathbb{E}_{\mathbf{Z}\sim p_\alpha(\mathbf{Z})}\big[\mathcal{L}_w(\mathbf{Z})\big], \tag{4}$$

where $R(\mathbf{Z})$ represents the reward, which is defined by training loss $\mathcal{L}_w(\mathbf{Z})$ in our framework. w represents the model parameters.

In Zoph & Le (2017) and Gao et al. (2020), Eq. (4) could be optimized by policy gradient methods (Williams, 1992), however, in this work, to make use of the differentiable nature of $L_\theta(\mathbf{Z})$, we design a differentiable search method to optimize Eq. (4). To be specific, we use the Gumbel softmax (Maddison et al., 2017; Xie et al., 2019; Noy et al., 2020) to relax the discrete architecture distribution to be continuous and differentiable with reparameterization trick:

$$\mathbf{Z}_{i,j}^k = f_{\alpha_{i,j}}(\mathbf{G}_{i,j}^k) = \exp(^{\log \boldsymbol{\alpha}_{i,j}^k + \mathbf{G}_{i,j}^k}/\lambda) / \sum\nolimits_{l=0}^{n} \exp(^{\log \boldsymbol{\alpha}_{i,j}^l + \mathbf{G}_{i,j}^l}/\lambda), \tag{5}$$

where $\mathbf{Z}_{i,j}$ is the softened one-hot random variable for operation selection at edge (i,j), $\mathbf{G}_{i,j}^k = -\log(-log(\mathbf{U}_{i,j}^k))$ is the kth Gumbel random variable, $\mathbf{U}_{i,j}^k$ is a uniform random variable. $\boldsymbol{\alpha}_{i,j}$ is the architecture parameter. λ is the temperature of the softmax, which is steadily annealed to be close to zero (Xie et al., 2019; Noy et al., 2020). In Maddison et al. (2017), it is proved that $p(\lim_{\lambda\to 0} \mathbf{Z}_{i,j}^k = 1) = \boldsymbol{\alpha}_{i,j}^k / (\sum_{l=0}^{n} \boldsymbol{\alpha}_{i,j}^l)$, which means that this relaxation is unbiased once converged.

In the GNN literature, node-level or graph-level classification tasks are commonly used, thus the cross-entropy loss is chosen. Then we can use gradient descent methods to optimize the operation parameters and model parameters together in an end-to-end manner. The gradients are given in Appendix A.2.

To improve the search speed and robustness of the performance, we further design an adaptive stochastic natural gradient method to update the architecture parameters in an end-to-end manner following Akimoto et al. (2019). To be specific, the update of $\boldsymbol{\alpha}$ at the mth iteration is given as:

$$\boldsymbol{\alpha}^{m+1} = \boldsymbol{\alpha}^m - \rho \mathbf{H}^{-1} \nabla_{\boldsymbol{\alpha}} \mathcal{L}, \tag{6}$$

where ρ is the step size. $\mathbf{H}^{-1}$ is the Fisher matrix, which can be computed as $\mathbf{H} = \mathbb{E}_{p_{\boldsymbol{\alpha}^m}}[\bar{p}_{\boldsymbol{\alpha}}(\mathbf{Z})\bar{p}_{\boldsymbol{\alpha}}(\mathbf{Z})^T]$ with $\bar{p}_{\boldsymbol{\alpha}}(\mathbf{Z}) := \nabla \log p_{\boldsymbol{\alpha}}(\mathbf{Z})$.

After the searching process terminates, we derive the optimal architecture by retaining the edge with largest weight, which is the same as existing DARTS (Liu et al., 2019a) and SNAS (Xie et al., 2019). To robust the final results, the search process is executed for 5 times with different random seeds, thus 5 architectures are obtained in the end of the search phase. Then the 5 architectures are re-trained from scratch with some hyper-parameters tunning on validation set, and the one with the best validation accuracy is returned as the final architecture.

3.3 Transfer learning paradigm

For large graphs, since NAS approaches can be extremely expensive in terms of search time and GPU memory, we may not want to directly search architectures in them. Motivated by transferring searched blocks and cells in CNN architectures from small to large data sets in the NAS literature Zoph et al. (2018); Tan & Le (2019), we propose to address the above problem by transfer learning **?**. The core idea of the transferable architecture search is to find a small proxy graph $\mathcal{G}_{\text{proxy}}$ (the source), then search in the proxy graph, finally tune the searched architecture $\{\boldsymbol{\alpha}_n, \boldsymbol{\alpha}_s, \boldsymbol{\alpha}_l\}$ in the large graph $\mathcal{G}$ (the target). However, in order to make the architecture transfer feasible, we need to make the proxy graph sharing the same property distribution with the original graph **?**. Since the properties vary across different graphs, it is not suitable to transfer across different datasets, like that from CIFAR-10 to ImageNet for image classification Zoph et al. (2018). Thus, we propose to sample a smaller graph from the original one, and then apply the transfer paradigm. Many distribution-preserving sampling schema have been proposed in the literature, e.g., random sampling by node or edge, proposed in an established work Leskovec & Faloutsos (2006). In this work, we adopt the Random PageRank Node (RPN) sampling method in Leskovec & Faloutsos (2006), which is empirically demonstrated to be able to preserve the properties with a sample size not less than 15% of the original graph.

[5] +qm+ better merge (4) and (5) into one equation; and simply description here✓✓

3.4 DISCUSSION ABOUT RECENT GNN AND WHOLE FRAMEWORK

[6]

Table 2: text

	space		algorithm	task
	node agg	layer agg		
GraphNAS/ Auto-GNN	$\surd$	$\times$	reinforcement learning	node-level
Ours	$\surd$	$\surd$	natural gradient	node/graph-level

Table 3: Dataset statistics of the datasets in the experiments. Here we use "CS", "Computer", "Photo" as the abbreviations for Coauthor CS, Amazon Computers and Amazon Photo, respectively.

