Auto-GNAS: A Parallel Graph Neural Architecture Search Framework

Jiamin Chen, Jianliang Gao, Yibo Chen, Babatounde Moctard Oloulade, Tengfei Lyu, and Zhao Li

Abstract—Graph neural networks (GNNs) have received much attention as GNNs have recently been successfully applied on non-euclidean data. However, artificially designed graph neural networks often fail to get satisfactory model performance for a given graph data. Graph neural architecture search effectively constructs the GNNs that achieve the expected model performance with the rise of automatic machine learning. The challenge is efficiently and automatically getting the optimal GNN architecture in a vast search space. Existing search methods serially evaluate the GNN architectures, severely limiting system efficiency. To solve these problems, we develop an **Auto**matic **G**raph **N**eural **A**rchitecture **S**earch framework (Auto-GNAS) with parallel estimation to implement an automatic graph neural search process that requires almost no manual intervention. In Auto-GNAS, we design the search algorithm with multiple genetic searchers. Each searcher can simultaneously use evaluation feedback information, information entropy, and search results from other searchers based on sharing mechanism to improve the search efficiency. As far as we know, this is the first work using parallel computing to improve the system efficiency of graph neural architecture search. According to the experiment on the real datasets, Auto-GNAS obtain competitive model performance and better search efficiency than other search algorithms. Since the parallel estimation ability of Auto-GNAS is independent of search algorithms, we expand different search algorithms based on Auto-GNAS for scalability experiments. The results show that Auto-GNAS with varying search algorithms can achieve nearly linear acceleration with the increase of computing resources.

Index Terms—Neural architecture search, parallel search, graph neural network

1 Introduction

Graph data is widespread in the real world, such as document networks, knowledge graphs, social networks, and biological networks [1], [2]. Graph data is an abstract structure representing the relationship between objects, including node sets, edge sets, and attribute features. Graph data can express rich relationships between objects. This relationship is not only structural but also a semantic relationship. Potential information on mining graph data has significant academic and application value. In recent years, graph neural networks (GNNs) have received extensive attention from many researchers as an effective method for mining the potential information of graph data [3], [4]. The classic graph neural network models, including GCN [5] and GAT [6], have achieved good results in graph data mining. However, to obtain expected performance on a given graph dataset, it is necessary to design the architecture of graph neural networks based on the specific characteristics of graph datasets, and it usually requires a lot of manual work and domain expert experience.

In recent years, with the rise of automatic machine learning and graph neural networks, researchers have used the neural architecture search (NAS) method [7], [8], [9], [10] to solve the challenges of graph neural network architecture design and make some breakthroughs. In brief, the NAS method is to design a search algorithm to automatically sample the neural network architecture in a predefined search space, which contains the architecture components required by the neural network. After, using the estimation strategy to evaluate the model performance based on the sampled neural network architecture and generate a piece of feedback information to the search algorithm for getting better GNN architecture. Graph neural architecture search uses the NAS method to identify the optimal graph neural network architecture for the given graph data. In the progress of graph neural architecture search, the system time overhead is mainly composed of three sections, sampling GNN architectures from the search space, estimating the sampled GNN architectures for generating the feedback information, and search algorithm iteration based on the feedback information. To study the proportion of time spent on each part of graph neural architecture search, we conduct a time-consuming experiment based on GraphNAS [8], which is the classic work for graph neural architecture search and uses a serial evaluation manner to estimate the sampled GNN architectures. The result shows that the total

- *Jiamin Chen, Jianliang Gao, Babatounde Moctard Oloulade, and Tengfei Lyu are with the School of Computer Science and Engineering, Central South University, Changsha, Hunan 410083, China. E-mail: {chenjiamin, gaojianliang, oloulademoctard, tengfeilyu}@csu.edu.cn.*
- *Yibo Chen is with State Grid Hunan Electric Power Company Limited, Changsha 410007, China. E-mail: chenyibo8224@gmail.com.*
- *Zhao Li is with the Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, Hangzhou, Zhejiang 311121, China. E-mail: lizhao.lz@alibaba-inc.com.*

Manuscript received 2 Sept. 2021; revised 17 Nov. 2021; accepted 10 Feb. 2022. Date of publication 16 Feb. 2022; date of current version 23 May 2022.
This work was supported in part by the National Natural Science Foundation of China under Grants 61873288 and 61836016, in part by State Grid Hunan Electirc Power Company under Grant 5216A6210075, and in part by Hunan Key Laboratory for Internet of Things in Electricity, CAAI-Huawei MindSpore Open Fund, and Hunan Provincial Science and Technology Program under Grant 2021JJ30055.
(Corresponding author: Jianliang Gao.)
Recommended for acceptance by A.J. Peña, M. Si and J. Zhai.
Digital Object Identifier no. 10.1109/TPDS.2022.3151895

time consuming of estimation is average 1.7×10^3 times that of search algorithm iteration and 2.4×10^3 times that of sampling GNN architectures on the Cora dataset [11]. It accounts for 99% of the total system time overhead on average. The primary time cost is concentrated in the estimation progress. Increasing the speed of estimation is the most important and effective way to improve graph neural architecture search system efficiency.

For the architecture designing of convolutional neural networks (CNNs) and recurrent neural networks (RNNs), many NAS works [12], [13] have successfully applied parallel computing capabilities to accelerate the estimation process of sampled CNNs and RNNs. As the neural network architecture becomes more and more complex and the size of data is quickly grown, the scale of search space and the cost of evaluating a sampled neural architecture increases rapidly for NAS. Using parallel computing to speed up the estimation process of NAS is becoming more and more critical. However, in the graph neural network architecture search field, no NAS framework currently provides parallel computing abilities to increase estimation efficiency. Simultaneously, due to the differences in network architecture and processed data structure between CNNs and GNNs, the parallel framework of NAS in the CNN architecture search can not be migrated to the GNN architecture search directly. GraphNAS [8] and Auto-GNN [9] utilize reinforcement learning to construct search algorithms for graph neural architecture search. This search algorithm can be continuously trained using the feedback information generated by the estimation strategy. The training process enables the search algorithm to gain the ability to sample the GNN architectures that have good performance. Genetic-GNN [10] proposes a graph neural network architecture search algorithm based on a genetic algorithm. The advantage of the genetic algorithm is that it has no training parameters, and the algorithm design is simple. However, these methods only use estimation feedback signals to iterate the search algorithm during the search process. For improving search efficiency, it is necessary to use additional effective information to constrain the search direction in the vast search space.

Graph neural architecture search is a useful way to design an effective graph neural network architecture for the given graph data. However, it is imperative to construct an efficient graph neural architecture search process. Two challenges that limit system efficiency need to be overcome. There is no general and customizable parallel framework based on graph neural architecture search in the current research, using parallel computing to speed up estimation efficiency. The first challenge is how to construct an efficient and flexible parallel estimation framework for graph neural architecture search. Most of the current search methods only use estimation feedback information to train search algorithms. In particular, the work based on genetic search methods [10] uses evaluation feedback as fitness to guide the search direction. However, when we use genetic algorithms to construct a parallel search process, if we only use fitness to guide the search direction, the parallel genetic search manner is equivalent to the serial search manner in terms of search performance[14]. The second difficulty is that, in addition to fitness, whether we can provide additional effective information for parallel genetic search to improve search performance for better search efficiency.

In this paper, we develop an open-source parallel search framework named Auto-GNAS for graph neural architecture search, automatically identifying server GPU and CPU resources to complete the parallel estimation process. Auto-GNAS constructs multiple genetic searchers with different mutation intensities and uses various information to improve search efficiency. At the same time, Auto-GNAS adopts sharing mechanism to ensure effective communication among each genetic searcher. We evaluate Auto-GNAS using multiple graph tasks. Compared to handcrafted GNN architectures [5], [6], [15], [16], [17], [18] and other NAS methods [8], [9], [10], Auto-GNAS can get competitive performance in node classification task. In comparison with the search efficiency of other NAS methods, Auto-GNAS has better search efficiency in multiple graph tasks. Furthermore, it is proved in the scalability experiment that the Auto-GNAS with different search algorithms obtains a nearly linear speedup.

