Comprehensive Architecture Search for Deep Graph Neural Networks

Yukang Dong, Fanxing Pan, Yi Gui, Wenbin Jiang, *Member, IEEE*, Yao Wan, Ran Zheng, and Hai Jin, *Fellow, IEEE*

***Abstract*—In recent years, Neural Architecture Search (NAS) has emerged as a promising approach for automatically discovering superior model architectures for deep Graph Neural Networks (GNNs). Different methods have paid attention to different types of search spaces. However, due to the time-consuming nature of training deep GNNs, existing NAS methods often fail to explore diverse search spaces sufficiently, which constrains their effectiveness. To crack this hard nut, we propose CAS-DGNN, a novel comprehensive architecture search method for deep GNNs. It encompasses four kinds of search spaces that are the composition of aggregate and update operators, different types of aggregate operators, residual connections, and hyper-parameters. To meet the needs of such a complex situation, a phased and hybrid search strategy is proposed to accommodate the diverse characteristics of different search spaces. Specifically, we divide the search process into four phases, utilizing evolutionary algorithms and Bayesian optimization. Meanwhile, we design two distinct search methods for residual connections (All-connected search and Initial Residual search) to streamline the search space, which enhances the scalability of CAS-DGNN. The experimental results show that CAS-DGNN achieves higher accuracy with competitive search costs across ten public datasets compared to existing methods.**

***Index Terms*—Deep graph neural networks (GNNs), neural architecture search (NAS), residual connection, evolutionary algorithm.**

Received 24 May 2024; revised 20 January 2025; accepted 10 March 2025. Date of publication 17 March 2025; date of current version 4 September 2025. This work was supported by the National Key Research and Development Program of China under Grant 2022YFB4501400 and in part by the National Natural Science Foundation of China under Grant 62372199. Recommended for acceptance by Z. Liu. *(Corresponding author: Wenbin Jiang.)*

Yukang Dong, Fanxing Pan, Yi Gui, Yao Wan, Ran Zheng, and Hai Jin are with the National Engineering Research Center for Big Data Technology, Huazhong University of Science and Technology, Wuhan 430074, China, also with the Service Computing Technology and System Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China, also with the Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: yukangdong@hust.edu.cn; plant310@hust.edu.cn; guiyi@hust.edu.cn; wanyao@hust.edu.cn; zhraner@hust.edu.cn; hjin@mail.hust.edu.cn).

Wenbin Jiang is with the National Engineering Research Center for Big Data Technology, Huazhong University of Science and Technology, Wuhan 430074, China, also with the Service Computing Technology and System Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China, also with the Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China, also with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Zhejiang Lab, Hangzhou 311121, China (e-mail: wenbinjiang@hust.edu.cn).

Digital Object Identifier 10.1109/TBDATA.2025.3552336

I. Introduction

GRAPH Neural Networks (GNNs) [1], [2], [3], [4], [5], [6] have emerged as a prominent topic in recent years. With the development of deep learning, deepening the layers of GNNs [7], [8], [9] has also attracted considerable attention. However, the manual design of effective deep GNNs remains highly challenging. Neural Architecture Search (NAS) [10], [11], [12], [13], [14], [15] presents a potential solution. Unfortunately, the majority of existing NAS methods tailored for GNNs [16], [17], [18], [19], [20], [21], [22], [23] are primarily constrained to simple search architectures. This is due to the formidable challenges of designing an effective search space and selecting efficient search strategies for deep GNNs. Moreover, the heightened training overheads inherent to deep GNNs further complicate the practical application of NAS techniques.

In deep GNNs, the impact of operator compositions and aggregation types on model performance should not be ignored. Most existing GNNs are built on the foundation of two types of operations: aggregate and update. The aggregate operator aggregates information from neighboring nodes, whereas the update operator facilitates the updating of node information via neural networks [2]. The composition of these two operators has a significant impact on model performance. DFG-NAS [17] employs an evolutionary algorithm with a unique mutation mechanism. After each round of evolution, it can either add an aggregate or update operator to the model or convert an aggregate operator into an update operator, enabling the exploration of operator compositions, and thereby achieving impressive results. However, by using a fixed type of aggregation operator, residual connection method, and hyper-parameter combination, DFG-NAS missed the opportunity to discover potentially more optimal model architectures. Nowadays, there are various aggregation types available, such as SUM [2], MEAN, MAX [3], Attention [4], Isomorphism [24], etc. Further exploration is necessary to combine different aggregation types under novel operator compositions.

Furthermore, residual connections have become almost indispensable for deep GNNs. Currently, there are many ways to implement residual connections, such as JK-Net [25], DAGNN [7], GCNII [8], AIR [9], etc. The fundamental idea of residual connections is to incorporate the output of the previous layer as part of the input to the subsequent layer. Ideally, the performance of a deep model should not deteriorate beyond that of the corresponding shallow model. However, what kind of

residual connection is the best? Is it necessary to connect every layer? In both DNNs [26], [27] and deep GNNs, addressing these questions theoretically is challenging, and NAS may offer insights.

Additionally, the performance of deep GNNs is highly sensitive to hyper-parameter settings, including hidden size, dropout, learning rate, L2 norm, etc. Therefore, conducting automatic searches for operator composition, aggregation types, residual connections, and hyper-parameters in deep GNNs would be a meaningful and challenging task.

Selecting practical and efficient search strategies is crucial in such a comprehensive and expansive search space. Nowadays, commonly used search strategies include reinforcement learning [10], [28], evolutionary algorithms [29], [30], [31], differentiable methods [11], [32], and Bayesian optimization [33], [34]. The utilization of reinforcement learning (RL) demands substantial computational resources and is not well-suited for searching deep GNNs. For instance, GraphNAS [35] and SNAG [36] employ reinforcement learning-based search strategies, yet they are limited to exploring GNNs with a depth of only three layers. Differentiable methods (DM), rooted in gradient descent techniques, face the challenge of lacking proven convergence, especially in deep GNNs. Meanwhile, evolutionary algorithms (EA) also demand substantial computational resources, as they rely on specialized mutation and crossover operations that must be carefully tailored to the specific search space. Despite the inherent randomness of EA, they demonstrate robustness and consistently deliver satisfactory results, highlighting their effectiveness in the NAS domain. On the other hand, Bayesian optimization (BO) has proven to be highly suitable for hyper-parameters search after years of development. Based on the analysis above, we select EA and BO as the search strategies in this paper. Additionally, by carefully choosing the appropriate number of evolutionary cycles, we managed to control the computational resource consumption effectively.

In this work, we propose CAS-DGNN, a novel comprehensive architecture search method for deep GNNs. We design unique search spaces and strategies to adapt to the search for deep GNN. The search spaces of CAS-DGNN include four elements: (1) the composition of aggregate and update operators, (2) five types of aggregate operators, (3) each residual connection between any two layers, and (4) the combinations of hyper-parameters. A single search strategy becomes impractical in such a comprehensive and expansive search space. Thus, we adopt a phased and hybrid search strategy to accommodate the diverse characteristics of different search spaces. Specifically, we divide the search process into four phases, utilizing evolutionary algorithms and Bayesian optimization to address the four distinct elements within the search space. Meanwhile, we design two distinct search methods for residual connections (All-connected search and Initial Residual search), denoted as CAS-DGNN v1 and v2. The primary issue with the All-connected search (CAS-DGNN v1) lies in the substantial memory usage, as this approach requires storing the computation results of all intermediate layers, which limits the scalability of our method. To alleviate this problem, we introduce the Initial Residual search (CAS-DGNN v2). This method only needs to store the initial node representations, significantly reducing memory pressure and allowing our method to scale to larger datasets. Finally, the experimental results demonstrate that CAS-DGNN outperforms existing methods across ten publicly available datasets with competitive search costs. The maximum improvement in test accuracy reaches 1.2%.

To summarize, the contributions of this work are as follows.

- In this work, we propose CAS-DGNN, a novel comprehensive architecture search method for deep GNNs. Our search space encompasses operator compositions, aggregation types, residual connections, and hyper-parameters.
- To accommodate the diverse characteristics of different search spaces, we adopt a phased and hybrid search strategy, encompassing four search phases and combining evolutionary algorithms and Bayesian optimization.
- To streamline the search space and enhance the scalability of our method, we design two distinct search methods for residual connections (All-connected search and Initial Residual search).
- The experimental results demonstrate that CAS-DGNN achieves the highest test accuracy with competitive search costs across ten publicly available datasets compared to existing methods.

II. Related Works

A. Graph Neural Networks

Most GNNs can be formulated using the neural message passing framework [2], [3], [4] which encompasses two operators: aggregate and update, as shown in (1) and (2). The aggregate operator denotes node v aggregate information from its neighboring nodes $\mathcal{N}_v$ while the update operator updates the information of node v by neural networks.

$$m_v^{(l)} = \text{Aggregate}\left(\left\{h_u^{(l-1)} | u \in \mathcal{N}_v\right\}\right) \tag{1}$$

$$h_v^{(l)} = \text{Update}(m_v^{(l)}) \tag{2}$$

where $h_u^{(l)}$ and $h_v^{(l)}$ represent the features of nodes u and v in layer l, respectively. $\mathcal{N}_v$ denotes the neighbor set of node v.