	Transductive						Inductive
	Cora	CiteSeer	PubMed	CS	Computer	Photo	PPI
#nodes	2,708	3,327	19,717	18,333	13,752	7,650	56,944
#edges	5,278	4,552	44,324	81,894	245,861	119,081	818,716
#features	1,433	3,703	500	6,805	767	745	121
#classes	7	6	3	15	10	8	50

4 EXPERIMENTS

In this section, we conduct extensive experiments to demonstrate the effectiveness of the proposed EGAN, including node-level and graph-level tasks.

4.1 NODE-LEVEL TASKS

Settings. For node-level tasks, we have three settings: transductive, inductive, and transfer. The former two settings are standard ones in GNN literature, and the latter one is designed by us to demonstrate the efficacy of transfer learning paradigm on EGAN in large graphs. For the former two tasks, four benchmark datasets are used, i.e., Cora, CiterSeer, PubMed, and PPI. Besides, we add three more datasets for transductive task. The statistics of the seven datasets are given in Table **??**. For sake of space, the details of datasets are given in Appendix A.3.

Baselines For baselines in node-level tasks, we compare with two types of methods: existing message passing GNN models, and existing NAS methods. More details can be checked in Appendix A.3.

4.2 PERFORMANCE COMPARISON

In this part, we give the analysis of the performance comparisons on different datasets. The results of transductive and inductive tasks are given in Table 4 and Table 5, respectively. Besides, we visualize the searched architectures on different datasets in Figure 3.

4.2.1 PERFORMANCE COMPARISONS IN TRANSDUCTIVE AND INDUCTIVE TASKS

From Table 4, we can see that SNAG models, including SNAG-WS, win over all baselines on most datasets except CiteSeer. Considering the fact that the performance of SNAG on CiteSeer is very close to the best one (Random), it demonstrates the effectiveness of the NAS methods on GNN. In other words, with SNAG, we can obtain well-behaved GNN architectures given a new task. When comparing SNAG with GraphNAS methods, the performance gain is evident. We attribute this to the

[6] +qm+ Table 2 can be a better choice. 1) how may existing node agg are used is not that important, we can focus on concept level here 2) otherwise, it is easy for reviewers to ask detailed questions for this table.

Table 4: Performance comparisons in transductive tasks. We show the mean classification accuracy (with standard deviation). We categorize baselines into human-designed GNNs and NAS methods. The best results in different groups of baselines are underlined, and the best result on each dataset is in boldface. Note that the performance of LGCN on Coauthor CS, Amazon Photo and Amazon Computer is "-" due to the OOM (Out Of Memory) problem when running the released code in our GPUs.

		Transductive					
	Methods	Cora	CiteSeer	PubMed	CS	Photo	Computer
Human-designed GNN	GCN	0.8761 (0.0101)	0.7666 (0.0202)	0.8708 (0.0030)	0.9345 (0.0047)	0.9401 (0.0067)	0.9116 (0.0074)
	GCN-JK	0.8770 (0.0118)	0.7713 (0.0136)	0.8777 (0.0037)	0.9490 (0.0030)	0.9512 (0.0052)	0.9101 (0.0059)
	GraphSAGE	0.8741 (0.0159)	0.7599 (0.0094)	0.8784 (0.0044)	0.9329 (0.0048)	0.9413 (0.0042)	0.9150 (0.0033)
	GraphSAGE-JK	0.8841 (0.0015)	0.7654 (0.0054)	0.8822 (0.0066)	**0.9516 (0.0022)**	0.9570 (0.0051)	0.9176 (0.0037)
	GAT	0.8719 (0.0163)	0.7518 (0.0145)	0.8573 (0.0066)	0.9291 (0.0033)	0.9468 (0.0046)	0.9171 (0.0057)
	GAT-JK	0.8726 (0.0086)	0.7527 (0.0128)	0.8674 (0.0055)	0.9340 (0.0035)	0.9483 (0.0063)	**0.9184 (0.0045)**
	GIN	0.8600 (0.0083)	0.7340 (0.0139)	0.8799 (0.0046)	0.9213 (0.0045)	0.5880 (0.0771)	0.8672 (0.0412)
	GIN-JK	0.8699 (0.0103)	0.7651 (0.0133)	0.8828 (0.0054)	0.9504 (0.0035)	0.9465 (0.0048)	0.9126 (0.0075)
	GeniePath	0.8670 (0.0123)	0.7594 (0.0137)	0.8796 (0.0039)	0.9262 (0.0044)	0.9344 (0.0042)	0.8606 (0.0081)
	GeniePath-JK	0.8776 (0.0117)	0.7591 (0.0116)	0.8818 (0.0037)	0.9252 (0.0048)	0.9314 (0.0112)	0.8831 (0.0065)
	LGCN	0.8687 (0.0075)	0.7543 (0.0221)	0.8753 (0.0012)	-	-	-
NAS methods	Random	0.8694 (0.0032)	**0.7820 (0.0020)**	0.8888(0.0009)	0.9391 (0.0008)	0.9353 (0.0023)	0.9019 (0.0025)
	Bayesian	0.8580 (0.0027)	0.7650 (0.0021)	0.8842(0.0005)	0.9428 (0.0005)	0.9357 (0.0014)	0.9078 (0.0014)
	GraphNAS	0.8840 (0.0071)	0.7762 (0.0061)	0.8896 (0.0024)	0.9284 (0.0024)	0.9465 (0.0015)	0.8996 (0.0028)
	GraphNAS-WS	0.8808 (0.0101)	0.7613 (0.0156)	0.8842 (0.0103)	0.9312 (0.0016)	0.9293 (0.0007)	0.9064 (0.0019)
ours	SNAS_ASGN	**0.9002 (0.0007)**	0.7587 (0.0002)	**0.8999 (0.0009)**	0.9466 (0.0026)	**0.9620 (0.0004)**	0.9154 (0.0026)

Table 5: Performance comparisons in inductive tasks. We show the Micro-F1 (with standard deviation). We categorize baselines into human-designed GNNs and NAS methods. The best results in different groups of baselines are underlined, and the best result is in boldface.