Our work focuses on automatic graph neural architecture search using parallel computing. As far as we know, this is the first parallel graph neural architecture search framework. The main contributions are as follows:

- We propose and implement a general framework for graph neural architecture search named Auto-GNAS.[1] It is the first parallel graph neural architecture search framework and solves the problem of search efficiency for different graph tasks, such as node classification, graph classification, and link prediction.
- We propose an efficient genetic search algorithm for Auto-GNAS. This search algorithm builds multiple genetic searchers with different mutation intensities based on sharing mechanisms and uses various information to guide the search direction to improve search efficiency.
- We evaluate Auto-GNAS using different tasks to demonstrate that Auto-GNAS is more efficient than other NAS methods. Furthermore, scalability experiments prove that Auto-GNAS can produce a nearly linear acceleration effect to graph neural architecture search systems.

2 Background and Motivation

2.1 Graph Neural Network

There are two types of GCNs, spectral-based [19], and spatial-based [20]. The spectral-based method needs to operate on the entire graph. It is not easy to parallel and hardly scale to big graphs. However, the spatial-based method is flexible to aggregate feature information between neighbor nodes. The GNNs mentioned in this work represent spatial-based graph convolutional neural networks. Table 1 shows the notations used in this paper.

The main idea of GNNs can be abstracted into the following processes. First, calculating the correlation coefficient $r_{ij}^{(l)}$ between the central node v_i and each neighbor node $v_j \in N(v_i)$. Then, aggregating the neighbor node v_j representation $h_j^{(l)}$ for the central node v_i with correlation coefficient $r_{ij}^{(l)}$ to get the representation $h_{agg}^{(l)}$. Next, using function $f^{(l)}$ to process the representation $h_i^{(l)}$ of central node v_i and $h_{agg}^{(l)}$ as

1. https://github.com/AutoMachine0/Auto-GNAS

TABLE 1
Main Notations In This Paper

Notations	Descriptions
v	A node in the graph.
$N(v)$	The neighbors of a node v.
e_{ij}	The edge of v_i and v_j
l	The number of GNN layer
r_{ij}	The correlation coefficient between v_i and v_j
h	The representation of node
f	The function for processing node representation
W	The learnable matrix
att	The attention function
agg	The aggregation function
act	The activation function
$head$	The number of multi-head
dim	The hidden dimension
S	The search space of GNN architecture components
s	The GNN architecture
E	The estimation metric
D_{val}	The validation dataset of graph
D_{train}	The training dataset of graph
θ	The model parameters of GNNs
L	The loss function of GNNs
N^*	The set of positive integers

the representation $h_f^{(l)}$. At last, reducing dimension of $h_f^{(l)}$ by a learnable matrix $W^{(l)}$ before using non-linear transformation for getting the next layer central node representation $h_i^{(l+1)}$. The formulation of the node hidden representation is

$$h_i^{(l+1)} = ACT^{(l)}(W^{(l)} \cdot f^{(l)}(h_i^{(l)}, AGG\{h_j^{(l)}, \forall v_j \in N(v_i)\})). \quad (1)$$

2.2 Graph Neural Architecture Search

GNN architecture usually consists of different components such as attention function, aggregation function, activation function, etc. Each component has different values, for example, sum, $mean$, and max are the values of aggregation function. Different combinations of values among components constitute different GNN architectures, and the combination space is the search space. Graph neural architecture search aims to explore the optimal combination from search space for maximizing model performance. As shown in Fig. 1, a search space S composed of three GNN architecture components mentioned above. Search algorithm samples a combination from search space S as a GNN architecture s, for instance, $\{gat, sum, tanh\}$. The GNN architecture s is further evaluated by performance estimation strategy. The search algorithm uses the performance as the feedback signal to train search model parameters or guide search direction.

2.3 Problem Formulation

Given a search space S of graph neural architecture, a validation dataset D_{val}, a train dataset D_{train}, and performance estimation metric E. The target is to gain the optimal graph neural architecture s_{opt} from search space S, which is trained on set D_{train} based on loss function L and achieves the best performance on set D_{val}. Mathematically, it is expressed as follows:

$$\begin{aligned} s_{opt} &= argmax_{(s \in S)} E(s(\theta^*), D_{val}) \\ \theta^* &= argmin_{\theta}\ L(s(\theta), D_{train}). \end{aligned} \quad (2)$$

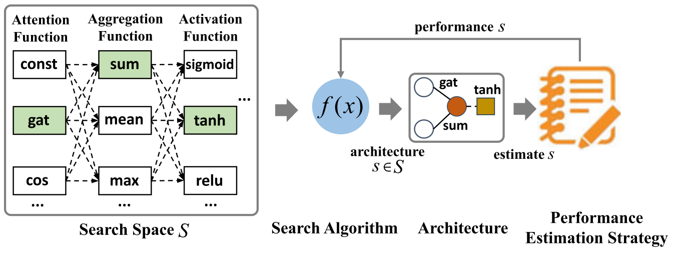

Fig. 1. Illustration of graph neural architecture search.

2.4 Challenge of Efficiency

Estimation Efficiency. GNN architecture performance evaluation is a critical process for graph neural architecture search, which generates the validation set accuracy of GNN architecture as the feedback signals for the search algorithm iteration. The search algorithm uses feedback information to train search model parameters or guide the search direction for getting better GNN architecture in the following sampling. GraphNAS [8], Auto-GNN [9], and Genetic-GNN [10] use serial strategy to evaluate GNN architecture to obtain feedback information that only considers one GNN architecture at one search epoch. However, to sample the optimal GNN architecture for a given graph data, many feedback signals are needed for search algorithm iteration, which means a large number of GNN architectures need to be evaluated during the search process for generating the feedback information. When the time cost of evaluating a GNN architecture increases, the serial estimation will seriously limit the system's efficiency.

Search Efficiency. The scale of search space is enormous. For example, there is about 10^8 possibility of combinations for two layers GNN architecture [8]. Exploring all possible GNN architectures to get the optimal GNN architecture in the vast search space is too time-consuming and insufferable. The feedback signals generated by the estimation strategy are effective information to help search algorithms[8], [9], [10] reduce the exploration rate in the large search space. However, when we need to process a large-scale graph dataset, such as the graph classification task in biological networks, it will significantly increase the time cost of evaluating a GNN architecture. It is necessary to improve search efficiency further to reduce the total number of sampled GNN architectures in the vast search space for getting the target GNN architecture that meets expectations. In addition to the feedback information generated by GNN architecture evaluation, we need to construct additional effective information for search algorithm iteration to improve search efficiency.

3 Auto-GNAS

This section will introduce the framework, the workflow, the search space, the search algorithm, and the parallel estimation of Auto-GNAS.

