A Graph Architecture Search Method Based On Grouped Operations

1st Luoyu Chen
School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
Beijing, China
chenluoyu19@mails.ucas.ac.cn

2nd Jungang Xu*
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
xujg@ucas.ac.cn

3rd Kun Jing
School of Computer Science and Technology
University of Chinese Academy of Sciences
Beijing, China
jingkun18@mails.ucas.ac.cn

4th Yingfei Sun
School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
Beijing, China
yfsun@ucas.ac.cn

***Abstract*—Graph data is ubiquitous in the real world and graph neural networks (GNNs) are effective for modeling the complex relationships and dependencies between the entities. However, it's difficult to design data-specific GNNs. Recently, researchers have started to apply neural architecture search (NAS) to design GNNs. In this work, we propose a graph architecture search method to decrease the instability with a large number of candidate operations. Following SANE(Search to Aggregate NEighborhood), we focus on searching to aggregate neighbourhoods but we divide the candidate operations into groups. We use a continuous relaxation of our search space and optimize the hyper-networks with a gradient-based algorithm. Extensive experiments on several node-level and graph-level tasks demonstrate that our method achieves a promising performance.**

***Index Terms*—neural architecture search, graph neural networks, node classification, graph classification, grouped operations**

I. Introduction

In recent years, the application of neural networks has promoted the research of pattern recognition and data mining. Convolution neural networks, Long Short-Term Memory (LSTM) [1] and AutoEcoder [2] can easily extract informative features from those data represented in the Euclidean space. However, lots of data from real world are in non-Euclidean space. For example, in a community networks, members are linked to each other via their behavioral characteristics and they need to be categorized into different groups. These data are in the form of graph and irregular. Motivated by traditional methods from deep learning, graph neural networks (GNNs) have emerged as an effective tool for analyzing data in non-Euclidean space. Graph neural networks are able to learn node embeddings or graph embeddings from the graph data. Graph neural networks show its great power on various graph-level tasks, e.g., online recommendation [3], fraud detection [4],bioinformatics [5], [6] and community detection [7].

*Corresponding author

Despite the popularity and success of GNN models, they still face some challenges. The data on graph-level tasks is usually from real world and these data are complicated and irregular. No single models can perform better than others on all the graph tasks. The existing works usually rely on manually designed architectures and hyper-parameters to achieve the best result. The manual experiments are time-consuming and need huge computational resources. It's significant to study how to design a data-specific GNN architecture for various graph-level tasks efficiently.

To reduce human efforts in developing neural architectures, neural architecture search (NAS) has been studied on deep learning tasks. NAS regards neural architecture design as a second-order optimization or search problem, which discovers local optimal architectures for target tasks in the predefined local optimal search space. For example, in computer vision, researchers [8]–[13] have employed reinforcement learning or evolution algorithm to design neural architectures on CIFAR-10 and ImageNet. There are also some attempts of using NAS approaches to design GNN architectures. GraphNAS [14] design a new search space that covers the operators from the state-of-the-art GNNs, such as GCN [15], GraphSAGE [16] and GAT [17]. GraphNAS automatically designs the best graph neural architecture using reinforcement learning. SANE [18] is a framework to use a differentiable search algorithm to search aggregation neighbourhoods.

In SANE, the authors use 11 node aggregators and 3 layer aggregators as the candidate operations in the search space. The architecture parameters of 11 node aggregators are optimized together and the best is chosen in the end. Actually, it's difficult to train so many architecture parameters at the same time. In search phase, the weights and architecture parameters of the hyper-networks update alternately and the candidate operations compete each other. When the search phase ends, the chosen node aggregator has maximal architecture parameter. However, the gap between the maximal architecture parameter and any of others is not large, which resulting in a gap between the hyper-networks and searched architectures. The gap is dubbed searching gap. So more candidate operations can lead to worse training stability.

In this work, we propose a novel graph neural network search method, which divides the node aggregators into groups. In our method, node aggregators with similar feature are divided into the same group. We divide the search method into two stages as well. In the first stage, we search the best group. In the second stage, we search the best single operation from the best group. As a result, in every stage, the number of architecture parameters is fewer and the searching gap is reduced to obtain a more stable training result. To verify the effectiveness, we conduct extensive experiments on multiple node and graph classification tasks.

