GraphPAS: Parallel Architecture Search for Graph Neural Networks

Jiamin Chen, Jianliang Gao, Yibo Chen, Oloulade Babatounde Moctard, Tengfei Lyu, and Zhao Li. 2021. GraphPAS: Parallel Architecture Search for Graph Neural Networks. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’21), July 11–15, 2021, Virtual Event, Canada. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3404835.3463007

1 INTRODUCTION

In recent years, Graph Neural Networks (GNNs) have demonstrated great practical performance for analyzing graph-structured data [1].

Graph neural architecture search has received a lot of attention as Graph Neural Networks (GNNs) has been successfully applied on the non-Euclidean data recently. However, exploring all possible GNNs architectures in the huge search space is too-time consuming or impossible for big graph data. In this paper, we propose a parallel graph architecture search (GraphPAS) framework for graph neural networks. In GraphPAS, we explore the search space in parallel by designing a sharing-based evolution learning, which can improve the search efficiency without losing the accuracy. Additionally, architecture information entropy is adopted dynamically for mutation selection probability, which can reduce space exploration. The experimental result shows that GraphPAS outperforms state-of-art models with efficiency and accuracy simultaneously.

CCS CONCEPTS
• Computing methodologies → Search methodologies.

KEYWORDS
neural architecture search; graph neural networks; parallel search

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGIR ’21, July 11–15, 2021, Virtual Event, Canada © 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-8037-9/21/07. ... $15.00 https://doi.org/10.1145/3404835.3463007
this paper, we propose a parallel graph neural architecture search framework (GraphPAS) that can speed up the search with high accuracy simultaneously. In the GraphPAS, we design sharing-based evolution learning to enhance the diversity of population. For the evolutionary search, we utilize information entropy to assess the mutation selection probability of architecture component, which can reduce exploration effectively in the huge search space for achieving good GNNs architecture. The main contributions of this paper are summarized as follows:

- To the best of our knowledge, we make the first attempt to parallel search framework for graph neural architecture search.
- We propose a novel neural architecture search framework GraphPAS to find the optimal GNNs architecture automatically. In the GraphPAS, we design sharing-based evolution learning and adopt architecture information entropy to restrict search direction, which can improve the search efficiency without losing the accuracy.
- The experiments show that GraphPAS outperforms state-of-art models in both efficiency and accuracy on real-world datasets.

2 THE PROPOSED GRAPHPAS METHOD

In this section, we first define the problem formulation of graph neural architecture search and then introduce the search space of GNNs. Finally, we discuss the parallel search algorithm of GraphPAS.

2.1 Problem Formulation

Given a search space $S$ of graph neural architecture, a train dataset $D_{train}$, a validation dataset $D_{val}$ and estimation metric $M$, our target is to gain the optimal graph neural architecture $s_{opt} \in S$ which is trained on $D_{train}$ based on loss function $L$ and achieves the best metric $M^*$ on set $D_{val}$. Estimation metric $M$ is represented by accuracy for node classification task. Mathematically, it is expressed as follows:

$$s_{opt} = \arg\max_{s \in S} M(s(\theta^*), D_{val})$$

$$\theta^* = \arg\max_{\theta} L(f(\theta), D_{train}).$$

The Figure 2 shows the parallel search framework, which applied to solve Eq. 1.

2.2 Search Space

It is flexible to customize the search space of graph neural architecture. Typically, one layer of GNNs involves five architecture components:

- **Attention function (att)**: Although each node need to aggregate information from its neighbors, different neighbor node has different contribution for the central node [5]. The attention function aims to compute the correlation coefficient $r_{ij}$ for each edge relationship $\varepsilon_{ij}$ between the nodes $v_i$ and $v_j$. The types of attention functions are shown in Table 1.

- **Aggregation function (agg)**: When we get all correlation coefficient $r_{ij}$ for the central node $v_i$ in its receptive field $N(v_i)$, in order to achieve central node hidden representation $h_i$, the aggregation operator is required to merge the feature information from the neighbor nodes based on correlation coefficient $r_{ij}$ [6]. The aggregation manners include mean, max and sum in this work.