Meanwhile, there are multiple types of aggregate operators available. For example, GCN [2] introduces the SUM operator, which is considered the most fundamental aggregate operator. In the aggregation process, each node in the graph aggregates its own information along with that of its neighbors directly. GraphSage [3] introduces MEAN, MAX, and LSTM operations based on GCN. Attention mechanism is introduced in GAT [4]. By computing attention scores between nodes, GAT can focus on more critical nodes in the graph. Similarly, GIN [24] employs a coefficient ϵ to distinguish the node itself and its neighboring. However, these efforts maintain a tight coupling between aggregate and update operators, which means each layer adopts one aggregate operator and one update operator. In contrast, SGC [5] decouples these two operators, arguing that the performance of GNNs primarily relies on the aggregate operators. Therefore, SGC removes the update operators in mid-layers and stacks multiple aggregate operators, sacrificing a small amount of

accuracy but significantly speeding up model training. In SGC, 'hops' is more suitable for describing the depth of the model. This design framework has inspired many shallow and deep GNNs.

As for deep GNNs, we focus on residual connections. To tackle the over-smoothing issue in deep GNNs [37], [38], [39], [40], [41], [42], researchers introduce residual connections to enhance GNN performance. For instance, JK-Net [25], a pioneering study, introduces residual connections in GNNs by aggregating results from previous layers in the final layer to obtain the ultimate representation. Similarly, GCNII [8] introduces the concepts of *Initial Residual* and *Identity Mapping*, applying them separately to aggregate and update operator. DAGNN [7] introduces a dense connection scheme and adaptive coefficients for each aggregate operator, deepening GNNs to hundreds of layers. AIR [9] builds on the advantages of these models and introduces *Adaptive Initial Residual*, which can be integrated with various GNN architectures, including GCN, SGC, and APPNP [6]. These works provide valuable insights and inspiration to us, significantly enhancing our ability to design and refine the search space.

B. Neural Architecture Search

Neural Architecture Search (NAS) methods significantly broaden the scope of automated GNNs development, enabling the discovery of exceptional and robust GNNs. Progress in NAS methods typically centers on two critical research dimensions: the search space and the search strategy. Many studies [16], [17], [20], [35], [43] categorize the search space into different levels: Micro, Macro, and HP. The Micro search space in GNN encompasses the design of graph convolution layers, specifically emphasizing the intricacies of the aggregate operator. In contrast, the Macro search space focuses on the interaction between different layers. Furthermore, HP denotes the combination of various hyper-parameters, which is independent of the Micro and Macro search space. According to this classification, in our search space, operator composition, and residual connections belong to the Macro level, while aggregation types belong to the Micro level. Without a doubt, the Hyper-parameters search belongs to the HP level.

In existing research, Auto-GNN [16] focuses on Micro search, while GraphNAS [44], GraphGym [20], and DFG-NAS [17] concentrate on Macro. Additionally, SANE [18] and F ^{2} GNN [22] include both Micro and Macro search space. Genetic-GNN [19] explores the combination of Micro and HP. And the search space of PaSca [43] encompasses Micro, Macro, and HP, simultaneously. As for search strategy, common methods include reinforcement learning [10], [28], evolutionary algorithms [31], differentiable methods [11], [32], and Bayesian optimization [33], [34]. Meanwhile, there are still many graph NAS methods being developed [45], [46], [47], [48], [49], each with its unique approach to automating the search for optimal GNNs.

In all the aforementioned methods, DFG-NAS [17] holds significant importance for us. They employed a genetic algorithm to search for operator compositions in GNNs. Unfortunately, the search space of DFG-NAS is too limited, as it does not consider the impact of aggregation types, residual connections, and hyper-parameters on model performance. Additionally, their use of a simple genetic algorithm leads to high computational costs during the search process. Therefore, regarding operation composition, there remains considerable scope for further exploration in both the search space and search strategy.

C. Comparison With Existing NAS Methods in GNNs

In this part, we provide a detailed comparison between CAS-DGNN and other state-of-the-art NAS methods in GNNs, including Auto-GNN [16], SANE [18], PaSca [43], and DFG-NAS [17]. As shown in Table I, it is evident that CAS-DGNN stands out due to its unique search space, search strategy, and search depth. Firstly, in terms of the search space, CAS-DGNN simultaneously explores operator composition, aggregation types, residual connections, and hyper-parameters, which is a novel approach not seen in previous methods. Secondly, our search strategy combines evolutionary algorithms with Bayesian optimization. Our innovation lies primarily in optimizing evolutionary algorithms. Specifically, we propose unique crossover and mutation operations during the search process for aggregation types and residual connections. Meanwhile, we introduce an innovative approach by leveraging the Residual Matrix to design different residual connection methods. Finally, CAS-DGNN explores the deepest architectures among the compared methods, with a maximum depth of up to 64 layers. This depth allows for a more comprehensive exploration of the architectural space, potentially leading to the discovery of more effective model architectures. However, it is worth noting that the search depth up to 64 does not necessarily imply that the models in the search results must reach 64 layers. For instance, a maximum search space of 30 layers was built in PaSca [43], but the deepest model found in the search results was only 13 layers. A similar phenomenon also occurred in our search results.

III. Analysis and Motivation

In this section, we analyze the performance fluctuations of deep GNNs. Taking the experimental results on the PubMed dataset as an example, we present four plots in Fig. 1, illustrating the impact of operator composition, aggregation types, residual connections, and hyper-parameters on the performance of deep GNNs. Meanwhile, we compared the differences between the DM and EA methods in operator composition search, further justifying our choice of the EA method. Detailed analysis follows.

A. Operator Composition Influence

In Fig. 1(a), we present the impact of operator composition on the performance of deep GNNs. There are three operator composition schemes: (1) [A, U, A, U,...], (2) [A, A, A, A,...], and (3) [A, A, U, U,...], where A represents the aggregate operator and U represents the update operator. It is worth noting that the preprocessing and classifier layers are added at the

TABLE I
A DETAILED COMPARISON BETWEEN CAS-DGNN AND EXISTING GRAPH NAS METHODS (RL: REINFORCEMENT LEARNING, EA: EVOLUTIONARY ALGORITHMS, DM: DIFFERENTIABLE METHOD, BO: BAYESIAN OPTIMIZATION.)

NAS Methods	Search Space				Search Strategy	Search Depth
	Operator Composition	Aggregation Types	Residual Connections	Hyper-parameters		
SANE [18]	✘	✔	✔	✔	DM	3
Auto-GNN [16]	✘	✔	✔	✘	RL	3 and 16
PaSca [43]	✘	✔	✔	✔	BO	Up to 30
DFG-NAS [17]	✔	✘	✘	✘	EA	Up to 45
CAS-DGNN (OURS)	✔	✔	✔	✔	EA + BO	Up to 64

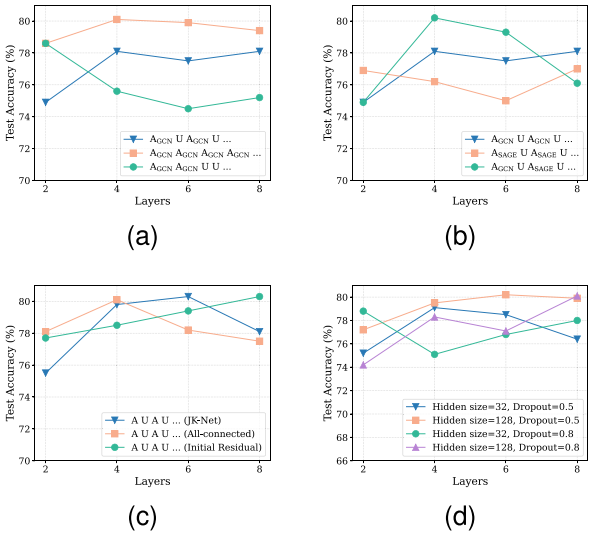

Fig. 1. Severe performance fluctuations of deep GNNs on the PubMed dataset. (a) Operator compositions influence. (b) Aggregation types influence. (c) Residual connections influence. (d) Hyper-parameters influence. (By default, the aggregate operator is set to GCN, the residual connection is set to JK-Net, and the hyperparameters are configured as follows: [hidden size: 32, dropout: 0.5, learning rate: 0.04, L2 norm: 2e-4, training epochs: 200].).

beginning and end of the model, respectively. In operator composition scheme (1), the coupling mode of aggregate and update operators is maintained, similar to GCN, which is the most basic operator composition scheme. In operator composition scheme (2), consecutive aggregate operators is allowed, while update operators only existed in the preprocessing and classifier layers. Finally, in the operator composition scheme (3), we allow consecutive aggregation and update operations simultaneously.