3.1 Framework Overview

The framework of Auto-GNAS is shown in Fig. 2. We abstract five functional modules according to the process of graph neural architecture search. The modules are relatively independent and are related by API and dynamic loading function. The framework of Auto-GNAS is very friendly for users

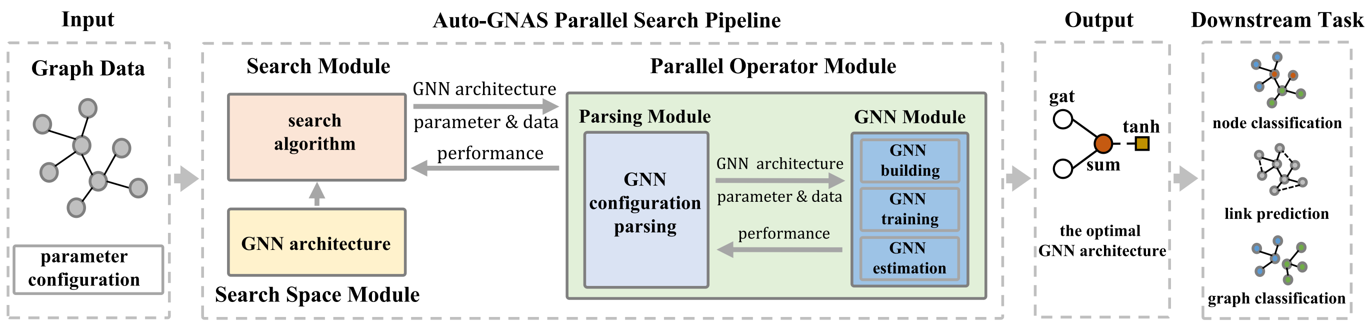

Fig. 2. Auto-GNAS framework.

to understand and customize functions. The *Search Module* realizes the search function and search management of GNN architecture. It receives graph data and configuration parameters then uses a search algorithm to sample GNN architecture from a predefined search space. The *Search Space Module* contains a predefined GNN architecture search space, and users can customize the search space for particular tasks. The *Parsing Module* is responsible for establishing the communication between the *Search Module* and the *GNN Module*, at the same time, it implements the parsing of the GNN training parameters configuration. *GNN Module* implements GNN model building, training, and estimation processes for different downstream graph tasks based on sampled GNN architecture. This module lets users customize downstream tasks, loss functions, optimization functions, and evaluation strategies according to their requirements. The *Parallel Operator Module* encapsulates the *Parsing Module* and *GNN Module* to form a simple and efficient API, which can make full use of GPU and CPU resources to realize the parallel evaluation process of GNN architecture.

A typical Auto-GNAS workflow is as follows. Users can define their graph data according to the data format of PyG [21], and set the new search configuration for their special graph task. As shown in Fig. 3, a user only needs to execute a few lines of code to complete the automatic search for GNN architectures by default configuration. When the top-level API receives a user call, first, the graph data and configuration parameters will be passed to the *Search Module*, and the search algorithm will sample the GNN architecture based on the search parameters and search space. Next, sampled GNN architectures, GNN training parameters and graph data are passed to the *GNN Module* through the *Parallel Operator Module* and the *Pasring Module*. And then, the *GNN Module* will build and train a GNN model, at the same time, it will evaluate the model performance using the validation set based on the downstream task to generate the feedback for *Parsing Module*. After, the *Parsing Module* will pass the feedback performance to *Search Module* for the updating of the search algorithm. Finally, the top-level API will return the optimal GNN architecture as an output for the user until the search stop condition is met. The Auto-GNAS framework provides users with many cases and documents to help users get started quickly.

```
from autognas.parallel import ParallelConfig
from autognas.auto_model import AutoModel
from autognas.datasets.planetoid import Planetoid
#open parallel estimation
ParallelConfig(Ture)
#obtain default graph datasets
graph = Planetoid("cora").data
# start GNN architecture search for node classification
AutoModel(graph)
```

Fig. 3. Implementing GNN architecture search based on a given data.

3.2 Search Space

It is flexible to customize the search space of the GNN architecture component based on the *Search Space Module*. Typically, the search space involves five architecture components for one layer of GNNs:

Attention Function(att). The representation of each node depends on the aggregation of the features of its neighbor nodes, but different neighbor node feature has different importance for the central node [22]. The attention operator aims to compute the correlation coefficient r_{ij} for each edge relationship e_{ij} between the nodes v_i and v_j. The types of attention functions are shown in Table 2.

Aggregation Function(agg). When all correlation coefficients r_{ij} are gotten by attention function for the central node v_i in $N(v_i)$, in order to represent central node hidden state h_i better, the aggregation function will gather the feature representation from its neighbor nodes with correlation coefficient r_{ij} [16]. The aggregation operators include *mean*, *max*, and *sum* in this work.

Multi-Head$(head)$. Instead of using the attention operator once in one layer of GNNs, GAT [6] shows that multiple independent attention operators can enhance the robustness of the learning process. The parameter multi-heads decide the number of independent attention operators in one layer of GNNs. We set this parameter within the set: $\{1, 2, 4, 6, 8\}$.

HiddenDimension(dim). The node hidden representation h_i needs to multiply a trainable matrix $W^{(l)}$ in each layer of GNNs, l is the number of GNN layers, which is used to make

TABLE 2
Attention Functions of Search Space S

Attention Functions	values
gat	$r_{ij}^{gat} = reaky\_relu(W_c * h_i + W_n * h_j)$
gcn	$r_{ij}^{gcn} = 1/\sqrt{d_i d_j}$
cos	$r_{ij}^{cos} = < W_c * h_i, W_n * h_j >$
const	$r_{ij}^{const} = 1$
sym-gat	$r_{ij}^{sym} = r_{ij}^{gat} + r_{ji}^{gat}$
linear	$r_{ij}^{lin} = tanh(sum(W_c * h_i))$
gene-linear	$r_{ij}^{gen} = W_b * tanh(W_c * h_i + W_n * h_j)$

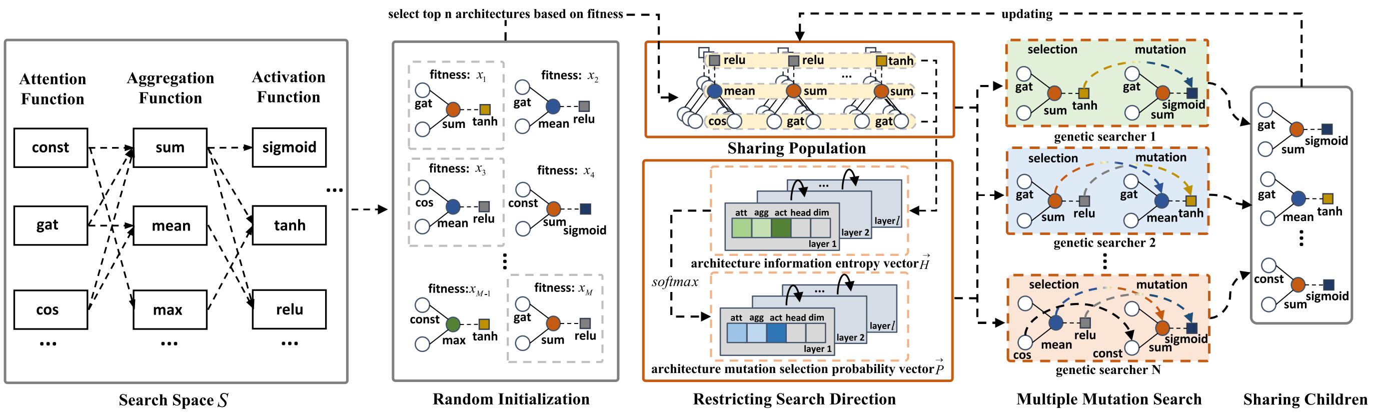

Fig. 4. Search algorithm based on multiple genetic searchers and various information constraints.

the hidden state denser for enhancing feature representation. In this work, the set of hidden dimensions is $\{8, 16, 32, 64, 128, 256, 512\}$.

Activation Function(act). For the target of smoothing hidden distribution, we need a non-linear transformer to process the hidden representation h_i before getting the final output. The choice of activation functions in our search space is listed as follows: $\{$ $tanh$, $sigmoid$, $relu$, $linear$, $relu6$, elu, $leaky\_relu$, $softplus$ $\}$.