	Methods	PPI
Human-designed GNN	GCN	0.9333 (0.0019)
	GCN-JK	0.9344 (0.0007)
	GraphSAGE	0.9721 (0.0010)
	GraphSAGE-JK	0.9718 (0.0014)
	GAT	0.9782 (0.0005)
	GAT-JK	0.9781 (0.0003)
	GIN	0.9593 (0.0052)
	GIN-JK	0.9641 (0.0029)
	GeniePath	0.9528 (0.0000)
	GeniePath-JK	0.9644 (0.0000)
	LGCN	0.7720 (0.0020)
NAS methods	Random	0.9882 (0.0011)
	Bayesian	0.9897 (0.0008)
	GraphNAS	0.9698 (0.0128)
	GraphNAS-WS	0.9584 (0.0415)
	SNAS-ASNG	**0.9902 (0.0003)**

superiority of the expressive yet simple search space. Another interesting observation is that Random and Bayesian show very strong capability in architecture search, which can also demonstrates the effectiveness of the designed search space in this work. Similar observation is also reported in two latest work in NAS literature (Li & Talwalkar, 2019; Sciuto et al., 2020), which is that random search can be a very strong baseline for CNN and RNN architecture search and the search space has a strong influence on the performance. When comparing SNAG and Random, the performance gain can be attributed to the RL-based search strategy comparing to uniformly randomly sampling one.

From Table 5, we can see that the performance trending is very similar to that in transductive task, which is that the NAS methods can obtain better or competitive performance than human-designed GNNs. When looking at the NAS methods, we can see that our proposed SNAG, Random and

Bayesian outperforms GraphNAS. This also demonstrates the superiority of the designed search space.

Taking into consideration the results of these two tasks, we demonstrate the effectiveness of SNAG models, especially the superiority of the search space. Besides, we emphasize two more interesting observations:

- Looking at the performance of all human-designed GNNs, we can see that the JK variants can consistently improve the performance in most cases, demonstrating the advantage of the layer aggregators proposed by JK-Network (Xu et al., 2018), which motivates the design of our search space.
- Another interesting observation is that the weight sharing mechanism have no definite performance gain in RL-based framework, which is also reported by GraphNAS and Auto-GNN. This also aligns with one latest research in NAS literature (Sciuto et al., 2020). Thus in practice, when applying SNAG to new graph-based tasks, we can use the one without weight sharing, which is memory-saving.

4.2.2 PERFORMANCE COMPARISONS IN TRANSFER TASKS

Table 6: Performance comparisons of transfer task on Reddit. We use Micro-F1 (with standard deviation) as the evaluation metric. The best results in different groups of baselines are underlined, and the best result is in boldface.

	Method	Micro-F1	Search cost (GPU hours)
Human-designed architectures	GraphSAGE	0.9379 (0.0003)	-
	GraphSAGE-JK	0.9421 (0.0005)	-
NAS approaches	Random	0.9301 (0.0023)	72
	Bayesian	0.9423 (0.0011)	72
Searched in a small graph	EGAN	**0.9528 (0.0008)**	0.013

4.2.3 THE SEARCHED ARCHITECTURES

Finally, we visualize the search architectures of EGAN on all datasets in Appendix.

4.3 GRAPH-LEVEL TASK

In this part, we evaluate SAAS algorithm on graph classification task with D&D Shervashidze et al. (2011) Lee et al. (2019) and PROTEINS Bastings et al. (2017) Lee et al. (2019) datasets. These two datasets are both the data set of protein graphs. In D&D dataset, nodes represent the amino acids and two nodes are connected iff the distance is less than 6 $\dot{A}$. In PROTEINS dataset, nodes are secondary structure elements and edges represent nodes are in an amino acid or in a close 3D space. More information are shown in Table.7

We add an global sum pooling method at the end of each architecture so that it can generate the graph representation vector. The performance are showed in Table. 8.

Table 7: The statistics of graph classification datasets.

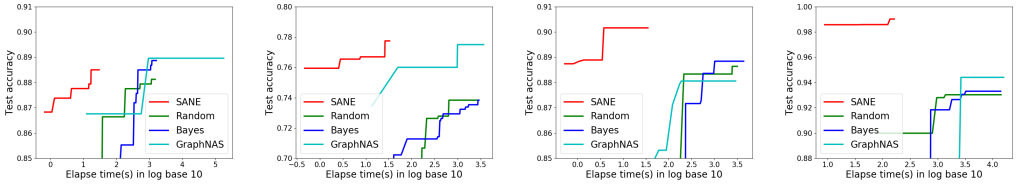
Dataset	Num. Graphs	Classes	Avg. Number of Nodes	Avg. Number of Edges
D&D	1178	2	384.32	715.66
PROTEINS	1113	2	39.06	72.82

Table 8: Performance comparisons on graph classification tasks. We show the test accuracy (with standard deviation). We categorize baselines into human-designed GNNs and NAS methods. The best results in different groups of baselines are underlined, and the best result is in boldface.