- **Activation function (act)**: After acquiring the node hidden representation $h_i$, for the sake of smoothing $h_i$, a non-linear transformer is usually used. The choice of activation functions in our GraphPAS is listed as follows: \{tanh, sigmoid, relu, linear, softplus, leaky relu, relu6, elu\}.

- **Multi-head (head)**: Instead of using the attention function once, research [7] shows that multiple independent attention operators can stabilize the learning process. We collect the number of multi heads within the set: \{1, 2, 4, 6, 8\}.

- **Hidden dimension (dim)**: The node hidden representation $h_i$ is multiplied by a trainable matrix $W^{(l)}$ in the each output layer of GNNs, $l$ is the number of GNNs layer, which is used to enhance the hidden representation and reduce feature dimensions. In this work, the set of hidden dimensions is \{8, 16, 32, 64, 128, 256, 512\}.

2.3 Search Algorithm

With the analyzing of architecture features of GNNs, we find that different architecture component value has different frequency of occurrence in the GNNs architectures that can get good performance on given dataset. Inspired by the theory that association algorithm mines frequent itemset[8], in order to restrict search direction in the huge search space for finding good GNNs architectures faster, we use architecture information entropy to dynamically evaluate the mutation selection probability. Next, architecture components are selected based on the mutation selection probability to perform random mutation operation.

**Parallel Initialization.** We use multiple workers to explore different areas in the huge search space simultaneously for speed up search process. Using $N$ workers to randomly explore $N \times M$ GNNs architectures as the initial population in parallel, estimating the validation accuracy as the fitness on the dataset $D_{val}$ for GNNs architectures of initial population.

**Sharing Population.** The sharing mechanism is designed to make each worker can utilize the GNNs architecture generated by other worker, it can improve diversity of population for evolution search. Ranking GNNs architectures based on fitness for initial population, and extracting top $n$ GNNs architectures as the sharing population.

**Restricting Search Direction.** Computing information entropy $h(i)$ for each architecture component by Eq. 2 based on frequency distribution of architecture component value in sharing population.

| Table 1: Attention functions of search space $S$. |
|-------------------------------|------------|
| Attention Functions | values |
| gat | $r_{ij}^{gat} = \text{reaky } \text{relu}(W_c \ast h_i \ast W_n \ast h_j)$ |
| gcn | $r_{ij}^{gcn} = 1/\sqrt{d_{ij}}$ |
| cos | $r_{ij}^{cos} = \leq W_c \ast h_i, W_n \ast h_j >$ |
| const | $r_{ij}^{const} = 1$ |
| sym-gat | $r_{ij}^{sym} = r_{ij}^{gat} + r_{ji}^{gat}$ |
| linear | $r_{ij}^{lin} = \text{tanh}(\text{sum}(W_c \ast h_j))$ |
| gene-linear | $r_{ij}^{gen} = W_h \ast \text{tanh}(W_c \ast h_i + W_n \ast h_j)$ |
Figure 2: Parallel search framework for graph neural architecture search. First, $N \times M$ GNNs architectures are generated randomly by $N$ workers in parallel as initial population. Then, selecting top $n$ GNNs architectures based on fitness as sharing population. Subsequently, computing information entropy vector $\overline{H}$ based on frequency distribution of architecture component value in sharing population. Next, utilizing vector $\overline{H}$ to dynamically identify mutation selection probability vector $\overline{P}$. After, each worker samples child architecture simultaneously based on sharing population and architecture mutation selection probability vector $\overline{P}$. Finally, merging child architectures generated by $N$ workers as sharing children, putting child architecture into sharing population if the fitness of child architecture is greater than threshold.

$$h(c_i) = -\sum_{j} f(v_j) \log_2 f(v_j)$$

$$i \in [1, 5 \times l], l \geq 1;$$

where $h(c_i)$ represents the information entropy of the $i$-th component of GNNs architecture, $v_j$ denotes the $j$-th value of the $i$-th component, which appears in sharing population, counting $v_j$ frequency of occurrence as $f(v_j)$, $l$ is the number of GNNs layer.