II. Related Works

A. Graph Neural Networks

The notion of graph neural networks was first proposed in [19], and GNNs have derived many variants in the past few years. Let $G = (V, E)$ be a graph with node features $X \in \mathbb{R}^{|V| \times d}$, where $V = \{v_1, v_2, ..., v_{|V|}\}$ is a set of nodes and $E \subseteq V \times V$ is a set of edges. We use $N(v_i) = \{v_j : (v_i, v_j) \in E\}$ to denote the first-order neighbours of a node v_i in G. Generally, graph neural networks can be applied to two categories of tasks, node-level tasks and graph-level tasks. For node-level task, GNN models usually learn the hidden representation $H_V \in \mathbb{R}^{|V| \times d}$ of nodes and then adopt a predictor on node representation to complete the task [20]. For graph-level task, a representation for the whole graph is learned to complete the task. Most GNN models follow a message-passing framwork [21] . The hidden representation h_v of a node v in a L-layer GNN can be formulated as

$$h_v^l = \sigma(W^l \cdot AGG_{node}(\{h_u^{l-1}, \forall u \in N(v)\}), \tag{1}$$

where h_v^l is the hidden features of a node v learned by the l-th layer. W^l are learnable weights and σ is an activation function. AGG_{node} is the aggregation function to aggregate the message of its neighbours (see example in Fig. 1(a)). Some GNN models incorporate residual mechanisms to improve the performance [22], [23], and then the final representation of the node v is updated by a layer aggregator as

$$z_v = AGG_{layer}(h_v^1, h_v^2, ..., h_v^L), \tag{2}$$

where AGG_{layer} can be max-pooling, concatenation and other operations. For the graph-level representation, pooling operations are applied to the node representations

$$H_G = POOLING(H_V), \tag{3}$$

where H_G is the representation of the graph G.

B. Graph Architecture Search

Neural architecture search (NAS) has changed the convention of model design from manual to automatic and it can be formulated as the following bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}(W^*(\alpha), \alpha) \\ s.t. \quad &W^*(\alpha) = \arg\min_{W}(\mathcal{L}_{train}(W, \alpha)), \end{aligned} \tag{4}$$

where α is the optimization objective of NAS algorithm. $\mathcal{L}_{val}$ refers to loss function on validation data and $\mathcal{L}_{train}$ refers to loss function on training data. Early NAS methods focus on designing convolution neural networks. Those methods use RL-based algorithm [9], [11], [24], neuro-evolutionary algorithms [25] or one-shot model [26], [27], to search well-performed convolution neural architectures for image classification tasks.

Motivated by the success of NAS methods on designing CNNs. There are growing interests in combining the strength of graph machine learning methods and NAS methods together. GraphNAS [28] designs a new search space that covers the operators from the popular GNNs and uses reinforcement learning to search the best graph neural architecture. The work [29] proposes a novel framework for one-shot graph neural architecture search with a dynamic search space and solves the challenges of applying one-shot hyper-networks under a large scale search space. SANE [18] defines an expressive search space including node and layer aggregators, and it applies a differentiable architecture search algorithm similar to DARTS [26]. GNAS [30] propose a novel paradigm with a tree-topology computation procedure and atomic operations to construct GNNs.

III. Methodology

A. The Design of the Search Space

We design a hierachical search space which contains powerful enough child architectures for target tasks. Like SANE, we focus on exploring the best aggregators among GNN layers for different tasks. We choose 11 node aggregators, 3 layer aggregators and skip (IDENTITY) operation as the candidate operations, as shown in Table I. We denote the node aggregator set by O_n, the layer aggregator set by O_l and skip operation set by O_s. The node aggregators come from the popular GNN models, such as GCN, GAT and GIN. The layer aggregators are used to combine hidden embeddings from different layers. Correspondingly, we need IDENTITY and ZERO operations to connect intermediate layer and the last layer.