	Methods	D&D	PROTEINS
Human-designed GNN	GCN	0.727 (0.039)	0.724 (0.023)
	GCN-JK	0.733 (0.043)	0.730 (0.026)
	GraphSAGE	0.734 (0.029)	0.734 (0.041)
	GraphSAGE-JK	0.728 (0.047)	0.730 (0.035)
	GAT	0.713 (0.052)	0.734 (0.037)
	GAT-JK	0.716 (0.056)	0.745 (0.030)
	GIN	0.732 (0.030)	0.730 (0.032)
	GIN-JK	0.733 (0.033)	0.737 (0.048)
	GeniePath	0.705 (0.051)	0.670 (0.045)
	GeniePath-JK	0.704 (0.039)	0.694 (0.035)
	SAGPOOL	0.762 (0.009)	0.724 (0.041)
	ASAP	0.748 (0.024)	0.733 (0.024)
NAS methods	Random	0.742 (0.043)	0.731 (0.031)
	Bayesian	0.746 (0.031)	0.676 (0.041)
	GraphNAS	0.719 (0.045)	0.725 (0.031)
	GraphNAS-WS	0.758 (0.044)	0.752 (0.025)
	EGAN	**0.7791 (0.0338)**	**0.7612 (0.0394)**

4.4 Search Efficiency

Show the test acc w.r.t elapsed time.



(a) Cora. (b) CiteSeer. (c) PubMed. (d) PPI.

Figure 2: Test accuracy w.r.t. elapsed time.

Huan: Can also give the time cost table as in SANE. Move to appendix in case the space is limited.

4.5 Ablation Study

Finally, we present four ablation studies: 1) The efficacy of the simplified search space ;2) The performance gain from the layer aggregators; 3) The failure of searching for MLP. 4) The influence of the number of layers K.

4.5.1 The advantage of the simplified search space

Huan: Add graph level tasks, i.e., two more datasets.

In Section 3.1, we discuss the advantages of search space between SAAS-SNAG and GraphNAS/Auto-GNN. In this part, we conduct experiments to further show the advantages. To be specific, we run GraphNAS over its own and SANE's search space, given the same time budget (20 hours), and compare the final test accuracy of the searched architectures in Table 9. From Table 9, we can see that despite the simplicity of the search space, SANE can obtain better or at least

Table 9: Performance comparisons of two search spaces on four benchmark datasets. We show the mean classification accuracy (with STD). "-WS" represents the weight sharing variant of GraphNAS.

Methods	Cora	CiteSeer	PubMed	PPI	DD	Protein
GraphNAS	0.8840 (0.0071)	**0.7762 (0.0061)**	0.8896 (0.0024)	0.9698 (0.0128)		
GraphNAS-WS	0.8808 (0.0101)	0.7613 (0.0156)	0.8842 (0.0103)	0.9584 (0.0415)		
GraphNAS (Our search space)	0.8826 (0.0023)	0.7707 (0.0064)	0.8877 (0.0012)	**0.9887 (0.0010)**		
GraphNAS-WS (Our search space)	**0.8895 (0.0051)**	0.7695 (0.0069)	**0.8942 (0.0010)**	0.9875 (0.0006)		

close accuracy compared to GraphNAS, which means better architectures can be obtained given the same time budget, thus demonstrating the efficacy of the designed search space.

4.5.2 Performance gain from layers aggregators

In this part, to show the stronger expressive capability of the designed search space, we design an experiment to demonstrate the effectiveness of the layer aggregators, which is the key difference between SNAG and GraphNAS methods. To be specific, we conduct experiments on all seven datasets using a search space only with the node aggregators, i.e., removing the layer aggregators, as comparisons. The results are shown in Table 10, and we report the test accuracies of both the SNAG and SNAG-WS.

From Table 10, we can see that the performance consistently drops on all datasets when removing the layer aggregators. This demonstrates the importance of the layer aggregators for the final performance, which also aligns with the observation in Section 4.2 that the performance of human-designed GNNs can be improved by adopting the JK-Network architecture, i.e., adding an extra layer to aggregate all outputs of the intermediate node aggregation layers. Moreover, we further generalize this architecture design schema to search for the existence of skip connections between the intermediate and the final layers, which is the unique advantage of introducing NAS approaches to GNN. Therefore, we demonstrate the stronger expressive ability of SNAG compared to GraphNAS.

Table 10: Performance comparisons of EGAN using different search spaces.

	EGAN	
	layer aggregators (w)	layer aggregators (w/o)
Cora	0.9002 (0.0007)	0.8709 (0.0009)
CiteSeer	0.7587 (0.0002)	0.7511 (0.0004)
PubMed	0.8999 (0.0009)	0.8839 (0.0011)
CS	0.9466 (0.0014)	0.9370 (0.0011)
Photo	0.9620 (0.0004)	0.9462 (0.0015)
Computer	0.9154 (0.0026)	0.9157 (0.0010)
PPI	0.9902 (0.0003)	0.9742 (0.0003)
DD	0.7791(0.0338)	0.7606 (0.0375)
Protein	0.7612 (0.0394)	0.7575 (0.0340)

5 Conclusion and Future work

In this work, to overcome the drawbacks in expressive capability and search efficiency of two existing NAS approaches for GNN, i.e., GraphNAS (Gao et al., 2020) and Auto-GNN (Zhou et al., 2019b), we propose the SNAG framework, i.e., Simplified Neural Architecture search for GNN. By revisiting existing works, we unify state-of-the-art GNN models in a message passing framework (Gilmer et al., 2017), and design a simpler yet more expressive search space than that of GraphNAS and Auto-GNN. A RL-based search algorithm is designed and a variant (SNAG-WS) is also proposed by incorporating the weight sharing mechanism. Through extensive experiments on seven real-world datasets, we not only demonstrate the effectiveness of the proposed SNAG framework comparing to various baselines including GraphNAS and Auto-GNN, but also give bet-

ter understanding of different components of the proposed SNAG. Besides, we also provide practical guidance on applying NAS approaches to GNN. Based on this work, we build a valuable connection between NAS and GNN fields, which can open up a new research direction and benefit both sides.