In this work, the GNN architecture is designed as a stack, which each layer of GNN architecture consists of the five components mentioned above. We give an example of two-layer GNN architecture as follows. The GNN architecture is described as an order list, and each element represents an architecture component value.

$$\overbrace{\begin{array}{ccccc} \text{att} & \text{agg} & \text{act} & \text{head} & \text{dim} \\ (\text{gcn}, & \text{sum}, & \text{tanh}, & 6, & 16, \end{array}}^{\text{first layer}} \quad \overbrace{\begin{array}{ccccc} \text{att} & \text{agg} & \text{act} & \text{head} & \text{dim} \\ \text{gat}, & \text{mean}, & \text{relu}, & 2, & 64) \end{array}}^{\text{second layer}}$$

There will be 5880^l combinations in the search space, where l is the number of GNN layers. To prevent the search space from being too large, we fix l to two in our experiment, which reduces the search space to 3.45×10^8 combinations.

3.3 Search Algorithm

With the analysis of GNN architecture features, we find that different architecture component values have different frequencies of occurrence in the GNN architectures that can get good performance on a given dataset. Inspired by the theory that association algorithm mines frequent itemset [23], we use information entropy to measure the correlation between GNN architecture component values and good performance. We design multiple genetic searchers with different mutation intensities for GNN architecture sampling to improve search efficiency. Each genetic searcher can simultaneously use information entropy and estimation feedback signal to constrain the search direction. The search algorithm is shown in Fig. 4, and it consists of five processes. We will introduce the five processes in this section.

Random Initialization. Sampling M GNN architectures randomly in the vast search space as the initial population and estimating them on the dataset D_{val}. We define validation set accuracy as the fitness x_i for each GNN architecture.

Sharing Population. Creating multiple genetic searchers simultaneously for searching, and the sharing mechanism is designed to make each genetic searcher utilize the GNN architectures generated by others. It can improve the diversity of the population to help search algorithm to sample better GNN architecture faster. Ranking GNN architectures based on fitness and extracting top n GNN architectures as the sharing population.

Restricting Search Direction. Computing information entropy $h(c_i)$ for each architecture component by Eq. (3) based on the frequency distribution of architecture component value in sharing population.

$$h(c_i) = -\sum_j f(v_j) log_2 f(v_j)$$
$$i \in [1, 5 \times l], l \in N^*; \tag{3}$$

where $h(c_i)$ represents the information entropy of the ith component of GNN architecture, v_j denotes the jth value of the ith component, which appears in sharing population, counting v_j frequencies of occurrence as $f(v_j)$, l is the number of GNN layers.

Then, acquiring architecture information entropy vector $\vec{H} = [h(c_1), h(c_2), \ldots, h(c_i)]$. Finally, based on Eq. (5) to get the architecture component mutation-selection probability p_i. And $\vec{P} = [p_1, p_2, \ldots, p_i]$ is represented as the architecture mutation-selection probability vector.

$$p_i = softmax(h(c_i)) \tag{4}$$

$$i \in [1, 5 \times l], l \in N^*. \tag{5}$$

The architecture mutation-selection probability vector $\vec{P}$ is a soft constraint strategy to limit search direction. It can restrict search direction on the region near the GNN architectures with good performance and simultaneously reserve the probability of exploring other areas in the vast search space.

Multiple Mutation Search. To further improve the diversity of the population, we construct N genetic searchers to sample the child architectures simultaneously. Each genetic searcher simultaneously selects k parent architectures from the sharing population using a wheel strategy. And then,

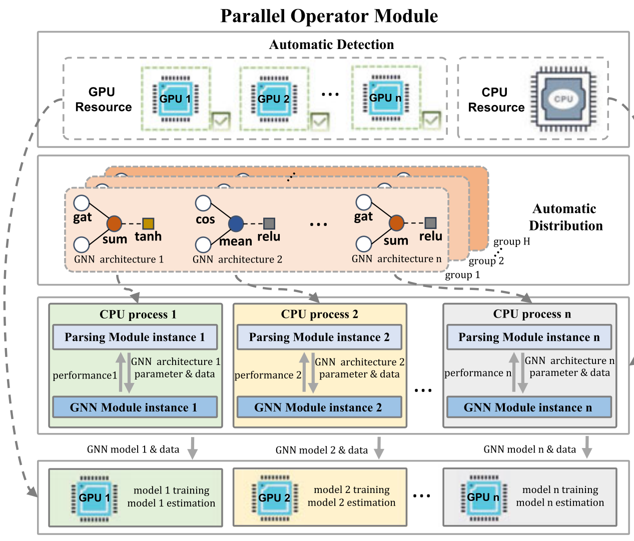

Fig. 5. Parallel estimation based on Parallel Operator Module.

each genetic searcher uses different mutation intensities, which m architecture components will mutate randomly for one GNN architecture, to generate k new child architectures based on mutation-selection probability vector $\vec{P}$.

Sharing Children & Updating. Merging $N \times k$ child architectures generated by N different genetic searchers as the sharing children, and estimating them for getting the fitness y_i. Then, putting child architecture into sharing population if the fitness y_i is greater than threshold F, which F is the average fitness of top t GNN architectures in the sharing population. Algorithm 1 presents the search logic in detail.

3.4 Parallel Estimation

Parallel evaluation is the most important manner of improving system efficiency for graph neural architecture search. Auto-GNAS contains an independent parallel evaluation function called *Parallel Operator Module*. It can help different search algorithms to quickly implement parallel evaluation of GNN architectures based on a few simple APIs. *Parallel Operator Module* consists of automatic detection and distribution. Fig. 5 shows the parallel estimation process of GNN architectures based on *Parallel Operator Module*. Users can implement parallel evaluation of GNN architectures with a few lines of code as Fig. 6.

Automatic Detection. First, this process will automatically identify the GPU and CPU resources on the server. Then, it will instantiate the *Parsing Module* and *GNN Module* for each resource. The *Parsing Module* contains the communication function and parsing of the GNN training parameters. The *GNN Module* consists of GNN model building, training, and estimation. To better understand the automatic detection process, we use the following example to illustrate it. If there are n GPU devices on the server, it will try to start n CPU processes. Each CPU process will instantiate the *Parsing Module*, *GNN Module* and bind a GPU computing resource for preparing of automatic distribution.

Automatic Distribution. This process will automatically divide the GNN architectures generated by *Search Module* into H groups. Furthermore, each group contains the number of GNN architectures less than or equal to the number of GPU. Next, it will automatically distribute one GNN

```
from autognas.parallel import ParallelOperator,ParallelConfig
from autognas.datasets.planetoid import Planetoid
#open parallel estimation
ParallelConfig(Ture)
#obtain default graph datasets
graph = Planetoid("cora").data
#instantiate ParallelOperator object
parallel_obj = ParallelOperater(graph)
#obtain sampled GNN architectures from search algorithm
gnn_architecture_list = UserCustomSearchAlogrithm()
#GNN architecture parallel estimation
result = parallel_obj.estimation(gnn_architecture_list)
```

Fig. 6. Implementing GNN architecture parallel estimation.

architecture for each CPU process based on one group. Subsequently, each *GNN Module* instance will construct a GNN model and training parameters and pass the GNN model and graph data into the bound GPU resources to complete the model training and estimation. Each *Parsing Module* will pass the estimation performance to *Search Module*.

Algorithm 1. Search Algorithm

Input:
search space S, validation set D_{val}, training set D_{train}, *search epoch*, initial population M, sharing population n, parents k, number of genetic searchers N, mutation intensity m, top t GNN architectures for threshold F.