As shown in Fig. 1(a), we observe from the overall trend of the curves that even slight changes in the number or sequence of aggregate and update operators can result in significant fluctuations in model test accuracy. Upon closer inspection, when the operator composition is [A, U, A, U,...], the test accuracy slightly improves with the increase in model depth. In contrast, compared to the other two operator composition schemes, when the operator composition is [A, A, A, A,...], the model's test accuracy consistently remains high. Conversely, when the operator composition is [A, A, U, U,...], the test accuracy sharply decreases with the increase in model depth. We speculate this phenomenon occurs due to the addition of neural network layers leading to model degradation [9], even with the implementation of residual connection strategies failing to prevent such degradation. This suggests that continuous update operators in deep GNNs may compromise model performance. Furthermore, in terms of the contribution to model test accuracy, the aggregate operators often outweigh the update operators. These observations are consistent with those reported in [9], [17]. The analysis above motivates us to assign a higher choice probability to aggregate operators during the search process, which will be detailed in Section IV-B.

B. Aggregation Types Influence

In Fig. 1(b), while keeping the residual connections and hyper-parameters unchanged, we base our analysis on the operator composition [A, U, A, U,...]. We then vary the aggregation types between GCN and SAGE-MEAN. When all aggregation types are SAGE-MEAN, denoted as [A_{SAGE}, U, A_{SAGE}, U,...], the two-layer model exhibits relatively high test accuracy. However, as the model deepens, there is no significant improvement in test accuracy, and it lags behind the model with all aggregation types as GCN. On the other hand, when alternating between GCN and SAGE-MEAN aggregation types, denoted as [A_{GCN}, U, A_{SAGE}, U,...], the four-layer model shows the highest test accuracy. However, as the model depth reaches eight layers, there is a noticeable decrease in test accuracy. This indicates that different combinations of aggregation types may enhance model performance, but it requires consideration of model depth and operator composition schemes.

C. Residual Connections Influence

In this experiment, we keep the operator composition as [A, U, A, U,...], maintaining the default aggregation types as GCN and unchanged hyper-parameters. We investigate the impact of various residual connection methods on model performance, including JK-Net, All-connected, and Initial Residual. Among these three methods, the All-connected approach is the most intricate, establishing residual connections between any two layers. Conversely, JK-Net links only the last layer to all preceding layers, whereas Initial Residual does the opposite, connecting solely the first layer to all subsequent layers. This comparison is illustrated in Fig. 4. As depicted in Fig. 1(c), significant performance differences are observed among different residual connection modes at varying depths.

However, our key observation is that diminishing the count of residual connections does not invariably result in performance decline; in certain scenarios, satisfactory outcomes can still be attained. For instance, at a depth of 8 in Initial Residual method, the test accuracy peaks at 80.3%, marking the optimal performance among all scenarios depicted in the figure.

TABLE II
OPERATOR COMPOSITION SEARCH USING DM AND EA

Datasets	DM (500 epochs)		EA (500 cycles)	
	Acc (%)	Time (h)	Acc (%)	Time (h)
Cora	82.1±0.3	**0.25**	**85.2±0.2**	3.35
CiteSeer	71.2±0.4	**0.18**	**74.1±0.4**	2.22
PubMed	77.9±0.3	**0.68**	**81.1±0.3**	6.40

This encourages us to contemplate individually searching for each residual connection. Furthermore, determining a suitable search strategy within such an expansive search space presents a challenge.

D. Hyper-Parameters Influence

In Fig. 1(d), we illustrate the impact of hyper-parameters on the performance of deep GNNs. We maintain a fixed operator composition of [A, U, A, U,...], with aggregation types defaulted to GCN. For the residual connection, we chose the Initial Residual, as it demonstrated greater robustness in previous experiments. For clarity, we vary the Hidden size and Dropout values while keeping other parameters consistent with previous experiments. Looking at the overall trend of the lines in Fig. 1(d), it is evident that hyper-parameter tuning has a stochastic effect on deep GNNs, making it challenging to discern a clear pattern. As the model depth increases, no single hyperparameter combination demonstrates a clear advantage, suggesting that manually setting dropout, hidden size, and model depth is challenging. Considering the influence of additional hyperparameters further complicates model architecture design. Therefore, utilizing NAS methods to search for optimal hyperparameter combinations may provide an effective solution.

E. DM vs. EA

Since both EA and RL search algorithms require substantial computational resources, and DFG-NAS [17] has already demonstrated the effectiveness of the EA algorithm in operator composition search. Thus, we opt not to consider the RL algorithm. However, the DM search algorithm has shown unique advantages in search efficiency. This raises the question: is DM suitable for operator composition search in deep GNNs? In this section, we compare the performance of the DM and EA search algorithms. Our DM and EA algorithms are based on SANE [18] and DFG-NAS [17], with residual connections and hyper-parameters set to match those in DFG-NAS.

As shown in Table II, the DM algorithm demonstrates remarkable search efficiency, with a total search time of approximately one-tenth that of the EA algorithm. However, the accuracy of DM's search results significantly lags behind that of EA. We observed that DM's results fluctuate in each epoch as it continuously updates the model architecture using backpropagation. In contrast, the EA algorithm updates the model architecture based on its prediction accuracy following the training process, which typically spans 200 epochs. This discrepancy accounts for the substantial difference in search efficiency between the two methods. Furthermore, the convergence of the DM algorithm has not been established, raising the question of whether this approach—updating the model architecture in each epoch—can effectively identify the optimal operator combination for deep GNNs. While this paper does not seek to address this question theoretically, our quantitative experimental results suggest that the DM algorithm may not be suitable.

Furthermore, both BO and RL have their respective limitations as search algorithms. BO is a well-established method for hyperparameter tuning, which is why we chose it for this task. In contrast, applying RL presents challenges, particularly when it comes to modifying the depth of the model during the search process. For example, both Auto-GNN and GraphNAS use RL to perform automated searches within a fixed three-layer GNN model. On the other hand, the crossover and mutation operations in EA allow for more flexible modifications to the model's depth, making it well-suited for the operator composition search. Moreover, our detailed exploration of the crossover and mutation operations throughout the research process enhanced our understanding of the EA algorithm, which motivated us to continue using it in the search for aggregation types and residual connections.

IV. CAS-DGNN DESIGN

In this section, the design of our proposed method, CAS-DGNN, is introduced. It starts with an overview of CAS-DGNN, followed by an explanation of the search space design. Subsequently, a detailed introduction to each phase of the search strategy is provided.

A. Overview

Fig. 2 illustrates the overall framework of our method, CAS-DGNN. We divide the entire search process into four phases. We employ the evolutionary algorithm in the first three phases to search for operator compositions, aggregation types, and residual connections. Finally, in phase 4, Bayesian optimization is employed to explore the combinations of optimal hyper-parameters. After four phases of searching, we obtain the optimal architectures for deep GNNs.

B. Search Space Design

Given that training deep GNNs often requires a significant amount of time, the search space design becomes even more critical. Therefore, based on the analysis in Section III, we include four elements that have a significant impact on deep GNNs: operator composition, aggregation types, residual connections, and hyper-parameters. To our knowledge, CAS-DGNN is the first to search for all these four elements compared to existing NAS methods, as detailed below:

a) Operator composition: We decouple the aggregate and update operators in GNNs, treating each operator as an independent layer. The consecutive appearance of aggregate or update operators is allowed. An operator list represents the order of operator execution, such as [Aggregate, Aggregate, Update, Aggregate], as shown in Fig. 3. Please note that we add a preprocessing layer and a classifier layer at the beginning and the end of the model, respectively. The operator list omits the preprocessing

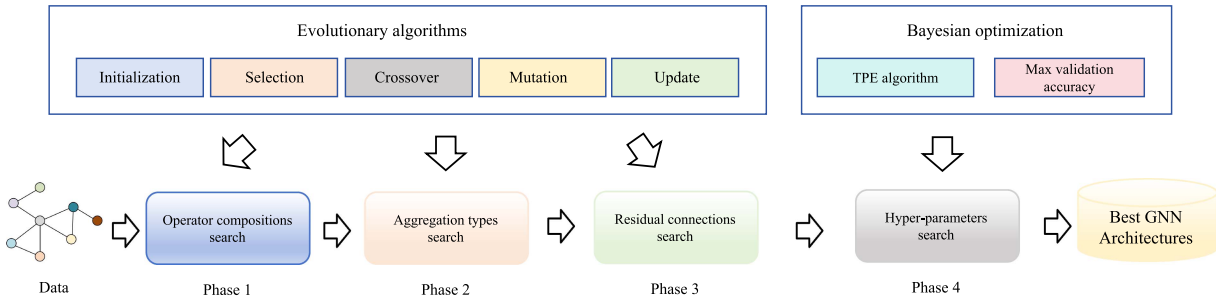

Fig. 2. The overall framework of the proposed CAS-DGNN.

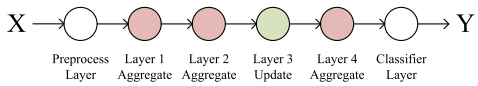

Fig. 3. An example GNN built by the operator list [Aggregate, Aggregate, Update, Aggregate].).