Operations that serve as similar functions are treated as the same group. In practice, we divide the 11 node aggregators into five groups. The node aggregators based-on the same model are in the same group. For example, SAGE-SUM, SAGE-MEAN and SAGE-MAX are in the same group. We mix the operations in the same group as a new candidate operation. We mix the node aggregators based on GraphSAGE and GAT as grouped operations G_SAGE and G_GAT, which are respectively formulated as

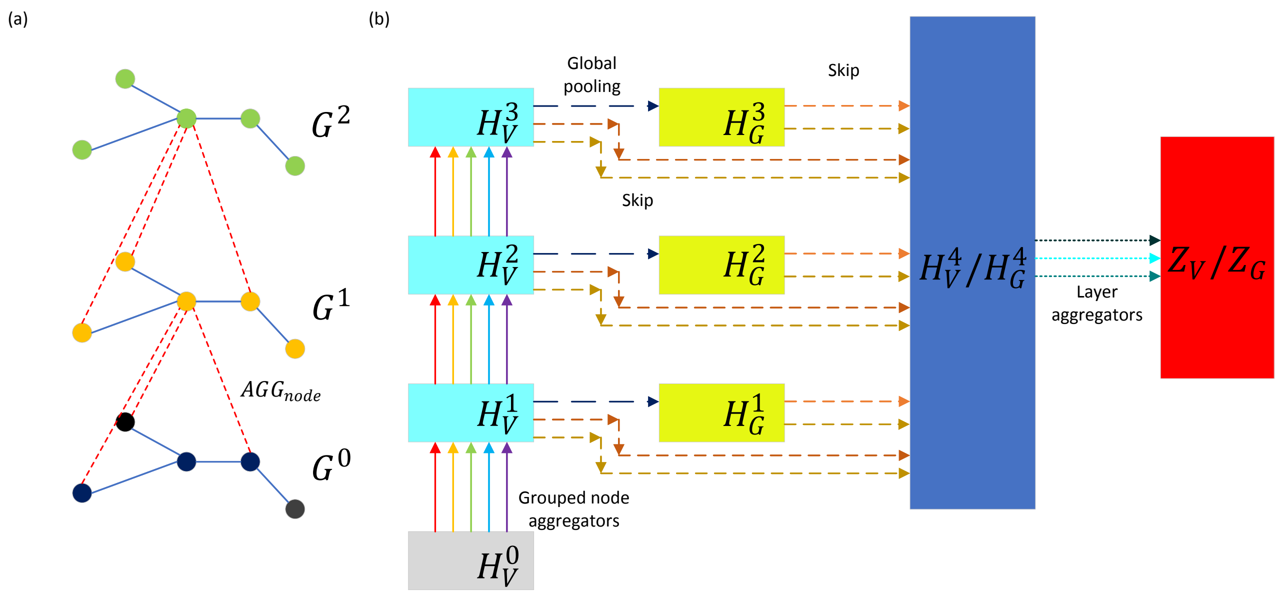

Fig. 1. (a)An example of message-passing in GNN models. The node aggregates information(message) from its neighbours through AGG_{node}. (b)Hyper-networks corresponding to our search space.

TABLE I
THE CANDIDATE OPERATIONS IN OUR SEARCH APACE, INCLUDING NODE AGGREGATORS O_n, LAYER AGGREGATORS O_l AND SKIP-CONNECTION.

	Models	Node aggregators	# of aggregators
O_n	GCN [15]	GCN	1
	GraphSAGE [16]	SAGE-SUM,SAGE-MEAN,SAGE-MAX	3
	GAT [17]	GAT, GAT-SYM,GAT-COS,GAT-LINEAR,GAT-GEN-LINEAR	5
	GIN [31]	GIN	1
	GeniePath [32]	GeniePath	1
O_l	-	CONCAT, MAX, LSTM	3
O_s	JK-Network [22]	IDENTITY, ZERO	2

$$G\_SAGE = SAGE\text{-}SUM + SAGE\text{-}MEAN + SAGE\text{-}MAX \tag{5}$$

$$G\_GAT = GAT + GAT\text{-}SYM + GAT\text{-}COS + GAT\text{-}LINEAR + GAT\text{-}GEN\text{-}LINEAR \tag{6}$$

For GCN, GIN and GeniePath, we consider them as the single group alone. We also split the search phase into two stages. In first stage, we search for the best group for node aggregators. In the second stage, we only search the operations in the chosen group for node aggregators.