Output:
The optimal GNN architecture s^*.

```
1:  // initialization
2:  $pop_i \leftarrow random\ initialization(M, S)$
3:  $fitness_i \leftarrow estimation(pop_i, D_{train}, D_{val})$
4:  $pop_{sp}, fitness_{sp} \leftarrow select\_top(n, fitness_i)$
5:  $p \leftarrow mutation\_computing(pop_{sp})$
6:  // searching
7:  for $i \leftarrow 0$ to search epoch do
8:    $children_{sc} \leftarrow \emptyset$
9:    for $j \leftarrow 1$ to $N$ do
10:     $parents \leftarrow select\_wheel(k, pop_{sp}, fitness_{sp})$
11:     $children \leftarrow mutation(parents, p, m, S)$
12:     $children_{sc}.append(children)$
13:   end for
14:   $fitness_{sc} \leftarrow estimation(children_{sc}, D_{train}, D_{val})$
15:   $F \leftarrow average(t, fitness_{sp})$
16:   for $child, fitness \leftarrow children_{sc}, fitness_{sc}$ do
17:     if $fitness > F$ then
18:       $pop_{sp}.append(child)$
19:       $fitness_{sp}.append(fitness)$
20:     end if
21:   end for
22:   $p \leftarrow mutation\_computing(pop_{sp})$
23: end for
24: // architecture deriving
25: Select top $K$ architectures from $pop_{sp}$ based on $fitness_{sp}$.
26: Re-train the $K$ architectures for $R$ times.
27: Select the best $s^*$ from the $K$ architectures.
```

Moreover, when the *Parallel Operator Module* does not identify the GPU resources on the server for GNN model training and estimation. It will try to start multiple CPU processes based on CPU logic cores instead of GPU to perform the parallel GNN architecture evaluation process.

TABLE 3
Benchmark Citation Networks

Dataset	Nodes	Edges	Node Features	Classes
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Pubmed	19,717	44,338	500	3

TABLE 4
Benchmark Molecular and Biological Networks

Dataset	Graphs	Avg. Nodes	Node Features	Classes
AIDS	2,000	16.2	38	2
Proteins	1,113	39.06	3	2

4 Experiments

In this section, we first introduce the experiment setup. Then we conduct extensive performance, efficiency, and analysis experiments to evaluate the effectiveness and the scalability of Auto-GNAS.

4.1 Experiment Setup

In this part, we show our experiment setup, including testbed, datasets, configurations, and evaluation metrics.

Testbed. We conduct our experiments on a server with 4 NVIDIA GeForce RTX 2080 Ti GPU clusters, and the server has an Intel Xeon 4114 CPU @2.20GHz with 40 CPU logical cores and 256 GB RAM. All experiments run on Ubuntu 16.04.7 LST.

Datasets. In our experiments, we use three benchmark citation networks Citeseer, Cora, and Pubmed [11] for the node classification task and conduct a link prediction task on Citeseer, Cora. For the graph classification task, we use AIDS and Proteins networks[24]. The Datasets characteristics of statistics are summarized in Tables 3 and 4.

Configurations. The GNN models sampled by Auto-GNAS are implemented based on the Pytorch-Geometric library[2] [21], and the parallel ability of Auto-GNAS is supported by the Ray library[3] [25]. Through many tests and the sensitivity analysis of search parameters, we select the search parameters that enable Auto-GNAS to obtain the best result for performance comparison experiments and efficiency comparison experiments. For the node classification task, the number of GNN layer l is fixed to 2, the size of random initial population M as 400, the number of initial sharing population n and the number of parents k as 20, the number of genetic searchers N as 4, the mutation intensity m is set to 1, 2, 3, 4 for each genetic searcher respectively, finally the $search\_epoch$ as 20. For training one GNNs architecture sampled by Auto-GNAS, the training $epoch$ as 300, and we used the $L2$ regularization with λ = 0.0005, the learning rate r = 0.005, and dropout probability d = 0.6. For the Citeseer, Cora, and Pubmed datasets, we use 120 nodes, 140 nodes, 60 nodes for training, 500 nodes for validation, and 1,000 nodes for testing.

Evaluation Metrics. The node classification task is one of the typical graph representation learning tasks. The goal of the task is to evaluate the representation ability of the GNN model for node features and use node representation to predict the type of node.

We use test set accuracy for performance evaluation, which is commonly used as performance evaluation in node classification. We compare the test accuracy of node classification with handcrafted architectures and NAS methods. Handcrafted architectures include Chebyshev [15], GCN [5], GraphSAGE [16], GAT [6], LGCN [17] and LC-GCN/GAT [18]. NAS methods involve Auto-GNN [9], GraphNAS[4] [8] and Gene-GNN [10].

For efficiency evaluation, we use the progression of top-10 averaged validation set accuracy of GNN architecture to measure the efficiency of the search algorithm, which effectively evaluates the search efficiency. The speedup ratio is a general way to measure scalability. It is defined as T_s/T_m, where T_s is the running time of the system with a single resource, and T_m is the running time of system with multiple resources. We compare the efficiency of different search methods, including the GraphNAS [8], random search [26], genetic algorithm search [27], and Auto-GNAS. We expand different search algorithms for Auto-GNAS, then test the scalability with different GPU resources for Auto-GNAS.

4.2 Performance Comparison

To validate the performance of the GNN model, we compare the GNN model discovered by Auto-GNAS(AGNAS) with handcrafted models and those designed by other search approaches. Test accuracy on the node classification task is summarized in Table 5, where the best performance is highlighted in bold.

The test accuracy of Auto-GNAS is averaged via randomly initializing the optimal GNN architecture 20 times. Those handcrafted architectures and NAS methods are reported directly from their works. The NAS methods can generally achieve better results than handcrafted methods, demonstrating that it is a practical approach to design good GNN architecture for a given graph data. Significantly, the Auto-GNAS get competitive performance on all three benchmark datasets. The handcrafted models need much tuning of GNN architecture by manual manner for different graph datasets, which is hard to obtain an optimal architecture. On the contrary, the NAS methods can automatically sample the GNN architecture with the progress that improves performance for a given dataset, gradually optimizing the GNN architecture with little human intervention.

4.3 Efficiency Comparison

To further investigate the search efficiency of the Auto-GNAS, we compare the progression of top-10 averaged validation set accuracy of GraphNAS (GNAS) [8], random search (RS) [26], genetic algorithm search (GAS), [27] and Auto-GNAS (AGNAS).

2,000 GNN architectures are explored in the same search space for each search approach. After sensitive analysis of search parameters, we set mutation intensity m as 1, 1, 2, 2

2. https://github.com/rusty1s/pytorch_geometric
3. https://github.com/ray-project/ray

4. https://github.com/GraphNAS/GraphNAS

TABLE 5
Test Accuracy Comparison of Node Classification

Category	model	Cora	Citeseer	Pubmed
Handcrafted Architectures	Chebyshev	81.2%	68.8%	74.4%
	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%
	AGNAS	**83.9 ± 0.4%**	**73.9 ± 0.3%**	**80.6 ± 0.3%**

for each genetic searcher, respectively, in the efficiency experiment, and the other parameters remain unchanged as Section 4.2 for AGNAS. The search configuration of GraphNAS is the same as the paper [8]. For search configuration of genetic algorithm, randomly generating an initial population, the selection strategy is wheel selection, the crossover manner is a one-point crossover with crossover rate one, and the mutation manner is a one-point random mutation with mutation rate 0.02. The random search only has an exploration rate parameter. As shown in Fig. 7, AGNAS is more efficient in sampling the well-performed GNN architectures during the search process than other search methods. AGNAS can obtain better average validation set accuracy in the same exploration of GNN architecture. The reason is that information entropy is an effective soft constraint for search direction. It can make each searcher focus on the region near the GNN architectures with good performance and simultaneously reserve the probability of exploring other areas in the vast search space. Due to the parallel evaluation, AGNAS requires less search time than other methods with serial evaluation manner to explore the same number of GNN architectures.

4.4 Scalability Comparison

To prove the scalability of AGNAS, we use different GPU resources to evaluate the average speedup ratio. As it is easy to expand different search algorithms based on AGNAS for parallel estimation, we test the speedup ratio of GNAS [8], RS[26], and GAS [27] at the same time.