TABLE III
AGGREGATION TYPES EXPLANATIONS (D(V) AND D(U) DENOTE DEGREES OF NODE V AND U.)

Aggregation Types	Explanations
GCN	$\sum_{u \in \mathcal{N}_v} (D_v \cdot D_u)^{1/2} \cdot h_u^{l-1}$
SAGE-MEAN	Apply mean operation to $h_u \vert u \in \mathcal{N}_v$
SAGE-MAX	Apply max operation to $h_u \vert u \in \mathcal{N}_v$
GAT	Attention score: $\sigma(a[W_u h_u \vert\vert W_v h_v])$
GIN [24]	$(1+\epsilon) \cdot h_v^{l-1} + \sum_{u \in \mathcal{N}_v} h_u^{l-1}$

and classifier layers to simplify the expression. The role of the preprocessing layer is to transform the dimensions of the input matrix, ensuring consistency with the output results of intermediate layers. This prepares for the subsequent residual connections. Meanwhile, the classifier layer alters feature dimensions, thereby facilitating the ultimate classification process. To constrain the size of the search space, we set the maximum depth of the model to 64 layers. This implies that the maximum length of the operator list is 64.

b) Aggregation Types: Aggregation types, also called node aggregators, describe how nodes aggregate information from their neighboring nodes. In our search space, we include five classic aggregation types, namely GCN, SAGE-MEAN, SAGE-MAX, GAT, and GIN, outlined in Table III. The aggregation type of GCN aggregates neighboring node information $h_u|u \in \mathcal{N}_v$ directly with the information of node h_v. Building upon this, GraphSAGE introduces operations like MEAN and MAX, while GAT integrates an attention mechanism. GAT biases the aggregation towards more salient nodes by computing attention scores between nodes. Conversely, GIN assigns different weights to self-nodes and neighboring nodes using ϵ, which can be manually set or calculated through training vectors. Since numerous popular aggregation types are often variations of these five, we opt to include only these fundamental types in our exploration. This decision aims to prevent excessive expansion of the search space.

It is important to elucidate the rationale behind separating operator composition and aggregation types, as aggregation types essentially alter only the aggregate operator within the operator composition. This design is motivated by the size of the search space. When considering solely the GCN aggregator in the operator composition, each layer can be either an aggregate or an update operator, resulting in 2^n possible operator compositions for an n-layer network. However, if we incorporate five aggregation types simultaneously, each layer can be one of the five aggregators or an update operator. In this scenario, for an n-layer network, there are 6^n potential operator compositions. As n grows, this number becomes dauntingly large. Therefore, a phased search method becomes highly necessary. In our method, we initially set the aggregate operator to GCN in the operator composition and then explore other aggregators in the search for aggregation types. A more detailed analysis of the search space size will be provided in Section IV-C.

c) Residual Connections: Residual connections, regarded as nearly indispensable for deepening GNNs, have garnered significant attention in recent searches. Our method treats the aggregate and update operators as independent layers. In Fig. 4, taking the operator list as [Aggregate, Aggregate, Update, Aggregate], we depict three different residual connection methods, including JK-Net, All-connected, and Initial Residual. To facilitate crossover and mutation operations in the evolutionary algorithm, we utilize a residual matrix to represent the presence of connections between layers. Specifically, $residual\ matrix[i][j] = 1$ denotes a residual connection between layer i and layer j. Thus, for an n-layer model (excluding the preprocessing and classifier layers), there are a total of $n(n-1)/2$ residual connections. When n is large, this can introduce considerable redundant information into the model. Therefore, we attempt to search for the necessity of each residual connection. However, both JK-Net and All-connected residual connections suffer from a fatal issue: during model training, it is necessary to temporarily store the computation results of all mid-layers, leading to a sharp increase in memory consumption. Consequently, the model cannot be trained on large datasets. In contrast, the Initial Residual method can address this problem. As illustrated in Fig. 4(c), this method only requires storing the results of the preprocessing layer, significantly reducing memory consumption. Additionally, the search space of this method is smaller, making it more suitable for larger datasets.

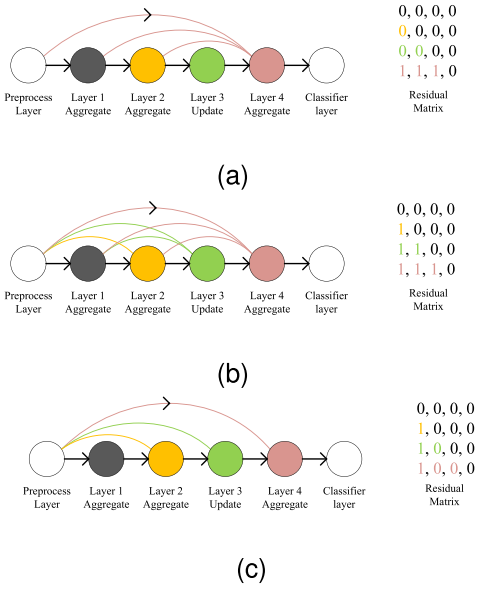

Fig. 4. An example of a 4-layer model with residual connections. (a) JK-Net. (b) All-connected. (c) Initial Residual.

TABLE IV
RESIDUAL CONNECTION METHOD IN CAS-DGNN V1 AND V2

Methods	Phase 1	Phase 2	Phase 3	Dataset size
CAS-DGNN (v1)	JK-Net	JK-Net	All-connected	S, M
CAS-DGNN (v2)	Initial	Initial	Initial	S, M, L

TABLE V
HYPER-PARAMETERS SEARCH SPACE

Hyper-parameters	Range
Hidden size	[16, 32, 64, 128, 256, 512]
Learning rate	[0.1, 0.2, 0.01, 0.02, 0.001, 0.002]
L2 Norm	[1e-4, 2e-4, 3e-4, 4e-4, 5e-4]
Dropout	[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Training epochs	[200, 500, 1000]

In summary, considering the various dataset sizes, we design two search spaces to explore residual connections. As shown in Table IV, for small and medium datasets, we designate the residual connection methods as JK-Net in the first two phases. In phase 3, we search for the necessity of each residual connection in the All-connected method. This method is denoted as CAS-DGNN (v1). Furthermore, for larger datasets, we explore the necessity of each residual connection in the Initial Residual, denoted as CAS-DGNN (v2). Additionally, CAS-DGNN (v2) is capable of conducting searches on small and medium datasets. We will compare the search results of these two methods in the experimental section.

d) Hyper-parameters: Hyper-parameters pose a classic challenge not only in deep GNNs but also in DNNs, where the performance of the deep model is highly sensitive to hyper-parameters setting. We consider factors such as hidden size, dropout, learning rate, L2 Norm, and training epochs. For detailed information, refer to Table V.

TABLE VI
PHASED AND HYBRID SEARCH STRATEGY

Phase	Space Space	Level	Search Strategy
1	Operator composition	Macro	Evolutionary
2	Aggregation types	Micro	Evolutionary
3	Residual connections	Macro	Evolutionary
4	Hyper-parameters	HP	Bayesian

C. Search Strategy

1) Complexity Analysis: First, we analyze the search space size for the non-phased search method. We define O1, O2, O3, and O4 as the search space sizes for operator composition, aggregation types, residual connection, and hyper-parameters, respectively. During a non-phased search, the search for operator composition and aggregation types can be considered collectively. Each layer has T possible operators, resulting in T^n possible configurations for an n-layer model. Hence, within the layer range $[s, n]$, $O1 + O2 = T^s + T^{s+1} + \ldots + T^n = 6^4 + 6^5 + \cdots + 6^{64} \approx 7.2 \times 10^{23}$. For residual connections under the All-connected method, there are a total of $n(n\text{-}1)/2$ connections for an n-layer model. Since each connection can either exist or not, $O3 = \sum_{i=s}^{n} 2^{i(i-1)/2} \approx 1.0 \times 10^{627}$. Calculating the hyper-parameter search space is straightforward, as we multiply the number of options for each hyper-parameter together. Hence, $O4 = 6 \times 6 \times 5 \times 10 \times 3 = 5400$. Therefore, the total search space size using the non-phased search method is $(O1 + O2) \times O3 \times O4 \approx 3.9 \times 10^{695}$, which is a daunting number.