B. Differentiable Architecture Search

In this subsection, we describe the computation procedure for an architecture and the continuous relaxation of our search space. We also introduce a differentiable search algorithm in [26].

We build a hyper-network with L layers corresponding to our search space, as shown in Fig. 1(b) ($L = 3$). The hyper-network can be viewed as a directed acyclic graph (DAG) with $L + 3$ (for node-level tasks) or $L + 6$ (for graph-level tasks) nodes. Each node x^l in the hyper-networks is a latent representation, including input features and node or graph embeddings in GNN models. The directed edge $e(i, j)$ is associated with some operation $o^{(i,j)}$ that transforms x^l. And the target can be transformed to choose a proper operation on each edge.

To make the search space continuous, we relax the categorical choice of a particular operation as a softmax over all possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o \in O} \frac{exp(\alpha_o^{i,j})}{\sum_{o' \in O} exp(\alpha_{o'}^{i,j})} o(x), \quad (7)$$

where the operation mixing weights for a pair of node (i, j) are parameterized by the vector $\alpha^{(i,j)}$ of dimension $|O|$. The vector $\alpha^{(i,j)}$ denotes architecture parameters. The set O is the set of candidate operations, including O_n, O_l, O_s. And we set the corresponding architecture parameters $\alpha_n, \alpha_l, \alpha_s$ for them. Significantly, O_n is dynamic. In the first searching stage, it contains five grouped candidate operations, and it only contains the operations in the chosen group in the second searching stage.

We denote $\bar{o}_n, \bar{o}_s$ and $\bar{o}_l$ as the mixed operations from O_n, O_l, O_s based on (7). In the hyper-networks, we denote the hidden embeddings of nodes as $H_V \in \mathbb{R}^{|V| \times d}$ in the graph and the embedding of each node h_v is

$$h_v^l = \sigma(W_n^l \cdot \bar{o}_n(h_u^{l-1}, \forall u \in N(v))) \quad (8)$$

where W_n^l are learnable weights and shared by candidate architectures. Different from node-level tasks, additional pooling operation is calculated for graph-level tasks. We add a global pooling function to obtain the graph embedding $H_G \in \mathbb{R}^d$. We calculate the graph embeddings as follows,

$$H_G^l = \sum_{v \in V} h_v^{l-1}. \quad (9)$$

For the last layer, we calculate the embeddings for node-level tasks, by

$$\begin{aligned} H_V^{L+1} &= [\bar{o}_s(H_V^1), \cdots, \bar{o}_s(H_V^L)], \\ Z_V^{L+1} &= \bar{o}_l(H_V^{L+1}). \end{aligned} \quad (10)$$

And for graph-level task, the embeddings are computed by

$$\begin{aligned} H_G^{L+1} &= [\bar{o}_s(H_G^1), \cdots, \bar{o}_s(H_G^L)], \\ Z_G^{L+1} &= \bar{o}_l(H_G^{L+1}). \end{aligned} \quad (11)$$

The operation $[\cdot]$ means that we stack all the embeddings from each intermediate layers. Then we apply layer aggregators to the stacked embeddings.

The search process of our method can be formulated as a bi-level optimization problem like (4). Following DARTS, we use training data to update the network weights W and use validation data to update the architecture parameters α in the search phase. The gradient descent w.r.t. α is given as,

$$\begin{aligned} &\bigtriangledown_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ &\approx \bigtriangledown_\alpha \mathcal{L}_{val}(w - \xi \bigtriangledown_w \mathcal{L}_{train}(w(\alpha), \alpha), \alpha) \end{aligned} \quad (12)$$

In the end of search, a discrete GNN model is obtained by replacing each mixed operation with the most likely operation, i.e, $o^{(i,j)} = argmax_{o \in O} \ \alpha_o^{(i,j)}$.

IV. Experiments

We apply our methods to multiple node-level and graph-level tasks, and the results demonstrate that our method can be used to design data-specific GNN models effectively.

TABLE II
Statistics of the datasets for transductive and inductive tasks in the experiment.

Datasets	# of Nodes	# of Edges	# of Features	# of Classes	Task
Cora	2708	5278	1433	7	Transductive
CiteSeer	3327	4552	3703	6	Transductive
PubMed	19717	44324	500	3	Transductive
PPI	56944	818716	121	50	Inductive

TABLE III
Statistics of the datasets for graph-level tasks.