Each search method explores 2,000 GNN architectures on Cora and Citeseer with different GPU resources and then computing average speedup radio based on multiple tests. Each search algorithm configuration is the same as the efficiency experiment. As shown in Fig. 8, with GPU resources increasing, the speedup ratios of each search algorithm have been continuously improved. The AGNAS acquires speed of 2.00x, 2.91x, 3.92x on Cora and 1.68x, 2.77x, and 3.58x on Citeseer at 2, 3, and 4 GPUs. The GNAS [8] acquires speed of 1.89x, 2.97x, 3.78x on Cora and 1.99x, 2.90x, 3.55x on Citeseer at 2, 3, 4 GPUs, respectively. The RS [26] acquires speed of 1.86x, 2.75x, 3.31x on Cora and 1.89x, 2.63x, 3.28x on Citeseer at 2, 3, 4 GPUs, respectively. The GAS [27] acquires speed of 1.88x, 2.56x, 3.22x on Cora and 1.88x, 2.65x, 3.13x on Citeseer at 2, 3, 4 GPUs, respectively. As the degree of parallelism increases, the cost of communication will also increase, resulting in a loss of speedup ratio. Since the search direction of AGNAS and GNAS is better, the speedup rate is more stable than genetic algorithm search and random search with different GPU resources. The result shows that AGNAS with different search methods almost achieves linear speedup.

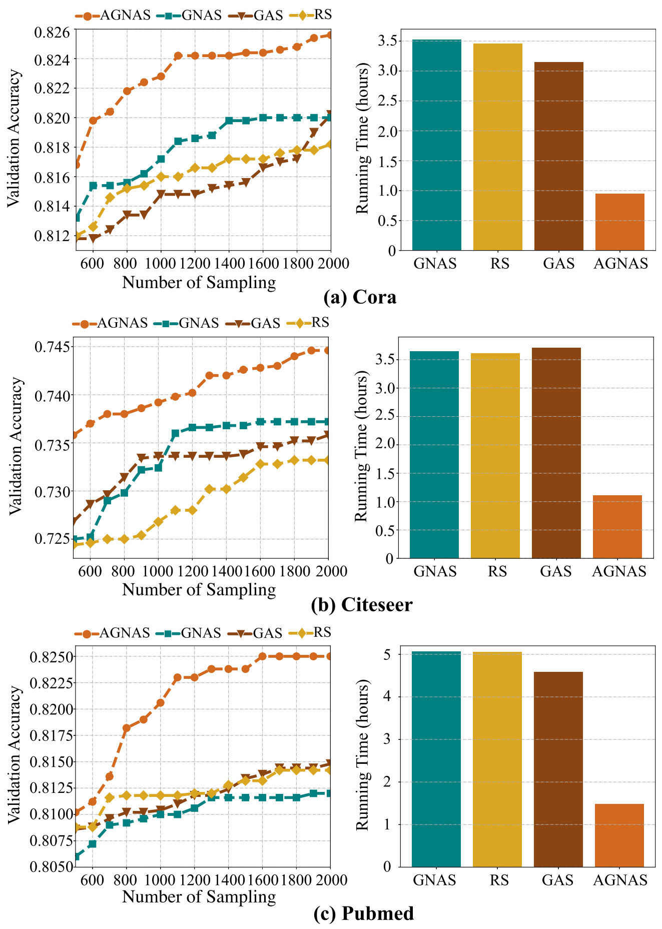

Fig. 7. Efficiency comparison for different search methods on Cora, Citeceer, and Pubmed including Auto-GNAS (AGNAS), GraphNAS (GNAS), genetic algorithm search (GAS) and random search (RS).

4.5 Ablation Test for Auto-GNAS

To explore how information entropy (IE) constraint, multiple genetic searchers, and multiple different mutation intensities improve the search efficiency of the AGNAS, we

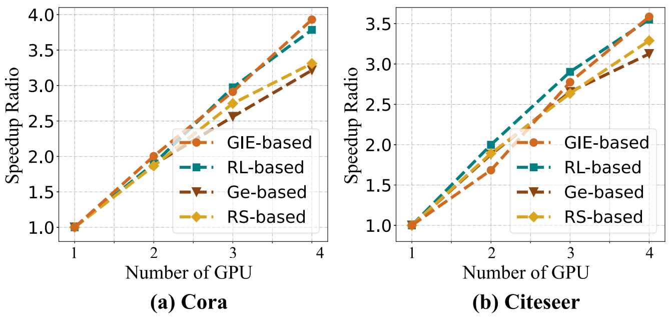

Fig. 8. The speedup ratio with different GPU resources and search methods on the Cora and Citeseer datasets. GIE-based, RL-based, Ge-based, and RS-based mean the Genetic based search algorithm with Information Entropy, Reinforcement Learning based search algorithm, Genetic based search algorithm without Information Entropy, and Random Search algorithm, respectively.

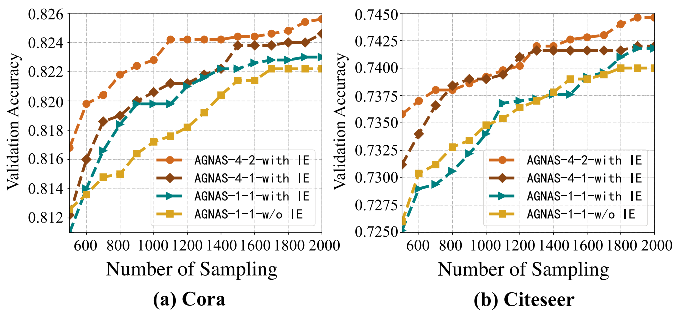

Fig. 9. The search efficiency comparison based on different variants of Auto-GNAS on Cora and Citeseer.

conduct the ablation study on the following variants on the Cora and Citeseer dataset. The AGNAS-1-1-w/o IE represents one genetic searcher without information entropy constraint, and the mutation intensity is 1. The AGNAS-1-1-with IE denotes one genetic searcher with information entropy constraint, and the mutation intensity is 1. The AGNAS-4-1-with IE stands for four genetic searchers with information entropy constraint, and the mutation intensity is 1 for all searchers. The AGNAS-4-2-with IE indicates four genetic searchers with information entropy constraint, and the mutation intensity is 1, 1, 2, 2 for each searcher, respectively. As shown in Fig. 9, the variant of AGNAS-4-2-with IE achieves the optimal search efficiency compared to other variants on the two datasets. The variants with the information entropy constraint can get better search efficiency, and it proves that the information entropy limit search direction is effective for improving search efficiency. The variants with multiple genetic searchers achieve better performance of search efficiency. Because multiple genetic searchers can enhance the population diversity, which is beneficial for genetic algorithms to avoid local optimal solutions. It is helpful for AGNAS to obtain better search efficiency. Comparing the search efficiency curves of AGNAS-4-2-with IE and AGNAS-4-1-with IE demonstrates that different mutation intensities play an essential role in enhancing the overall exploration ability of four genetic searchers for improving search efficiency.

4.6 Multitask Performance and Efficiency Analysis

To prove the universality of AGNAS, we conducted multiple graph task experiments to compare the performance and efficiency of AGNAS with RS, GAS, and GNAS. The task of graph classification is to predict which class an unseen graph belongs to, and the link prediction task is to predict whether an edge exists between a pair of nodes.