For future work, we point out two potential directions. The first one is that we will explore more NAS approaches beyond RL based ones, e.g., differentiable architecture search (Liu et al., 2019a; Yao et al., 2020; Xie et al., 2019) and evolutionary-based methods (Real et al., 2019; Peng et al., 2020). The second one is that we will explore the SNAG framework in more graph-based tasks besides node classification.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, pp. 21–29, 2019.

Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, and Kouhei Nishida. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, pp. 171–180, 2019.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1957–1967, 2017.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In *ICML*, pp. 549–558, 2018.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NeurIPS*, pp. 2546–2554, 2011.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ICLR*, 2019.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, pp. 3730–3740, 2020.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *JMLR*, 2018.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLRW*, 2019.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *KDD*, pp. 1416–1424, 2018.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *IJCAI*, pp. 1403–1409, 2020.

Vikas K Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. pp. 5204–5215, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, 2017.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *IJCNN*, volume 2, pp. 729–734, 2005.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1024–1034, 2017.

Shengli Jiang and Prasanna Balaprakash. Graph neural network architecture search for molecular property prediction. *arXiv preprint arXiv:2008.12187*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2016.

Kwei-Herng Lai, Daochen Zha, Kaixiong Zhou, and Xia Hu. Policy-gnn: Aggregation optimization for graph neural networks. In *KDD*, pp. 461–471, 2020.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pp. 3734–3743, 2019.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *NN*, 6(6): 861–867, 1993.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *KDD*, pp. 631–636, 2006.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019a.

Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. Geniepath: Graph neural networks with adaptive receptive paths. In *AAAI*, volume 33, pp. 4424–4431, 2019b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *AISTAT*, pp. 493–503, 2020.

Matheus Nunes and Gisele L Pappa. Neural architecture search in graph neural networks. *arXiv preprint arXiv:2008.00077*, 2020.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *ICLR*, 2020.

Wei Peng, Xiaopeng Hong, Haoyu Chen, and Guoying Zhao. Learning graph convolutional network for skeleton-based human action recognition by neural searching. In *AAAI*, 2020.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pp. 4092–4101, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pp. 4780–4789, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *ICLR*, 2020.

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

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pp. 6105–6114, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *MLJ*, 8(3-4):229–256, 1992.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pp. 6861–6871, 2019.

Wenyi Xiao, Huan Zhao, Haojie Pan, Yangqiu Song, Vincent W Zheng, and Qiang Yang. Beyond personalization: Social content recommendation for creator equality and consumer satisfaction. In *KDD*, pp. 235–245, 2019.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pp. 5449–5458, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Quanming Yao and Mengshuo Wang. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. Efficient neural architecture search via proximal iterations. In *AAAI*, 2020.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pp. 974–983, 2018.

Jiaxuan You, Jure Leskovec, Kaiming He, and Saining Xie. Graph structure of neural networks. In *ICML*, pp. 327–337, 2020.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. *ICLR*, 2020.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *ICLR*, 2020.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, pp. 7603–7613, 2019a.

Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-GNN: Neural architecture search of graph neural networks. Technical report, arXiv preprint arXiv:1909.03184, 2019b.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pp. 8697–8710, 2018.

A APPENDIX

A.1 DETAILS OF NODE AGGREGATORS

As introduced in Section **??**, we have 12 types of node aggregators, which are based on well-known existing GNN models: GCN Kipf & Welling (2016), GraphSAGE Hamilton et al. (2017), GAT Veličković et al. (2018), GIN Xu et al. (2019), and GeniePath Liu et al. (2019b). Here we give key explanations to these node aggregators in Table 11. For more details, we refer readers to the original papers.

Table 11: More explanations to the node aggregators in (1).

GNN models	Symbol in the paper	Key explanations
GCN (Kipf & Welling, 2016)	`GCN`	$F_N^l(v) = \sum_{u \in \widetilde{N}(v)} \big(\text{degree}(v) \cdot \text{degree}(u)\big)^{-1/2} \cdot \mathbf{h}_u^{l-1}$
GraphSAGE (Hamilton et al., 2017)	`SAGE-MEAN`, `SAGE-MAX`, `SAGE-SUM`, `SAGE-LSTM`	Apply mean, max, sum, or LSTM operation to $\{\mathbf{h}_u \vert u \in \widetilde{N}(v)\}$.
GAT (Veličković et al., 2018)	`GAT`	Compute attention score: $\mathbf{e}_{uv}^{gat} = \text{Leaky_ReLU}\big(\mathbf{a}\big[\mathbf{W}_u\mathbf{h}_u \vert\vert \mathbf{W}_v\mathbf{h}_v\big]\big)$.
	`GAT-SYM`	$\mathbf{e}_{uv}^{sys} = \mathbf{e}_{uv}^{gat} + \mathbf{e}_{vu}^{gat}$.
	`GAT-COS`	$\mathbf{e}_{uv}^{cos} = \langle \mathbf{W}_u\mathbf{h}_u, \mathbf{W}_v\mathbf{h}_v \rangle$.
	`GAT-LINEAR`	$\mathbf{e}_{uv}^{lin} = \tanh\big(\mathbf{W}_u\mathbf{h}_u + \mathbf{W}_v\mathbf{h}_v\big)$.
	`GAT-GEN-LINEAR`	$\mathbf{e}_{uv}^{gen-lin} = \mathbf{W}_G \tanh\big(\mathbf{W}_u\mathbf{h}_u + \mathbf{W}_v\mathbf{h}_v\big)$.
GIN (Xu et al., 2019)	`GIN`	$F_N^l(v) = \text{MLP}\Big((1+\epsilon^{l-1}) \cdot \mathbf{h}_v^{l-1} + \sum_{u \in N(v)} \mathbf{h}_u^{l-1}\Big)$.
LGCN (Gao et al., 2018)	`CNN`	Use 1-D CNN as the aggregator.
GeniePath (Liu et al., 2019b)	`GeniePath`	Composition of GAT and LSTM-based aggregators
JK-Network (Xu et al., 2018)		Depending on the base above GNN