Datasets	# of Graphs	Avg. # of Nodes	# of Edges	# of Features	# of Classes
D&D	1178	384.3	715.7	89	2
PROTEINS	1113	39.1	72.8	3	2
IMDB-BINARY	19.8	96.5	500	0	2
IMDB-MULTI	13	65.9	500	0	3
COX2	467	41.2	43.5	3	2

A. Datasets and Tasks

For node-level task, we search for architectures and test their performances on several node classification tasks. We use three benchmark datasets such as Cora, CiteSeer and PubMed for transductive task. In transductive task, the nodes in one graph are divided into three parts: training, validation and test data. Cora, CiteSeer, PubMed are all citation networks, provided by [33]. Each paper in the networks is viewed as a node, and the citation relation between two papers is regarded as an edge connecting two nodes. The task is to classify each node (paper) into different categories (subjects). The nodes in all graphs are split into 60%, 20%, 20% for training, validation and test. We also use the PPI dataset for inductive task. In inductive task, we randomly choose part of the graphs for training data, and other unseen graphs for validation and test data. The PPI dataset, provided by [16], is to classify protein roles in term of their cellular functions from gene ontology. The PPI dataset consists 24 protein-protein interaction graphs, and each graph corresponds to human tissue. The nodes use position gene sets, motif gene sets and immunological signatures as features and gene ontology sets as labels. The statistics of the datasets for transductive and inductive tasks can be seen in Table II.

For graph-level task, we use five datasets as shown in Table III. These tasks are associated with the whole graph. D&D and PROTEINS datasets are both for predicting protein function. In the D&D dataset, the nodes represent the amino acids and the edges are built if the distance between two nodes is less than $6\dot{A}$. In the PROTEINS dataset, the nodes are secondary structure elements and two nodes are connected if they are in an amino or in a close 3D space. IMDB-BINARY and IMDB-MULTI datasets [34] are movie-collaboration datasets. They contains actor/actress and genre information of different movies on IMDB. For each graph in IMDB-BINARY and

TABLE IV
RESULTS OF THE EXPERIMENTS FOR NODE-LEVEL TASKS.

		Transductive			Inductive
	Model	Cora	PubMed	CiteSeer	PPI
Human-designed	GAT	0.8611±0.0030	0.8558±0.0002	0.7127±0.0055	0.8105±0.0001
	GCN	0.8685±0.0031	0.8738±0.0008	0.7485±0.0000	0.9134±0.0000
	GIN	0.8704±0.0012	0.8823±0.0023	0.7139±0.0019	0.9770±0.0062
	GraphSAGE	0.8544±0.0036	0.8845±0.0004	0.7304±0.0000	**0.9864±0.0001**
	GeniePath	0.8426±0.0035	0.8624±0.0018	0.7259±0.0051	0.6980±0.0299
NAS approaches	SANE	0.8652±0.0022	0.8879±0.0003	**0.7554**±0.0738	0.9798±0.0013
	Ours	**0.8804±0.0015**	**0.8905±0.0013**	0.7545±0.0000	0.9857±0.0004

TABLE V
RESULTS OF THE EXPERIMENTS FOR GRAPH-LEVEL TASKS.

	Model	D&D	PROTEINS	IMDB-BINARY	IMDB-MULTI	COX2
Human-designed	GAT	0.7573±0.0299	0.7387±0.0000	0.7000±0.0000	**0.5133±0.0000**	0.8174±0.0106
	GCN	0.7795±0.0205	0.7568±0.0000	0.7380±0.0147	0.4933±0.0000	0.8130±0.0106
	GIN	0.7231±0.0000	0.7387±0.0000	0.7140±0.0206	0.4880±0.0027	0.8435±0.0222
	GraphSAGE	0.7949±0.0000	0.7297±0.0000	0.7100±0.0000	**0.5133±0.0000**	0.8000±0.0087
	GeniePath	0.7726±0.0166	0.7477±0.0000	0.6980±0.0299	0.5040±0.0033	0.8478±0.0000
NAS approaches	Ours	**0.7966±0.0034**	**0.7658±0.0000**	**0.7500±0.0000**	**0.5133±0.0000**	**0.8675±0.0020**

IMDB-MULTI, the nodes represent actors/actress and the edges between them mean that they appear in the same movie. The task is to identify which genre an ego-network graph belongs to. COX2 [35] dataset is from Chemistry domain and it is a set of 467 cyclooxygenase-2(COX-2) inhibitors. The task is to classify the compounds as active or inactive with vitro activities against human recombinant enzyme values. The graphs in all datasets are split into 80%, 10%, 10% for training, validation and test.