However, when we employ the phased search strategy as shown in Table VI, the size of the search space significantly decreases. In the first phase, we fix the aggregation operator in the operator composition as GCN. This means that each layer of the operator had only two choices: the aggregation operator of GCN or the update operator. Therefore, $O1 = 2^4 + 2^5 + \cdots + 2^{64} \approx 1.8 \times 10^{19}$. In the second phase, we obtain a fixed model depth. Although the crossover operation in this phase of the evolutionary algorithm might change the model depth, for a rough calculation of the search space size, we assume the obtained model depth n=32. Meanwhile, we extend the aggregate operator of GCN to five more types. Thus, $O2 = 5^{32} \approx 8.9 \times 10^{24}$. As for residual connection search, we design two search spaces: $O3_{v1} = 2^{n(n-1)/2} \approx 3.1 \times 10^{149}$ and $O3_{v2} = 2^{32} \approx 4.3 \times 10^{9}$. O4 remains unchanged. Since we adopt phased search, when calculating the total search space, we only need to sum the search spaces of each phase. Thus, in the case of phased search, the search space size of CAS-DGNN (v1) is $O1 + O2 + O3_{v1} + O4 \approx 10^{149}$, while the search space size of CAS-DGNN (v2) is $O1 + O2 + O3_{v2} + O4 \approx 10^{24}$. Comparing the search space sizes between non-phased and phased methods, as well as between CAS-DGNN (v1) and CAS-DGNN (v2), it becomes evident that our optimization of the search space is substantial and indispensable. Additionally, we will compare the search results of phased and non-phased search methods in the ablation study to demonstrate the necessity of phased search experimentally.

2) Implementation Details: We provide a detailed explanation of the specific implementation of each phase

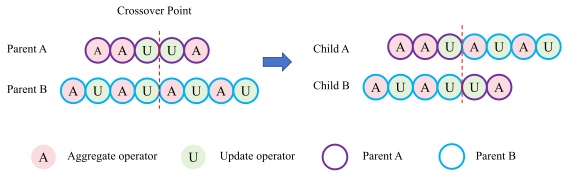

Fig. 5. Simulated binary crossover based on the operator list in Phase 1.

TABLE VII
MUTATION OPERATIONS WITH DIFFERENT PROBABILITIES (A: AGGREGATE OPERATOR, U: UPDATE OPERATOR.)

Mutation	Description	Probability
+ A	Add an aggregate operator.	0.3
+ U	Add an update operator.	0.15
−	Remove an operator.	0.1
A → U	Replace A with U.	0.15
U → A	Replace U with A.	0.3

as below. Our custom evolutionary algorithm consists of five essential steps: Initialization, Selection, Crossover, Mutation, and Update.

a) Phase 1. Operator compositions search: In phase 1, we employ an evolutionary algorithm to search for the optimal operator compositions. The detailed operations of the five steps are as follows:

Initialization in phase 1: We randomly generate k (default 20) different operator compositions with depths ranging from 4 to 6. Based on these operator compositions, we build corresponding GNNs and evaluate them. These randomly initialized GNNs constitute the population set Q.

Selection in phase 1: From the population set Q, m individuals are randomly selected, where $m < k$, and the two top-performing individuals are assigned as parent A and B.

Crossover in phase 1: Two offspring, child C and child D are generated after applying the simulated binary crossover (SBX) [50] strategy to parent A and parent B. The crossover process is illustrated in Fig. 5. The crossover point is fixed at the midpoint of the operator list. When the length of the operator list is odd, we choose the position closer to the last layer.

Mutation in phase 1: We introduce five mutation operations for operator list: +A, +U, −, A → U, and U → A. Based on our previous analysis in Section III, in terms of the contribution to model test accuracy, the aggregate operators often outweigh the update operators, so we assigned lower probabilities to U-related mutation operations, as specified in Table VII. During execution, we randomly select a position in the operator list for the abovementioned mutation. The mutation process in our method represents an enhancement compared to the mutation used in DFG-NAS [17]. We introduce a new "remove" mutation operation and assign different probabilities to each mutation operation. This contributes to the efficiency of the evolutionary algorithm in searching for the optimal operator composition. We will showcase that our improvements to the evolutionary algorithm are highly effective in ablation study.

Update in phase 1: After the crossover and mutation operations, we obtain two entirely new children, denoted as child C and child D. However, only the child with the higher performance (test accuracy) is integrated into the population set Q. Additionally, we implement the population management approach detailed in [51]. Specifically, we control the population size using a variable ϕ, which involves removing the poorest-performing model with a probability of ϕ and the oldest model with a probability of $1 - \phi$.

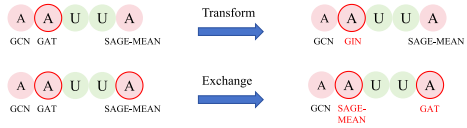

Fig. 6. Mutation operations in phase 2.

b) Phase 2. Aggregation types search: After completing the search in phase 1, we obtain the top-k performing operator compositions. It is important to note that in Phase 1, the default aggregate operator is GCN, while in Phase 2, we explore additional aggregation types. Next, we mainly describe the differences between the evolutionary algorithm used in Phase 2 and that used in Phase 1.

Initialization in phase 2: The initial population set Q in phase 2 comprises two components. One-half is derived from the top-k performing GNNs in phase 1. The other half is generated using a randomization strategy, where the GCN aggregate operators are randomly converted to SAGE-MEAN, SAGE-MAX, GAT, or GIN aggregate operators. This randomization strategy is introduced because, in evolutionary algorithms, high-quality parents do not always guarantee the production of superior offspring through crossover and mutation operations [31]. Thus, we incorporate randomization to enhance the robustness of the algorithm in both phases 2 and 3. We will demonstrate the effectiveness of the randomization strategy in the ablation study.

Selection in phase 2: Same with Selection in phase 1.

Crossover in phase 2: Same with Crossover in phase 1.

Mutation in phase 2: In this phase, the mutation operations primarily focus on aggregation types. We use only two types of mutation operations: *Transform* and *Exchange*, as shown in Fig. 6. The *Transform* mutation operation involves randomly selecting an aggregate operator from the operator list and changing it to one of the five aggregation types with equal probability. During this process, there is a 20% probability that the aggregation type remains unchanged. The *Exchange* mutation operation involves randomly selecting two aggregation types from the operator list and swapping their positions.

Update in phase 2: Same with Update in phase 1.

c) Phase 3. Residual connections search: After the searches in the first two phases, we obtain the optimal operator compositions and aggregation types, which can be uniformly represented using operator lists. In phase 3, the depth of the operator list remains unchanged. Here, our focus shifts to exploring the necessity of each residual connection in either the All-connected or Initial Residual method. The specific evolutionary algorithm is as follows:

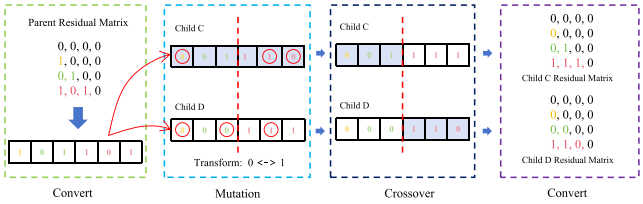

Fig. 7. Mutation and Crossover operations in phase 3.

Initialization in phase 3: The initial population set Q in this phase also comprises two components: one half consists of the top-k results obtained after phase 2, while the other half is generated using the All-connected or Initial Residual connection method with a randomization strategy. Then, the GNNs generated in the latter half are re-evaluated and compared with those from the former half to reconstruct Q.

Selection in phase 3: Randomly select a GNN architecture from top-performing GNNs in population set Q, as a parent.

Mutation and Crossover in phase 3: In phase 3, instead of selecting two parents as in the previous phases, we only choose one model as the parent. This adjustment is necessary because the residual matrix dimensions of different models may vary, rendering crossover infeasible. Additionally, we adjust the execution order for mutation and crossover, as depicted in Fig. 7. We first convert the lower triangular part of the residual matrix to an array and then use the mutation operation to generate two child arrays. We randomly select n positions in the parent array to transform between 0 and 1. After repeating this process twice, two entirely new child arrays are produced. Then, we perform the crossover operation (SBX) on the two child arrays. Finally, the child arrays are converted back to a residual matrix. Performing crossover and mutation operations based on the residual matrix has the advantage that all residual connection methods share a unified representation, facilitating the search and even the replacement of different residual connection methods.

Update in phase 3: Similar to the update process in phase 1, only the child with the higher performance is incorporated into the population set Q.

d) Phase 4. Hyper-parameters search: After years of development, many Bayesian optimization algorithms and their variants [52], [53] have been successfully applied to hyper-parameters tuning. Here, we opt for the Tree-structured Parzen Estimator Approach (TPE) [33], [34] to maximize the accuracy of the validation set. By default, we select the top 5 GNNs obtained from phase 3 for hyper-parameters search, with each GNN undergoing 50 iterations.

V. Experiments and Results

In this section, the experimental setup details are described initially. Subsequently, CAS-DGNN is compared with state-of-the-art manually designed GNNs and graph NAS methods on the node classification task. The architectures searched using our method are showcased, accompanied by a comprehensive analysis. Furthermore, three ablation studies are conducted. The first ablation study compares the phased search methods with non-phased search methods, while the second investigates the impact of each search phase. The final ablation study evaluates different residual connection methods in the first two phases. These studies aim to provide a deeper understanding of the contributions and limitations of our proposed CAS-DGNN.

A. Experimental Settings

1) Baselines: We compare CAS-DGNN with five basic GNNs, three state-of-the-art deep GNNs, and four notable graph NAS methods.