TABLE 6
Test Performance Comparison of Graph Classification and Link Prediction

Task	Graph Classification		Link Prediction	
Metrics	Accuracy		AUC	
Dataset	AIDS	Proteins	Cora	Citeseer
RS	$71.9 \pm 6.9\%$	$68.2 \pm 1.0\%$	$99.0 \pm 0.1\%$	$98.7 \pm 0.1\%$
GAS	$85.2 \pm 7.3\%$	$71.9 \pm 1.5\%$	$98.9 \pm 0.2\%$	$99.1 \pm 0.1\%$
GNAS	$91.0 \pm 4.2\%$	$74.4 \pm 3.3\%$	$99.0 \pm 0.1\%$	$99.3 \pm 0.1\%$
AGNAS	$\mathbf{92.0 \pm 2.0\%}$	$\mathbf{76.2 \pm 0.6\%}$	$\mathbf{99.2 \pm 0.1\%}$	$\mathbf{99.6 \pm 0.1\%}$

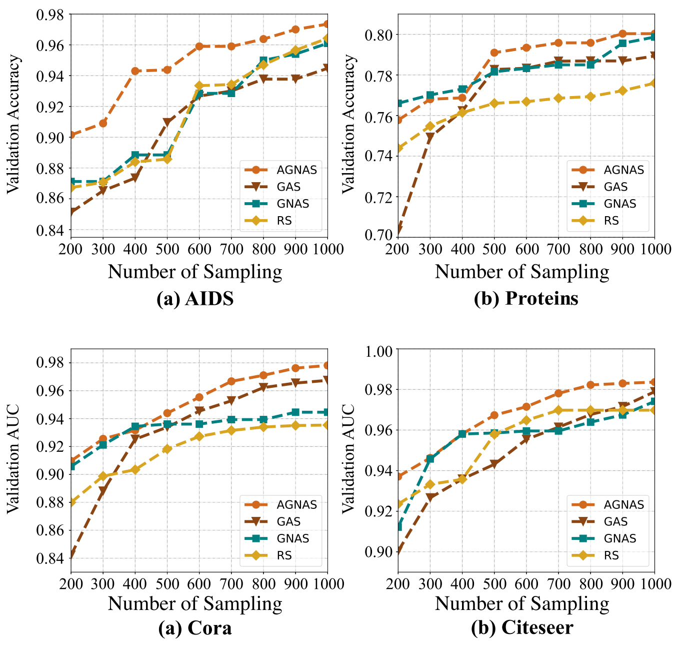

Fig. 10. Efficiency comparison for different search methods on AIDS, Proteins, Cora, and Citeseer.

In the graph classification task, the datasets are divided randomly, and 60% for training, 20% for validating, and 20% for testing. For the link prediction task, a balanced set of positive and negative(false) edges to the original graph, with 20% edges used for validation and 20% edges used for testing. Each search algorithm samples 1,000 GNN architectures from the same search space and uses the same search parameters as the above node classification task in Section 4.2. We set 25 training epochs for each sampled GNN architecture for the graph classification task to generate feedback in the search process. And in the testing step, we train the optimal GNN architecture for 100 epochs and get the average performance based on ten experiments. In the link prediction task, we use five training epochs in the search process for each sampled GNN architecture and 50 training epochs for the testing step. The performance results are shown in Table 6, and the efficiency results are summarized in Fig. 10. Experimental results show that AGNAS can achieve competitive performance for different graph tasks compared with other search algorithms while the search efficiency is better than other search algorithms.

4.7 Sensitivity Analysis of Search Parameters

To study the influence of search parameters on the performance of AGNAS to obtain the optimal GNN architecture, we design a search parameter sensitivity experiment based on node classification with the Cora dataset. All parameters for each parameter sensitivity experiment, except for the search parameters, are consistent with Section 4.2. Since the search parameters n, k, t are related, search parameters N, m are related, and the search parameter t has little effect on the performance experiment, we discuss the search parameter n, k together and the search parameter N, m together. As shown in Fig. 11, M stands for random initialization population size in Fig. 11a. n, k is the scale of initial sharing population and parents respectively in Fig. 11b. N is the number of the genetic searcher in Fig. 11c. m represents mutation intensity for each genetic searcher in Figs. 11c and 11d. In

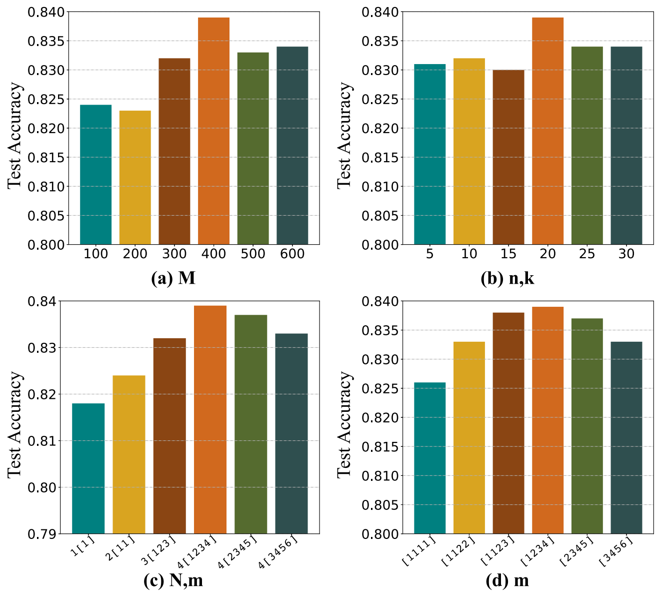

Fig. 11. Accuracy performance comparison with different search parameters. M stands for random initialization population size, and n, k is the scale of the initial sharing population and parents, respectively. N is the number of the genetic searcher. m represents mutation intensity for each genetic searcher.

the Fig. 11c, 1[1] means we set one genetic searcher with mutation intensity 1 for AGNAS, 3[1 2 3] represents AGNAS uses three genetic searchers with different mutation intensity 1, 2, 3. In the Fig. 11d, AGNAS uses four genetic searchers with same or different mutation intensity for each genetic searcher, such as 1, 1, 1 ,1 and 1, 2, 3, 4. To better analyze the impact of different search parameters on performance, we use the search parameters that achieve the best performance for node classification as the benchmark.

As shown in Fig. 11a, AGNAS can obtain better performance by increasing M. This is because increasing the size of M also increases the probability of AGNAS exploring different regions of the search space, thus increasing the possibility of obtaining better performance. However, with the large-scale increase of M, the probability of AGNAS falling into the local optimal solution increases, which reduces the possibility of continuous performance improvement. As shown in Fig. 11b, the effect of the initial sharing population-scale n and the parents k is not evident to the performance of the AGNAS. The reason is that AGNAS has multiple genetic searchers with different mutation intensities to sample GNN architecture simultaneously based on the sharing mechanism, which enhances the diversity of the population and helps improve the search performance of AGNAS. At the same time, it can be observed that expanding the size of n and k can improve the search performance of AGNAS. As shown in Figs. 11c and 11d, we can find that too few genetic searchers N and single mutation intensity m for each genetic searcher are not conducive to better performance of AGNAS. The reason is that reducing N and the single m for each genetic searcher will harm the population diversity in the search process, which will make AGNAS fall into the optimal local solution early in the search process and affect the search performance. The excessive mutation intensity m reduces the stability of AGNAS in the search process, which affects the search performance of AGNAS.

TABLE 7
Test Accuracy Performance Comparison With Different Number of Sampling (NS)

NS	RS	GAS	GNAS	AGNAS
500	$89.2 \pm 0.4\%$	$89.4 \pm 0.4\%$	$88.4 \pm 0.3\%$	$89.7 \pm 0.4\%$
1000	$86.9 \pm 0.4\%$	$89.5 \pm 0.4\%$	$87.7 \pm 0.3\%$	$89.5 \pm 0.5\%$
1500	$87.0 \pm 0.5\%$	$89.3 \pm 0.4\%$	$88.6 \pm 0.4\%$	$89.6 \pm 0.5\%$
2000	$89.2 \pm 0.4\%$	$87.3 \pm 0.4\%$	$87.9 \pm 0.6\%$	$89.4 \pm 0.3\%$

4.8 Number of Sampling and Performance Analysis

We use the Cora dataset to perform a node classification experiment and study the performance of different search algorithms at different GNN architecture numbers of sampling. We randomly shuffle the dataset, 60% for training, 20% for validation, and 20% for testing, and use test accuracy as the evaluation metric. The search and GNN training parameters of different search algorithms are consistent with Section 4.2.