B. Implementations Details

In the search phase, we use the training data to train the model weights and the validation data to train architecture parameters. For model weights, we use a SGD optimizer with an initial learning rate of 0.025 following a cosine schedule, the momentum of 0.9 and the weight decay of 5e-4. For architecture parameters, we use an Adam optimizer with an initial learning rate of 3e-3 and the weight decay of 0.001.

In the evaluation phase, we train the searched candidate GNNs from the corresponding search space and other baselines. All the human-crafted GNNs and the searched candidates are tuned individually with hyper-parameters, such as hidden size, learning rate, dropout, etc. After some hyper-parameters are tuned, we repeat the re-training of the architectures for five times, and report the final mean accuracy with standard deviation.

C. Performance Analysis

We choose five human-designed GNN models for comparisons, including GAT, GCN, GIN, GraphSAGE and GeniePath. The human-designed GNNs focus on designing aggregation, and we build the models without skip connection by only setting the node aggregators. We also compare our method with SANE on node-level tasks. We choose a 3-layer backbone network for all the baselines and our method.

TABLE VI
TIME (SECONDS) SPENDING ON THE SEARCHING PHASE.

Datasets	SANE	Ours		
		1st stage	2nd stage	total
Cora	5.50	6.47	0.66	7.13
PubMed	5.47	7.91	0.73	8.66
CiteSeer	4.13	5.15	0.63	5.78
PPI	85.06	106.62	38.32	144.94

The results of experiments are shown in Table IV and Table V. We can see that no single human-designed GNN model can outperform other models on all datasets. For example, GIN performs best on Cora, while GraphSAGE performs best on PubMed. For graph-level task, Gin performs better than GraphSAGE on PROTEINS, IMDB-BINARY and COX2, but performs worse on D&D and IMDB-MULTI. It demonstrates that obtaining adaptive architectures for various specific tasks is necessary.

As shown in Table IV, we can see that our searched architecture outperforms other models on Cora and PubMed. The accuracies of our searched models are worse than SANE on CiteSeer and GraphSAGE on PPI dataset, but the gap is very small. For graph-level tasks, our models perform better than all the human-designed models on four datasets except for IMDB-MULTI. For IMDB-MULTI, our searched model shares the best performance with GAT and GraphSAGE. In

TABLE VII
GROUP INFORMATION OF THE EXPERIMENTS FOR GROUPING RANDOMLY.

Gounp Method	Random 1	Random 2	Random 3
Group 1	GAT-SYM/GCN	SAGE-MEAN/GeniePath	SAGE-MEAN/GAT-GEN-LINEAR
Group 2	SAGE-MAX/GENIEPATH	SAGE-SUM/GAT-GEN-LINEAR	SAGE-SUM/GAT
Group 3	GAT-COS/SAGE-MEAN	SAGE-MAX/GAT-COS	GAT-LINEAR/GAT-COS
Group 4	GAT-GEN-LINEAR/GAT-LINEAR	GCN/GAT-LINEAR	GCN/SAGE-MAX
Group 5	GIN/GCN/SAGE-SUM	GAT/GAT-SYM/GIN	GeniePath/GIN/GAT-SYM

TABLE VIII
RESULTS OF THE EXPERIMENTS FOR GROUPING RANDOMLY.

Gounp Method	Cora	D&D
Random 1	0.8526±0.0034	0.7607±0.0021
Random 2	0.8407±0.0000	0.7265±0.0000
Random 3	0.8585±0.0009	0.7538±0.00013
Ours	0.8804±0.0015	0.7966±0.0034

Table V, it can be seen that the accuracies of our models are increased by 0.012 on IMDB-BINARY and 0.019 on COX2. These results demonstrate the need for data-specific methods for graph learning, and the effectiveness of our method on automatically designing adaptive GNN models. All the searched architectures are shown in Fig. 2.