- *Basic GNNs:* GCN [2], GraphSAGE [3], GAT [4], SGC [5], and APPNP [6] are selected for comparison. In GCN, GraphSAGE, and GAT, aggregate and update operators are interleaved and executed alternately. SGC, on the other hand, disentangles GNNs by performing multiple aggregate operators and transforming propagated features using an update operator. Besides, APPNP utilizes personalized PageRank to reinforce GCN, which can be considered an alternative form of residual connection.
- *Deep GNNs:* DAGNN [7], GCNII [8], and AIR [9] are selected. These three GNNs employ distinct residual connection techniques and have achieved impressive results. They have also provided us with valuable insights for designing the search spaces. It should be noted that AIR is a plug-and-play module, and it can be equipped with GCN, SGC, and APPNP.
- *Graph NAS methods:* Auto-GNN [16], GraphNAS [35], SANE [18] and DFG-NAS [17] are selected for comparison. The search spaces and search strategies of these four graph NAS methods have distinctive characteristics, all of which have yielded impressive results. We select these four NAS methods that employ different search strategies. Among them, Auto-GNN and GraphNAS utilize reinforcement learning for searching, while SANE, and DFG-NAS utilize differentiable methods and evolutionary algorithms, respectively. We reproduce all baselines based on open-source code. The source code for PaSca [43] is unavailable, hence it is not included in our baselines.

2) Datasets: We concentrate on the node classification task and perform experiments on ten publicly available graph datasets, encompassing two small datasets (Cora and Citeseer), six medium datasets (PubMed, Computers, Photo, CS, Physics, and DBLP), and two large datasets (Flickr and ogbn-arxiv). The statistics of these datasets are presented in Table VIII. For the datasets of Cora, Citeseer, and PubMed, we follow the public fixed split from [54]. For other datasets, we divide the nodes in all graphs into 60%/20%/20% for training/validation/test.

3) Parameters Settings: In each phase, the top 5 GNNs obtained from the search are passed on to the next phase. For the evolutionary algorithms in the first three phases, we set the total evolutionary cycles to 600, with each phase consisting of 200 cycles. The initial hyper-parameters are set as follows: hidden size=32, dropout=0.5, learning rate=0.01, L2 norm=5e-4, and training epochs=200. In phase 4, each GNN undergoes 50 iterations of hyper-parameters tuning.

TABLE VIII
STATISTICS OF DATASETS

Datasets	Nodes	Edges	Features	Classes	Size
Cora	2,708	10,556	1,433	7	S
CiteSeer	3,327	9,104	3,703	6	S
PubMed	19,717	88,648	500	3	M
Computers	13,752	491,722	767	10	M
Photo	7,650	238,162	745	8	M
CS	18,333	163,788	6,805	15	M
Physics	34,493	495,924	8,415	5	M
DBLP	17,716	105,734	1,639	4	M
Flickr	89,250	899,756	500	7	L
ogbn-arxiv	169,343	1,166,243	128	40	L

The top two results are highlighted in bold black font.

4) Implementation Details: Our experiments are running with Pytorch (version 1.13.1) on an NVIDIA Tesla V100 GPU (Memory: 32 GB, CUDA version: 11.8). We implement CAS-DGNN on top of the building code provided by PyG (version 2.5)[1] and Hyperopt (version 0.2.7).[2] For the optimal results obtained from the search, we retrain the model five times to calculate the standard deviation.

B. Search Results on Node Classification Task

In this section, we delve into a comprehensive analysis of both the performance measured in terms of test accuracy and the architectures searched out by our method.

1) Performance Comparison: In Table IX, the performance (test accuracy) of the optimal GNNs obtained from CAS-DGNN is presented. We can see that CAS-DGNN consistently outperforms all baselines on eight small and medium datasets, demonstrating the effectiveness of CAS-DGNN. Please note that the only difference between CAS-DGNN (v1) and CAS-DGNN (v2) lies in how residual connections are searched. Refer to Section IV-B for more details. Compared to existing methods, CAS-DGNN (v1) achieves an average improvement in accuracy of 0.7% across eight datasets. Notably, CAS-DGNN (v1) achieves the highest test accuracy improvement of 1.2% on PubMed. Although the overall search results of CAS-DGNN (v2) are not as promising as those of CAS-DGNN (v1), they still outperform existing methods on PubMed and CS.

When examining the results of basic GNNs, we find that no single type of GNN exhibits absolute superiority. This is due to most of these models consisting of only two or three layers, and the limited depth of the models restricts their expressive power. When analyzing the results of deep GNNs, we observe that these models generally outperformed basic GNNs significantly. However, similarly to basic GNNs, no single type of deep GNN exhibit absolute superiority. This underscores the importance of searching for optimal operator compositions and residual connection methods. Among NAS methods, the most promising method in terms of overall performance is DFG-NAS. This is attributed to its ability to search for architectures with depths reaching several dozen layers. Despite being limited to a maximum search depth of 3 layers, SANE demonstrates notable competitiveness on the Computers and Photo datasets, highlighting the advantages of hyper-parameters tuning. Among existing methods, SGC+AIR and DFG-NAS achieve the highest test accuracy of 81.1% on the PubMed dataset. Surpassing them by 1.2% and 0.4%, our method CAS-DGNN (v1) and CAS-DGNN (v2) achieve test accuracy of 82.3% and 81.5%, marking the highest improvement across all eight small and medium datasets.

2) Search Results on Larger Datasets: This part showcases the search results on the datasets of Flickr and ogbn-arxiv. As detailed in the design of the residual connections search space, our focus is solely on determining the necessity of each residual connection in Initial Residual connection method on larger datasets. As presented in Table X, our method CAS-DGNN (v2) attains the highest test accuracy on both Flickr and ogbn-arxiv, achieving improvements of 1.1% and 0.9% over the state-of-the-art method DFG-NAS, respectively. This underscores the scalability of our method. If researchers wish to apply our method to search larger datasets, they only need to adjust the residual connection methods used and set an appropriate maximum search depth.

3) Search Cost Comparison: In this section, we compare the search costs of SANE, DFG-NAS, and our method CAS-DGNN. As shown in Fig. 9 across the eight small and medium datasets, the average search cost of CAS-DGNN (v1) is 6.1× and 2.2× higher than that of SANE and DFG-NAS, respectively. Specifically, due to the fixed model depth of 3 layers in SANE, its search time across the eight small and medium datasets does not exceed one hour. On the other hand, the search depth of DFG-NAS can reach up to 45 layers, resulting in a search cost of approximately 2 to 3 hours. CAS-DGNN (v1) consumes significantly more time than existing methods, peaking at a maximum of 6.8 hours on PubMed. Meanwhile, the average search costs of CAS-DGNN (v2) are only 0.5× and 1.1× those of CAS-DGNN (v1) and DFG-NAS, respectively. Notably, the search cost of CAS-DGNN (v2) is even less than DFG-NAS on the PubMed dataset. Combining the results from Table IX, it is evident that CAS-DGNN (v2) outperforms DFG-NAS in both test accuracy and search efficiency, further emphasizing the superiority of our method.

Additionally, on the large datasets of Flickr and ogbn-arxiv, the search cost of CAS-DGNN (v2) on an NVIDIA V100 GPU totals 41.2 and 63.3 hours, respectively, which is only approximately 1.3× that of DFG-NAS. Meanwhile, in Fig. 10, we compare the best results over search time of SANE, DFG-NAS, and CAS-DGNN(v2). We observe that CAS-DGNN(v2) tends to have lower initial accuracy due to its broader search space, but it exhibits strong late-stage performance, consistently achieving the best search outcomes. Despite the longer total search time required by CAS-DGNN(v2), it surpasses existing methods in less time. For instance, on the Flickr dataset, although the total search time of CAS-DGNN(v2) exceeds 40 hours, it only takes approximately 24 and 27 hours to outperform SNAE and DFG-NAS, respectively, further demonstrating the effectiveness of our approach.