A.2 GRADIENTS OF $\mathcal{L}$

Here we give the gradients of $\mathcal{L}_w(\mathbf{Z})$ w.r.t. the hidden embeddings, model parameters, and the architecture parameters, which is in the following:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial x_j} &= \sum\nolimits_{m>j} \frac{\partial \mathcal{L}}{\partial x_m} \mathbf{Z}_m^T \frac{\partial \mathbf{O}_m(x_j)}{\partial x_j}, \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{i,j}^k} &= \frac{\partial \mathcal{L}}{\partial x_j} \mathbf{Z}_{i,j}^k \frac{\partial \mathbf{O}_{i,j}(x_i)}{\partial \mathbf{W}_{i,j}^k}, \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}_{i,j}^k} &= \frac{\partial \mathcal{L}}{\partial x_j} \mathbf{O}_{i,j}^T(x_i)(\boldsymbol{\delta}(k'-k) - \mathbf{Z}_{i,j})\mathbf{Z}_{i,j}^k \frac{1}{\lambda \boldsymbol{\alpha}_{i,j}^k}.
\end{aligned} \tag{7}
$$

For full derivation, we refer readers to the appendix of Xie et al. (2019).

A.3 DATASETS

Transductive Setting Only a subset of nodes in one graph are used as training data, and other nodes are used as validation and test data. For this setting, we use three benchmark dataset: Cora, CiteSeer, PubMed. They are all citation networks, provided by (Sen et al., 2008). Each node represents a paper, and each edge represents the citation relation between two papers. The datasets contain bag-of-words features for each paper (node), and the task is to classify papers into different subjects based on the citation networks.

Besides the three benchmark datasets, we use three more datasets: Coauthor CS, Amazon Computers and Amazon Photo, provided by (Shchur et al., 2018). Coauthor CS is a co-authorship graph where nodes are authors which are connected by an edge if they co-author a paper. Given paper keywords for each author's paper as node features, the task is to map each author to its most active field of study. Amazon Computers and Amazon Photo are segments of the Amazon co-purchase graph where nodes represent goods which are linked by an edge if these goods are frequently bought together. Node features encode product reviews as bag-of-word feature vectors, and class labels are given by product category.

For all datasets, We split the nodes in all graphs into 60%, 20%, 20% for training, validation, and test. For the transductive task, we use the classification accuracy as the evaluation metric.

Inductive Setting In this task, we use a number of graphs as training data, and other completely unseen graphs as validation/test data. For this setting, we use the PPI dataset, provided by (Hamilton et al., 2017), on which the task is to classify protein functions. PPI consists of 24 graphs, with each corresponding to a human tissue. Each node has positional gene sets, motif gene sets and immunological signatures as features and gene ontology sets as labels. 20 graphs are used for training, 2 graphs are used for validation and the rest for testing, respectively. For the inductive task, we use Micro-F1 as the evaluation metric.

Transfer Setting In this task, we use two datasets, Reddit and Arxiv, which are two orders of magnitude larger than Cora and CiterSeer in number of nodes, as shown in Table 3. The Reddit dataset is provided by Hamilton et al. (2017), and the task is to predict the community to which different Reddit posts belong. Reddit is an online discussion forum where users comment on different topics. Each node represents a post, and each edge represents a link between two posts, when they are commented by the same user. The dataset contains word vectors as node features. The graph is constructed from Reddit posts made in the month of September 2014, and we follow the same settings in the original paper Hamilton et al. (2017), which uses the first 20 days for training and the remaining days for test (with 30% used for validation).

The Arxiv dataset is constructed based on the citation network between all papers, and we use the specific version of `ogbn-arxiv`, provided by a recent open graph benchmark (OGB) project **?**, where the task is to predict the 40 subject areas of Arxiv CS papers, e.g., cs.AI, cs.LG. Each node (paper) has a 128-dimensional feature vector obtains by averaging the embedding of words in its title and abstract. And all papers are also associated with the year that the corresponding paper was published. The dataset is split by time. To be specific, papers published before 2017 are used as training data, while those in 2018 and 2019 are used, respectively, as validation and test set. For more details, we refer readers to **?**.

A.3.1 Compared Methods.

We compare SNAG with two groups of state-of-the-art methods: human-designed GNN architectures and NAS methods for GNN.

Human-designed GNNs. We use the following popular GNN architectures:

- GCN (Kipf & Welling, 2016) proposes a sum aggregator normalized by the degrees of nodes.
- GraphSAGE (Hamilton et al., 2017) proposes scalable graph neural network with different aggregators: Mean, Sum, Max-Pool, LSTM.
- GAT (Veličković et al., 2018) proposes the attention aggregators, and it has different variants according to the attention functions: GAT, GAT-SYS, GAT-LINEAR, GAT-COS, GAT-GENERALIZED-LINEAR. The detail of these attention functions are given in (Gao et al., 2020).
- GIN (Xu et al., 2019) proposes to use Multi-layern Perceptron (MLP) as aggregators.
- LGCN (Gao et al., 2018) proposes to automatically select topK neighbors for each node, and use the 1-D regular convolutional operation as the aggregator.
- GeniePath (Liu et al., 2019b) proposes a composition of attention and LSTM-style aggregators, which can learn adaptive neighborhood for each node.