Then, acquiring architecture information entropy vector $\overline{H} = [h(c_1), h(c_2), ..., h(c_l)]$. Finally, based on Eq. 3 to get the architecture component mutation selection probability $p_i$. And $\overline{P} = [p_1, p_2, ..., p_l]$ is represented as the architecture mutation selection probability vector.

$$p_i = \text{softmax}(h(c_i))$$

$$i \in [1, 5 \times l], l \geq 1.$$  

The architecture mutation selection probability vector $\overline{P}$ is a soft constraint strategy for evolution search, which can restrict search direction on the region near the GNNs architectures that have a good performance and reserve probability of exploring other architectures at the same time.

Parallel Search. Using wheel selection strategy to sample $k$ GNNs architectures from sharing population as the parents for each worker. Selecting $m$ architecture components to perform random mutation based on architecture mutation selection probability vector $\overline{P}$. Each worker will generate $k$ child architectures and evaluate fitness for these child architectures on the dataset $D_{val}$.

Sharing Children & Updating. Merging $N \times k$ child architectures generated by $N$ workers as the sharing children. Then, putting child architecture into sharing population if the fitness of child architecture is greater than threshold $F$, which $F$ is the average fitness of top $n$ GNNs architectures in sharing population.

| Table 2: Benchmark Citation Networks. |
|--------------------------------------|
| Items       | Citeseer | Cora | Pubmed |
| #Nodes      | 3,327    | 2,708| 19,717 |
| #Edges      | 4,732    | 5,429| 44,338 |
| #Features   | 3,703    | 1,433| 500    |
| #Classes    | 6        | 7    | 3      |
| #Training Nodes | 120  | 140  | 60     |
| #Validation Nodes | 500  | 500  | 500    |
| #Testing Nodes | 1000 | 1000 | 1000   |

3 EXPERIMENTS

In this section, we conduct extensive experiments on three benchmark datasets to evaluate the effectiveness of GraphPAS for node classification task.

3.1 Datasets

In this work, we use three benchmark citation networks including Citeseer, Cora and Pubmed [9] which nodes represent documents and edges denote citation links. Dataset characteristics of statistics are summarized in Table 2.

3.2 Baseline Methods

In this study, we compare GNNs architecture identified by our model against baselines including the state-of-art handcrafted architectures and NAS methods for node classification task. Handcrafted architectures are included: Chebyshev [10], GCN [11], GraphSAGE [6], GAT [7], LGCN [12] and LC-GCN/GAT [13]. NAS methods are involved: Auto-GNN [14], GraphNAS [3] and Gene-GNN [4].
3.3 Parameter Settings
The GNNs architecture designed by GraphPAS are implemented by the Pytorch-Geometric library[15].

Hyper-parameters of search algorithm: The parallel workers N as 4, the number of layer l is fixed to 2, the size of random initial population M as 100 for each worker, the number of sharing population n and the number of parents k as 20, the number of mutation m is set to 1, 2, 3, 4 for each worker separately, finally the search_epoch as 20.

Hyper-parameters of GNNs: For training one GNNs architecture designed by GraphPAS, the training epoch as 300, and we used the L2 regularization with \( \lambda = 0.0005 \), the learning rate \( r = 0.005 \), and dropout probability \( p = 0.6 \), we utilized Adam SGD to minimize cross-entropy loss function.

3.4 Test Performance Comparison
To validate the performance of our model, we compare the GNNs models discovered by GraphPAS with handcrafted models and those designed by other search approaches. Test accuracy on the node classification task is summarized in Table 3, where the best performance is highlighted in bold.

| Category         | model           | Cora Accuracy | Citeceer Accuracy | Pubmed Accuracy |
|------------------|-----------------|---------------|-------------------|---------------|
| Handcrafted      | Chebyshev       | 81.2%         | 68.8%             | 74.4%         |
| Architectures    | GCN             | 81.5%         | 70.3%             | 79.0%         |
|                  | GAT             | 83.0±0.7%     | 72.5±0.7%         | 79.0±0.3%     |
|                  | LGCN            | 83.3±0.5%     | 73.0±0.6%         | 79.5±0.2%     |
|                  | LC-GCN          | 82.9±0.4%     | 72.3±0.8%         | 80.1±0.4%     |
|                  | LC-GAT          | 83.5±0.4%     | 73.8±0.7%         | 79.1±0.5%     |
| NAS Method       | Auto-GNN        | 83.6±0.3%     | 73.8±0.7%         | 79.7±0.4%     |
|                  | GraphNAS        | 83.7±0.4%     | 73.5±0.3%         | 80.5±0.3%     |
|                  | Gene-GNN        | 83.8±0.5%     | 73.5±0.8%         | 79.2±0.6%     |
|                  | GraphPAS        | 83.9±0.4%     | 73.9±0.3%         | 80.6±0.3%     |