[1]https://github.com/pyg-team/pytorch_geometric
[2]https://github.com/hyperopt/hyperopt

TABLE IX
PERFORMANCE (TEST ACC%± STD) ON THE NODE CLASSIFICATION TASK

Methods	**Cora** (Small)	**CiteSeer** (Small)	**PubMed** (Medium)	**Computers** (Medium)	**Photo** (Medium)	**CS** (Medium)	**Physics** (Medium)	**DBLP** (Medium)
Basic GNNs								
GCN [2]	81.3±0.6	71.1±0.2	78.8±0.4	89.2±0.2	93.5±0.3	92.4±0.3	95.5±0.3	85.5±0.4
GraphSAGE [3]	79.2±0.6	71.6±0.5	77.4±0.5	88.6±0.3	93.2±0.3	92.5±0.4	94.3±0.2	84.2±0.3
GAT [4]	82.9±0.2	70.8±0.5	79.1±0.1	90.5±0.3	93.5±0.3	93.5±0.2	95.2±0.4	86.4±0.3
SGC [5]	81.7±0.2	71.3±0.2	78.8±0.1	88.1±0.2	91.2±0.3	90.8±0.3	95.0±0.3	84.7±0.4
APPNP [6]	83.1±0.5	71.8±0.4	80.1±0.2	89.3±0.4	94.1±0.5	92.1±0.4	94.1±0.6	86.1±0.4
Deep GNNs								
DAGNN [7]	84.3±0.2	73.3±0.6	80.5±0.5	90.5±0.6	94.5±0.4	92.7±0.5	96.1±0.6	86.6±0.4
GCNII [8]	85.1±0.5	73.4±0.6	80.2±0.5	92.3±0.5	95.8±0.4	94.9±0.4	96.5±0.3	86.5±0.4
GCN+AIR [9]	83.2±0.7	71.6±0.6	80.2±0.7	90.1±0.4	94.6±0.4	92.1±0.6	95.8±0.4	87.1±0.4
SGC+AIR	84.0±0.6	72.0±0.5	81.1±0.6	90.3±0.4	94.9±0.5	93.2±0.4	95.6±0.5	86.2±0.3
APPNP+AIR	83.8±0.6	73.4±0.5	81.0±0.6	92.0±0.5	95.3±0.4	95.4±0.4	96.6±0.4	85.5±0.4
NAS Methods								
Auto-GNN [16]	83.6±0.3	73.8±0.7	79.7±0.4	92.3±0.5	95.1±0.4	94.7±0.6	96.2±0.4	86.5±0.4
GraphNAS [35]	83.7±0.4	73.5±0.3	80.5±0.3	92.1±0.4	94.8±0.6	94.0±0.5	96.5±0.4	86.6±0.5
DFG-NAS [17]	**85.2±0.2**	**74.1±0.4**	81.1±0.3	93.1±0.5	96.1±0.4	95.1±0.3	**97.2±0.4**	**87.3±0.4**
SANE [18]	84.1±0.4	73.8±0.5	80.8±0.3	**93.2±0.4**	**96.5±0.5**	93.2±0.6	95.8±0.5	86.8±0.4
CAS-DGNN (v1)	**85.6±0.4**	**74.7±0.3**	**82.3±0.4**	**93.8±0.5**	**97.0±0.4**	**96.1±0.6**	**97.5±0.3**	**87.9±0.2**
CAS-DGNN (v2)	84.8±0.4	73.8±0.3	**81.5±0.4**	92.5±0.5	95.8±0.4	**95.8±0.6**	96.8±0.3	86.8±0.5

The top two results are highlighted in bold black font.

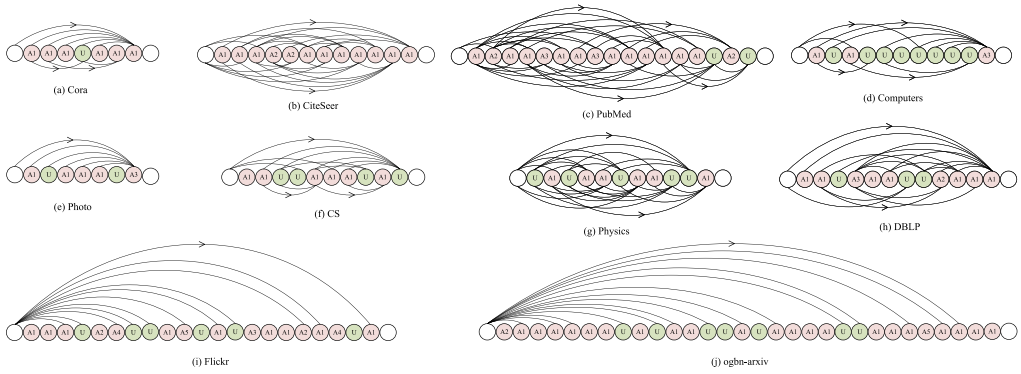

Fig. 8. Searched architectures on different datasets. (The white balls at both ends represent the preprocessing and classification layers, respectively. A1, A2, A3, A4, and A5 correspond to aggregation types of GCN, SAGE-MEAN, SAGE-MAX, GAT, and GIN, respectively, while U represents the update operator. All models and residual connections are directed from left to right. For Flickr and ogbn-arxiv datasets, we present search results from CAS-DGNN (v2). For other small and medium sized datasets, we present search results from CAS-DGNN (v1).).

TABLE X
SEARCH RESULTS (TEST ACC%± STD) ON LARGER DATASETS

Methods	**Flickr** (Large)	**ogbn-arxiv** (Large)
GCN	50.9±0.5	71.5±0.3
GCNII	42.5±0.6	72.0±0.3
DFG-NAS	**52.0±0.4**	**72.3±0.2**
SANE	51.5±0.3	71.9±0.4
CAS-DGNN (v2)	**53.1±0.4**	**73.2±0.3**

TABLE XI
SEARCHED HYPER-PARAMETERS

Datesets	Hyper-parameters [hidden size, dropout, learning rate, L2 norm, epochs]
Cora	[256, 0.7, 0.02, 5e-4, 1000]
CiteSeer	[128, 0.8,0.01, 5e-4, 500]
PubMed	[256, 0.5, 0.01, 5e-4, 1000]
Computers	[512, 0.4, 0.001, 2e-4, 1000]
Photo	[512, 0.4, 0.001, 5e-4, 500]
CS	[32, 0.6, 0.01, 2e-4, 200]
Physics	[256, 0.7, 0.02, 2e-4, 1000]
DBLP	[32, 0.5, 0.01, 2e-4, 200]
Flickr	[512, 0.7, 0.01, 5e-4, 500]
ogbn-arxiv	[512, 0.4, 0.001, 5e-4, 1000]

The above analysis demonstrates the successful and significant optimization of our residual connections search process. Despite the increased search cost associated with our method, it remains within an acceptable range and does not escalate to an extravagant level, such as several thousand GPU days.

4) Searched Architectures: We visualize the searched operator composition, aggregation types, and residual connections (top-1) by CAS-DGNN (v1) on small and medium datasets and CAS-DGNN (v2) on large datasets in Fig. 8. And Table XI presents the corresponding hyper-parameters combinations. These architectures showcase the data-dependent

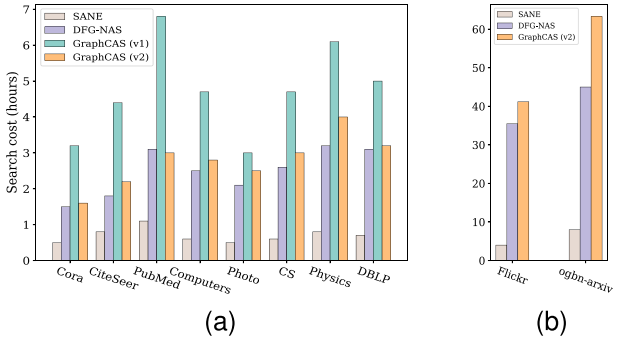

Fig. 9. Search cost (hours on NVIDIA V100 GPU) comparison of SANE, DFG-NAS, and our method. (a) Small and medium datasets. (b) Large datasets.

TABLE XII
THE DEPTH OF BEST GNN ARCHITECTURES DISCOVERED THROUGH NAS

Datesets	Depth			
	Auto-GNN	SANE	DFG-NAS	CAS-DGNN
Cora	3	3	8	7
CiteSeer	3	3	14	12
PubMed	3	3	22	17
Computers	3	3	16	11
Photo	3	3	8	7
CS	3	3	12	10
Physics	3	3	14	11
DBLP	3	3	19	12
Flickr	3	3	38	21
ogbn-arxiv	3	3	43	30

characteristics, highlighting the necessity of employing the NAS method for deep GNNs.

Upon closer examination of operator compositions and aggregation methods, we observe that the number of aggregate operators significantly outweighs the number of update operators across most datasets. Particularly noteworthy is the dataset of CiteSeer, where all 12 mid-layers consist solely of aggregate operators. An exception to this trend is observed only in the Computer dataset, where the number of update operators exceeds that of aggregate operators. As for aggregation types, GCN dominates, followed by SAGE-MEAN and SAGE-MAX. Notably, the aggregation types of GAT and GIN only appear on the datasets of Flickr and ogbn-arxiv, and occur infrequently. This suggests that the methods requiring the computation of adaptive coefficients during training, such as attention score in GAT and ϵ in GIN, may not be suitable for the new operator compositions. Meanwhile, a notable finding from the hyper-parameters search results is the maximum dropout value reaching 0.8. This indicates that there is a significant amount of information redundancy in deep GNNs.