For models with variants, like different aggregators in GraphSAGE or different attention functions in GAT, we report the best performance across the variants. Besides, we extend the idea of JK-Network (Xu et al., 2018) in all models except for LGCN, and obtain 5 more baselines: GCN-JK, GraphSAGE-JK, GAT-JK, GIN-JK, GeniePath-JK, which add an extra layer. For LGCN, we use the code released by the authors [7]. For other baselines, we use the popular open source library Pytorch Geometric (PyG) (Fey & Lenssen, 2019) [8], which implements various GNN models. For all baselines, we train it from scratch with the obtained best hyper-parameters on validation datasets,

[7] https://github.com/HongyangGao/LGCN

[8] https://github.com/rusty1s/pytorch_geometric

and get the test performance. We repeat this process for 5 times, and report the final mean accuracy with standard deviation.

NAS methods for GNN. We consider the following methods:

- Random search (denoted as "Random") is a simple baseline in NAS, which uniformly randomly samples architectures from the search space, and keeps track of the optimal architecture during the search process.
- Bayesian optimization[9] (denoted as "Bayesian") (Bergstra et al., 2011) is a popular sequential model-based global optimization method for hyper-parameter optimization, which uses tree-structured Parzen estimator as the metric for expected improvement.
- GraphNAS[10] (Gao et al., 2020), a NAS method for searching GNN architecture, which is based on reinforcement learning (Zoph & Le, 2017).

Note that AutoGNN (Zhou et al., 2019b) is not compared for three reasons: 1) the search spaces of Auto-GNN and GraphNAS is the actually the same; 2) both of these two works use the RL method; 3) the code of Auto-GNN is not publicly available.

For Random and Bayesian, the search is executed on the designed search space of SNAG. For GraphNAS, we use the its own search space and experimental settings. For other NAS methods, include the proposed SNAG, we set the search steps to 5,000 on all transductive tasks, and 200 for inductive tasks for the sake of efficiency. For Random and Bayesian methods, during the search process, a GNN architecture is sampled and trained for enough epochs ($600 \sim 1000$ depending on datasets) to obtain the validation performance. After searching finished, the architecture with the best validation performance is trained from scratch, and do some hyper-parameter tuning on validation dataset, run five times with the best hyper-parameters. For RL-based methods, in each epoch, a GNN architecture is sampled, and trained for enough epochs with early stop ($600 \sim 1000$ depending on datasets), then the validation accuracy is used to update the parameters of the RNN controller. After searching finished, we sample 10 architectures from the trained controller and collect the best architecture that achieves the best validation accuracy. Then the best architecture is trained from scratch. Again, we do some hyper-parameter tuning based on the validation dataset, and run five times with the best hyper-parameters. The mean accuracy with standard deviation are reported.

Note that for human-designed GNNs and NAS methods, for fair comparison and good balance between efficiency and performance, we choose set the number of GNN layers to be 3, which is an empirically good choice in the literature (Veličković et al., 2018; Liu et al., 2019b).

A.3.2 Implementation details of SNAG.

Our experiments are running with Pytorch (version 1.4.0) on a GPU 2080Ti (memory: 12GB, cuda version: 10.2). We implement SNAG on top of the building codes provided by GraphNAS and PyG (version 1.4.2). For the RNN controller, the relevant parameters are: the hidden embedding size is 100, the optimizer is $adam$, and the initial learning rate is 0.00035. For training each candidate architecture, the relevant parameters are: the hidden embedding size is 32, the number of attention head is 2, the optimizer is $adam$, the learning rate is 0.005, the dropout is 0.6, L2 regularization is 0.0005, and the training epochs is set to 800 with an early stop patience to 50. For the non-linear activation functions, we choose from ReLu or Elu according to the performance for simplicity and found that they are enough to obtain good performance in our experiments.

A.4 Results of the arxiv dataset in the transfer task

A.5 Searched architectures

Finally, we visualize the search architectures of EGAN on all datasets in Figure 3, and we can see that the node aggregators vary in different layers across different datasets. Besides, the skip-connections between all intermediate and final layers do not necessarily exist, while in JK-Network (Xu et al., 2018), they are required to be chosen. These two observations demonstrate