Considering the deviation of testing accuracy, the test performance of GraphPAS is averaged via randomly initializing the optimal GNNs architecture 20 times, and those of handcrafted architectures as well as NAS methods are reported directly from their works. In general, the NAS methods can achieve better results than handcrafted methods, which demonstrated that it is effective for NAS methods to design good GNNs architecture for the given graph data. Especially, the GraphPAS get the best performance on all three benchmark datasets. The reason is that the handcrafted models need a lot of tuning of GNNs architecture parameters by manual method for different graph datasets, which is hard to obtain an optimal model. On the contrary, the NAS methods can automatically search the GNNs architecture with the ability that makes the performance of GNNs model better for given graph datasets, which can gradually optimize the GNNs model performance with litter human intervention.

3.5 Search Efficiency Comparison
To further investigate the effectiveness of the GraphPAS, we compare the progression of top-10 averaged validation performance of GraphNAS, random search and GraphPAS.

For each search approach, 2,000 GNNs architectures are explored in the same search space. As shown in Figure 3, GraphPAS is more efficient to find the well-performed GNNs architectures during the search process. Because GraphPAS can search the GNNs architecture in different areas simultaneously in the huge search space based on parallel ability. Then, the sharing mechanism can enhance the diversity of population, which is beneficial to improve the search efficiency for evolution algorithm. Finally, GraphPAS utilizes architecture information entropy to dynamically compute mutation selection probability that can restrict search direction effectively for achieving good GNNs architecture faster.

4 CONCLUSION
In this paper, we propose a parallel architecture search framework GraphPAS for graph neural networks. GraphPAS utilizes parallel ability and restricts search direction effectively to speed up search process and achieve optimal GNNs architecture automatically. Experiment results on the three benchmark datasets show that GraphPAS can obtain GNNs architecture that outperforms the state-of-art method with efficiency and accuracy.

5 ACKNOWLEDGMENTS
This work was supported by the National Natural Science Foundation of China (61873288), Zhejiang Lab (2019K60AB01), Hunan Key Laboratory for Internet of Things in Electricity (2019TP1016), and CAAI-Huawei MindSpore Open Fund.
REFERENCES

[1] Feng Yu, Yanqiao Zhu, Quan Liu, Shu Wu, Liang Wang, and Tieniu Tan. Tagnn: Target attentive graph neural networks for session-based recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1921–1924, 2020.

[2] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. Journal of Machine Learning Research, 20(55):1–21, 2019.

[3] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In Proceedings of the 29th International Joint Conference on Artificial Intelligence, volume 20, pages 1403–1409, 2020.

[4] Min Shi, David A Wilson, Xingquan Zhu, Yu Huang, Yuan Zhuang, Jianxun Liu, and Yuefei Tang. Evolutionary architecture search for graph neural networks. arXiv preprint arXiv:2009.10199, 2020.

[5] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In International Conference on Machine Learning, pages 7134–7143, 2019.

[6] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Annual Conference on Neural Information Processing Systems, pages 1024–1034, 2017.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lions, and Yoshua Bengio. Graph attention networks. In 6th International Conference on Learning Representations, 2018.

[8] Michael Hahsler, Bettina Grün, and Kurt Hornik. Introduction to arules–mining association rules and frequent item sets. ACM SIGKDD Explorations, 2(4):1–28, 2007.

[9] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. AI magazine, 29(3):95–103, 2008.

[10] Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems, pages 3837–3845, 2016.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In 5th International Conference on Learning Representations, 2017.

[12] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1416–1424, 2018.

[13] Bingbing Xu, Junjie Huang, Liang Hou, Huawei Shen, Jimhua Gao, and Xueq Cheng. Label-consistency based graph neural networks for semi-supervised node classification. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1897–1900, 2020.

[14] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. arXiv preprint arXiv:1909.03184, 2019.

[15] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.