We report the time consuming in the searching phase in Table VI. We use a single Tesla P100 for the experiments. In our method, the search cost in the first stage is significantly higher than that in the second stage. Compared to SANE, although the searching phase is divided into two stages, the growth rate for search cost is no more than twice. Considering that the search cost is counted in seconds, the increased search time is acceptable.

V. Ablation Study

In our method, we group the node aggregators according to their similar functions. If the aggregators are grouped randomly, the number of architecture parameters in the first searching stage would decrease as well. But grouping the aggregators might add other bad factors during the searching phase. We conduct experiments for grouping the node aggregators randomly to study the influence of different grouping elections.

For grouping randomly, we divide the 11 node aggregators into five groups, where four groups with two operations and another one with three operations. Each group chooses aggregators from the candidate aggregators randomly. The experiments are conducted on Cora and D&D datasets. Other experimental settings keep the same with those in Section IV. Table VII shows the group information in the experiment Random 1, Random 2 and Random 3. The results are reported in Table VIII. It can be seen that the accuracies of the architectures searched by grouping aggregators randomly are lower than our method. For D&D dataset, there exists also great gaps among Random 1, Random 2 and Random 3. It demonstrates that grouping aggregators randomly can bring some bad factors to different groups and then influences the searching results.

VI. Conclusion and Future Work

In this paper, we propose a framework to automatically search data-specific GNN architectures. Similar to SANE, we focus on finding proper aggregators between layers. To decrease the instability because of optimizing too much candidate node aggregators, we group the node aggregators according to their similar feature extraction. We use a differetiable architecture search algorithm to optimize the network weights and architecture parameters. We conduct experiments on four node-level tasks and five graph-level tasks and the experimental results demonstrate the superiority of our method.

For future work, we plan to add more candidate operations to our method so that we can find more novel GNN architectures beyond human expertise. Besides, we plan to explore the possibility for searching larger GNNs with deeper layers and evaluate it on other larger datasets.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[2] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[3] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proc. AAAI*, 2019.

[4] J. Zhang, B. Dong, and P. S. Yu, "Fakedetector: Effective fake news detection with deep diffusive neural network," in *Proc. ICDE*, 2020.

[5] C. Su, J. Tong, Y. Zhu, P. Cui, and F. Wang, "Network embedding in biomedical data science," *Briefings in Bioinformatics*, vol. 21, no. 1, pp. 182–197, 2020.

[6] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[7] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," in *Proc. ICLR*, 2019.

[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. ICLR*, 2017.

[9] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. ICLR*, 2017.

[10] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. ICML*, 2018, pp. 4092–4101.

[11] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. CVPR*, 2018.

[12] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. ICLR*, 2018.

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI*, 2019.