As shown in Table 7, GAS and AGNAS have good convergence speed and stability. They can obtain the best performance with less GNN architecture number of sampling, and the best performance is stable as the number of sampling increases. With the increase of GNN architecture number of sampling, the best performance generated by RS and GNAS is unstable. This shows that the convergence of the RS and GNAS is poor. Under different GNN architecture numbers of sampling, Auto-GNAS obtains the best performance compared to other search algorithms, which shows that AGNAS is a reliable and efficient search framework.

5 Related Work

In this section, we introduce the related work about graph neural architecture search comprehensively.

Based on Reinforcement Learning. Reinforcement Learning is an efficient and dynamic learning process in which an agent learns how to achieve a goal based on continuous feedback signals to maximize the expected reward. It has three main components: the agent, the learner and generates the action, the environment, the agent's interactive object, and the reward for each action generated by the agent. GraphNAS [8] use an LSTM model as the agent to sample different GNN architectures, and the estimation of GNN architecture is the environment. The LSTM is trained based on policy gradient to maximize the expected validation set accuracy of the sampled GNN architectures. Another contribution of this work is to propose a typical graph neural network architecture search space. Different from GraphNAS [8], Auto-GNN [9] constructs multiple different simple RNNs as agents to sample corresponding component values for one GNN architecture. In each iteration of agent training, one or more components are randomly selected for sampling with the corresponding agents to generate a new GNN architecture.

Based on Evolution Learning. Inspiring by natural selection, evolution learning is a simple and effective optimization method. Evolutionary learning contains many algorithms, among which genetic algorithm is typical evolutionary learning algorithm. The genetic algorithm mainly includes several processes. First is the initializing of the population that encodes the problem's solution and obtains the encoding settings of the initial solution. And then, the estimation function evaluates the initial solution and outputs the fitness that indicates the quality of the solution. Next, the selection process chooses the evolution object as the parents based on fitness from the population. Subsequently, the crossover and mutation operation creates a new solution encoding as the child. Finally, the estimation function evaluates the child's fitness, and updating operation decides which child will replace the previous individual in the population. Genetic-GNN [10] designs a GNN search space including two subspaces. One is the architecture components subspace, and another is the hyper-parameters subspace. It uses the naive genetic algorithm to determine the optimal GNN architecture and hyper-parameters simultaneously. AutoGraph [28] uses the idea of age evolution [8] to construct the genetic search algorithm, which selects the oldest individual in the population and removes it before adding the new child to the population.

Based on Random Search. Random search is the simplest search method and is used in many search scenarios as a baseline [9]. The basic random search design idea is to randomly sample a new solution in the search space and replace the current solution if the new solution is better than it. You [29] proposes a controlled random search for sampling the GNN architecture, which is a variant of random search. Moreover, their work discusses the importance of designing an effective GNN search space for graph neural architecture search.

6 Conclusion

This paper proposes a search framework named Auto-GNAS, which can speed up the GNN architecture estimation efficiency in parallel. We open source the Auto-GNAS framework. It is easy for users to expand different search algorithms, search space, and downstream graph tasks. To improve search efficiency, Auto-GNAS constructs multiple genetic searchers with different mutation intensities to sample the GNN architectures simultaneously. Each searcher can simultaneously use the feedback information of GNN architecture estimation and information entropy to accelerate the search process for getting better GNN architecture. Experiment results on the different graph tasks show that Auto-GNAS can obtain competitive performance and more efficiency than other search methods. The scalability testing proves that Auto-GNAS can achieve a nearly linear speedup ratio. In addition, the ablation studies show the effectiveness of the design idea for the search algorithm of Auto-GNAS.

References

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[2] D. Song, F. Zhang, M. Lu, S. Yang, and H. Huang, "DTransE: Distributed translating embedding for knowledge graph," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2509–2523, Oct. 2021.

[3] D. Yang, J. Liu, and J. Lai, "EDGES: An efficient distributed graph embedding system on GPU clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1892–1902, Jul. 2021.

[4] T. Geng *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 922–936.

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learn. Representations*, 2018, pp. 1–12.

[7] T. Elsken *et al.*, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[8] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2020, vol. 20, pp. 1403–1409.

[9] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-GNN: Neural architecture search of graph neural networks," 2019, *arXiv:1909.03184*.

[10] M. Shi *et al.*, "Evolutionary architecture search for graph neural networks," 2020, *arXiv:2009.10199*.

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

[12] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, pp. 4780–4789.

[13] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 1946–1956.

[14] E. Cantú-Paz *et al.*, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Syst. Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3837–3845.

[16] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[17] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1416–1424.

[18] B. Xu, J. Huang, L. Hou, H. Shen, J. Gao, and X. Cheng, "Label-consistency based graph neural networks for semi-supervised node classification," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 1897–1900.

[19] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun , "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*.

[20] D. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," 2015, *arXiv:1509.09292*.

[21] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*.

[22] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *Proc. 30th Int. Conf. Mach. Learn.*, 2019, pp. 7134–7143.

[23] M. Hahsler, B. Grün, and K. Hornik, "Introduction to arules–mining association rules and frequent item sets," *ACM SIGKDD Explorations*, vol. 2, no. 4, pp. 1–28, 2007.

[24] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "TUDataset: A collection of benchmark datasets for learning with graphs," 2020, *arXiv:2007.08663*.

[25] P. Moritz *et al.*, "Ray: A distributed framework for emerging AI applications," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implementation*, 2018, pp. 561–577.

[26] M. Nunes and G. L. Pappa, "Neural architecture search in graph neural networks," in *Proc. Braz. Conf. Intell. Syst.*, 2020, pp. 302–317.

[27] J. McCall , "Genetic algorithms for modelling and optimisation," *J. Comput. Appl. Math.*, vol. 184, no. 1, pp. 205–222, 2005.

[28] Y. Li and I. King, "AutoGraph: Automated graph neural network," in *Proc. Int. Conf. Neural Inf. Process.*, 2020, pp. 189–201.

[29] J. You, Z. Ying, and J. Leskovec, "Design space for graph neural networks," *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 17009–17021, 2020.

Jiamin Chen received the BS degree in applied physics from Nanchang University, Nanchang, Jiangxi, China, in 2014, and the MS degree in radio physics from Nanchang University, Nanchang, Jiangxi, China, in 2017. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Central South University, Changsha, Hunan, China. His research interests include automatic machine learning and graph neural networks.

Jianliang Gao received the PhD degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, China. He is currently a professor with the School of Computer Science and Engineering, Central South University, China. He is the general chair of the 2016 IEEE Conference on Big Data. His research interests include machine learning, and graph data mining.

Yibo Chen received the PhD degree in computer computer system structure, Wuhan University, China, in 2012. He is currently a senior engineer in Information and Communication Branch of State Grid Hunan Electric Power Company Limited, Changsha, China. His research interests include Big data processing, Internet of Things, and edge computing.

Babatounde Moctard Oloulade received the BS degree in computer science from the National School of Applied Economics and Management, University of Abomey-Calavi, Cotonou, Benin, in 2012, and the MS degree in computer science and technology from the Wuhan University of Technology, Wuhan, China, in 2020. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include Big Data and graph mining.

Tengfei Lyu received the BS degree from Bohai University, Jinzhou, Liaoning, China, in 2019. He is currently working toward the master's degree in the School of Computer Science and Engineering, Central South University, Changsha, Hunan, China. His research interests include machine learning, and graph neural networks.

Zhao Li received the PhD degree (Hons.) from the Computer Science Department, University of Vermont, Burlington, Vermont. He is currently with the Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies, specializing in e-commerce ranking and recommendation systems. He has published several articles in prestigious conferences and journals, including NIPS, AAAI, IJCAI, SIGIR, and *IEEE Transactions on Knowledge and Data Engineering*.