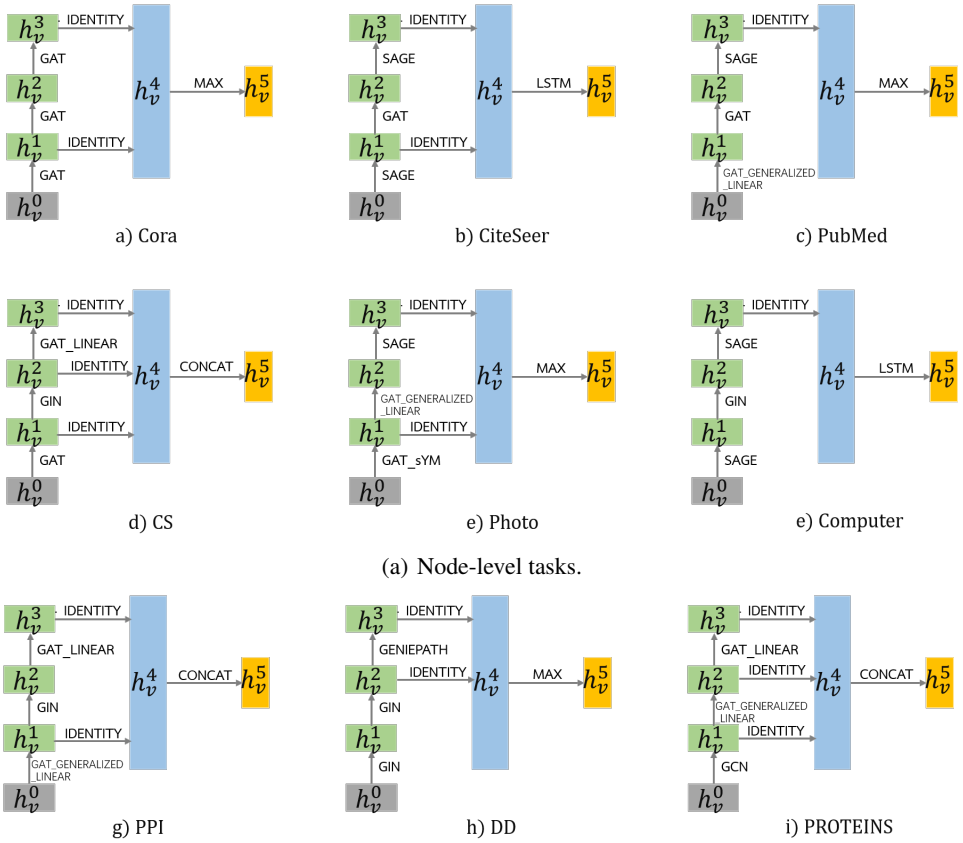
[9] `https://github.com/hyperopt/hyperopt`

[10] `https://github.com/GraphNAS/GraphNAS`

Table 12: Performance comparisons of transfer task on Arxiv. We use the accuracy (with standard deviation) as the evaluation metric. The best results in different groups of baselines are underlined, and the best result is in boldface.

	Method	Accuracy	Search cost (GPU hours)
Human-designed architectures	GCN	**0.7174 (0.0029)**	-
	GraphSAGE	0.7149 (0.0027)	-
NAS approaches	Random	0.6912 (0.0051)	72
	Bayesian	0.7033 (0.0039)	72
Searched in a small graph	EGAN	0.7121 (0.0006)	0.0025

the worthiness of incorporating NAS approaches to GNN literature, since we can obtain unknown effective architectures across different scenarios.



(a) Node-level tasks.

(b) Graph-level tasks.

Figure 3: The searched architectures by EGAN on for node-level and graph-level tasks.**Huan:** transfer task to be added..

A.6 More ablation studies

A.6.1 Failure of search for MLP

Table 13: The performance of searching for MLP. We list the best performance of EGAN from Table **??** as comparison.

Dataset	Random	Bayesian	EGAN
Cora	0.8698 (0.0011)	0.8470 (0.0032)	
CiteSeer	0.7298 (0.0078)	0.7103 (0.0057)	
PubMed	0.8662 (0.0030)	0.8699 (0.0065)	
PPI	0.8166 (0.0089)	0.8685 (0.0017)	

Huan: Can remove this part for this submission. In this part, we further show the performance of searching Multi-Layer Perception (MLP) as node aggregators since it is a universal function approximator: any continuous function on a compact set can be approximated with a large enough MLP (Leshno et al., 1993). And in Xu et al. (2019), Xu et al. shows that using MLP aggregators can be as powerful as the Weisfeiler-Lehman (WL) test under some conditions, which upper bounds the expressive capability of existing GNN models. However, in practice, it is very challenging to obtain satisfying performance by MLPs without prior knowledge, since it is too general to design the MLP of a suitable structure given a task. This motivates us to incorporate various existing GNN models as node aggregators in our search space, thus the performance of any search algorithm can be guaranteed. Then to show the challenges of directly searching for MLP as node aggregators, we propose to search for a specific MLP as node aggregators with NAS approaches. To be specific, we set the parameter space of the MLP as $w \in [8, 16, 32, 64]$ and $d \in [1, 2, 3]$, where w and d represents the hidden embedding size (width) and the depth of the MLP, respectively. For simplicity, we adopt the Random and Bayesian as the NAS approaches, and search on the four benchmark datasets, where the settings are the same as those in Section **??**. The performance is shown in Table 13, from which we can see that the performance gap between searching for MLP and EGAN are large. This observation demonstrates the difficulties of searching MLP as node aggregators despite its powerful expressive capability as WL test. On the other hand, it demonstrates the necessity of the designed search space of SANE, which includes existing human-designed GNN models, thus the performance can be guaranteed in practice.

A.6.2 THE INFLUENCES OF DIFFERENT KS

In the experiments of Section 4.2, we choose a 3-layer GNN as the backbone of the search space, i.e., $K = 3$ for its empirically good balance between the performance and computational resources. In this part, we conduct experiments with SNAG method by varying layer $K \in \{1, 2, 3, 4, 5, 6\}$ and show the test accuracy in Figure **??**(a). The results suggest that along with the increment of layers, the test accuracy may decrease. Considering the computational resources, 3-layer architecture is a good choice on these datasets.

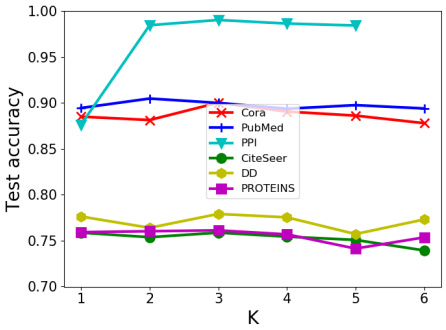


Figure 4: The test accuracy w.r.t. different Ks.**Huan:** Figure to be replaced.