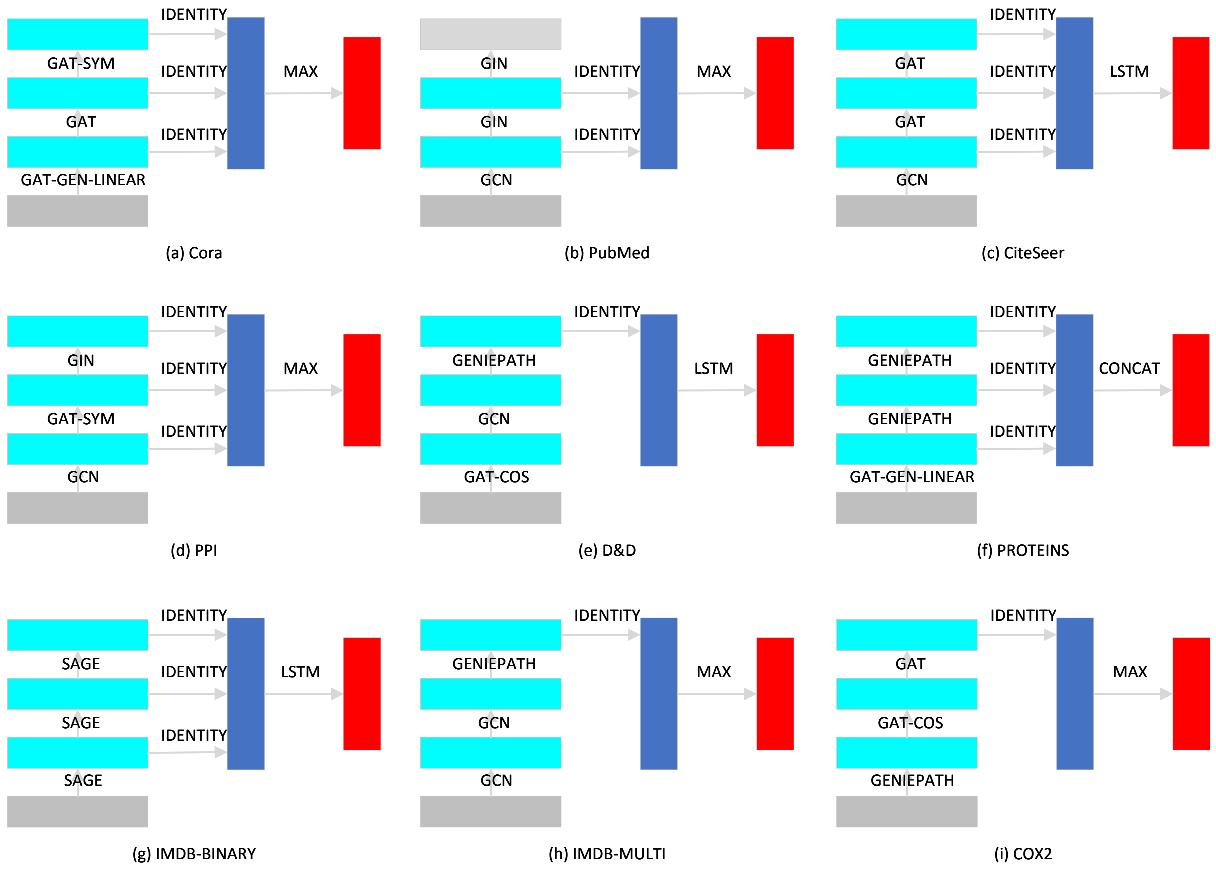

Fig. 2. The searched GNN architectures by our method. For graph-level tasks, we omits the global pooling components for simplication.

[14] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *Proc. IJCAI*, 2020.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017.

[16] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NeurIPS*, 2017.

[17] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. ICLR*, 2018.

[18] H. Zhao, Q. Yao, and W. Tu, "Search to aggregate neighborhood for graph neural network," *CoRR*, vol. abs/2104.06608, 2021.

[19] M. Gori, G. Monfardini, and F. Scarselli, "A new model for earning in raph domains," 2005.

[20] Z. Zhang, X. Wang, and W. Zhu, "Automated machine learning on graphs: A survey," in *Proc. IJCAI*, 2021.

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

[22] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. ICML*, 2018.

[23] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. ICML*, 2020.

[24] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu, "Practical block-wise neural network architecture generation," in *Proc. CVPR*, 2018.

[25] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. ICLR*, 2019.

[26] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *Proc. ICLR*, 2019.

[27] X. Chu, B. Zhang, and X. Li, "Noisy differentiable architecture search," *CoRR*, vol. abs/2005.03566, 2020.

[28] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *CoRR*, vol. abs/1904.09981, 2019.

[29] Y. Li, Z. Wen, Y. Wang, and C. Xu, "One-shot graph neural architecture search with dynamic search space," in *Proc. AAAI*, 2021.

[30] S. Cai, L. Li, J. Deng, B. Zhang, Z. Zha, L. Su, and Q. Huang, "Rethinking graph neural architecture search from message-passing," in *Proc. CVPR*, 2021.

[31] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. ICLR*, 2019.

[32] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proc. AAAI*, 2019.

[33] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[34] P. Yanardag and S. V. N. Vishwanathan, "Deep graph kernels," in *Proc. SIGKDD*, 2015.

[35] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships," *Journal of Chemical Infomation and Computer Science*, vol. 43, no. 6, pp. 1906–1915, 2003.