Furthermore, the depth of the architectures searched by our method, CAS-DGNN, along with other methods (Auto-GNN, SANE, and DFG-NAS), is shown in Table XII. It is important to note that, since we developed two versions of CAS-DGNN, only the depth of the best search results is presented here. The results show that the depth of Auto-GNN and SANE remains consistently fixed at 3 across all datasets, whereas DFG-NAS

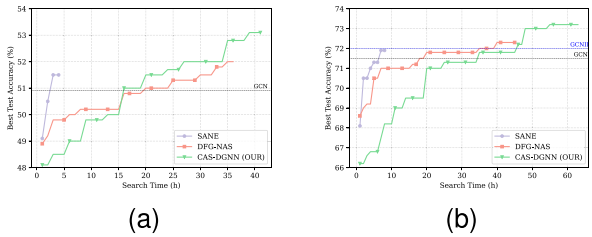

Fig. 10. Best results over search time. (a) Flickr. (b) ogbn-arxiv.

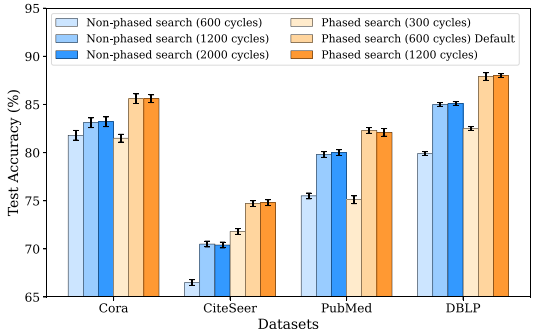

Fig. 11. Comparison of results between phased and non-phased search methods with different evolutionary cycles. (The caps on top of the bar represent the standard deviation.).

reaches the deepest architectures. Our method achieves depths that lie between those of DFG-NAS and Auto-GNN/SANE. These findings indicate that, compared to Auto-GNN and SANE, our approach is capable of discovering deeper models. Furthermore, in contrast to DFG-NAS, the broader search space of our method proves beneficial in identifying GNN models that balance both effectiveness and efficiency. This suggests that while model depth is essential in our approach, the comprehensiveness of the search space plays a more critical role in enhancing overall performance.

C. Ablation Study

1) Phased vs. Non-Phased: The distinction between the phased and the non-phased methods refers to the search for operator composition, aggregation types, and residual connections. In both methods, the search for hyper-parameters remains a separate phase. In the non-phased method, we utilize evolutionary algorithms to simultaneously search for operator composition, aggregation types, and residual connections. We simultaneously perform crossover and mutation operations on the operator list and residual matrix throughout the search process. Additionally, the evolutionary cycles are set to 300, 600, and 1200 for phased search, while the evolutionary cycles are set to 600, 1200, and 2000 for non-phased search. The default method we employ in Table IX is the phased search with 600 evolutionary cycles.

As illustrated in Fig. 11, we present the search results on the datasets of Cora, CiteSeer, PubMed, and DBLP as examples. If we focus solely on the non-phased search method (first three bars in each dataset), we observe that the search results reach the optimal when 1200 cycles are done. Further increasing the

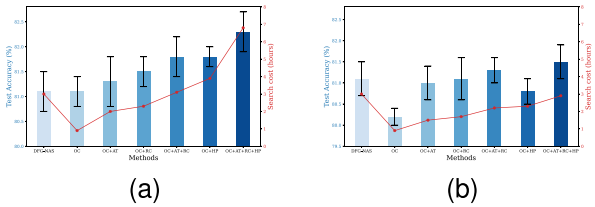

Fig. 12. Comparison of search results breakdown for each phase on PubMed dataset. (a) CAS-DGNN v1. (b) CAS-DGNN v2. (OC, AT, RC, and HP represent operation compositions, aggregation types, residual connections, and hyper-parameters search in our method, respectively.).

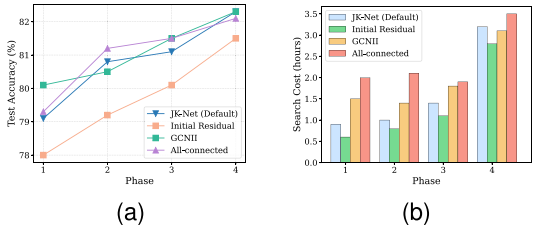

Fig. 13. Comparison between different residual connections in CAS-DGNN v1 on the PubMed dataset. (a) Test accuracy comparison. (b) Search cost comparison.

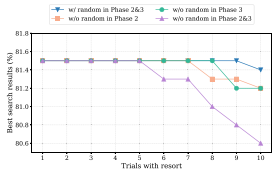

Fig. 14. Robustness of randomization strategy. (CAS-DGNN v2 on PubMed dataset.).

cycles to 2000 hardly leads to any improvement. In contrast, for the phased search method (the last three bars in each dataset), the results reach the optimal with 600 cycles, and increasing the cycles to 1200 has minimal impact on the final results. However, decreasing the cycles for both methods noticeably deteriorates the search results. This indicates that 600 cycles are sufficient to explore phased search spaces adequately. Additionally, the phased search method (600 cycles) demonstrates significantly higher test accuracy across the four datasets compared to the non-phased search method (1200 cycles), resulting in an average improvement of approximately 3.0%. Nevertheless, it is important to acknowledge that the non-phased search method might discover superior model architectures with more evolutionary cycles. However, due to limited computational resources, we did not employ larger evolutionary cycles, such as 10,000 or more.

This experiment underscores the necessity of phased search, especially when dealing with extensive search spaces.

2) Breakdown of Each Phase: In this part, we provide a detailed analysis of the impact of each phase on the overall search results. Since all our work is based on the operator composition in phase 1, we continuously add additional elements to this foundation to demonstrate the variations in search results. While DFG-NAS also explores operator composition, our phase 1 search builds upon their work, thus we include DFG-NAS in our comparison. The overall trend in Fig. 12 indicates that as more search components are added on top of the operator composition, the model accuracy improves, albeit with a noticeable increase in search cost.

Upon closer examination, the search cost for operator composition in both CAS-DGNN v1 and v2 is significantly lower than that of DFG-NAS. CAS-DGNN v1 achieves nearly the same search accuracy as DFG-NAS, while the search results for v2 are somewhat lower, indicating that using only an initial residual connection for operator composition search may lead to performance loss. In CAS-DGNN v1, the search cost for OC+AT and OC+RC is significantly lower than that of DFG-NAS, while search accuracy demonstrates a noticeable improvement. Additionally, the results of OC+AT+RC are similar to those of OC+HP, though the search cost for OC+HP is noticeably higher, highlighting the time-consuming nature of hyper-parameter search. While OC+AT+RC+HP achieves the highest search accuracy, it also incurs the highest search cost, reaching 6.8 hours.

In CAS-DGNN v2, the accuracy of OC+AT+RC also surpasses that of DFG-NAS, suggesting that hyper-parameter search is an optional search component. Even without hyper-parameters search, both v1 and v2 have the potential to outperform DFG-NAS. These results further demonstrate that in practical applications, users need not utilize all four components simultaneously. Instead, they can select the components that best suit their requirements, allowing them to balance accuracy with time efficiency.

3) Different Residual Connection Methods in Phase 1&2: In our method, since we only perform residual connection searches in phase 3, the influence of different residual connection methods adopted in the first two phases on the search results is worth exploring. In this experiment, taking CAS-DGNN (v1) on the dataset of PubMed as an example, we set the residual connection methods used in the first two phases to JK-Net (default), Initial Residual, GCNII, and All-connected. As shown in Fig. 13(a), when the residual connection method is set to Initial Residual, the test accuracy obtained in each phase is significantly lower than other methods. Among the other three methods, although GCNII is slightly ahead in the initial search phase, the final search accuracy of the three methods is very close, at approximately 82.3%. Furthermore, by comparing the search cost in Fig. 13(b), the method of Initial Residual consumes the least time, while All-connected consumes the most.

Moreover, from Fig. 13(b), it is evident that the search cost in phase 4 significantly exceeds that of the first three phases. This is because the time required for Bayesian optimization to update hyper-parameters is notably slower than the crossover and mutation operations in the evolutionary algorithms. This is also one of the significant reasons why we do not use Bayesian optimization in the first three phases.

4) Effectiveness of Randomization Strategy in Phase 2&3: In our EA algorithm, we introduce a randomization strategy to build the initial population set Q in phases 2 and 3. Here, we explore

the impact of the randomization strategy on our evolutionary algorithm. As shown in Fig. 14, we repeated the search process of CAS-DGNN(v2) on the PubMed dataset 10 times, systematically removing the randomization strategy in phases 2 and 3. We observe that when the randomization strategy was present in both phases, the algorithm demonstrated the highest robustness, consistently achieving optimal search results. However, as we gradually removed the randomization strategy, the algorithm's robustness declined, especially when it was entirely removed from both phases; in this case, only 5 out of 10 trials achieved the peak search accuracy of 81.5%. This highlights the substantial impact of the randomization strategy on the robustness of our method.

VI. Conclusion

This paper focuses on addressing the critical challenge of designing effective deep GNNs by introducing an innovative NAS method. We have designed a highly comprehensive search space and significantly reduced its size by employing a phased search strategy and optimizing the search for residual connections. Our method is highly compatible with deep GNNs and exhibits strong scalability. Additionally, we have demonstrated that a phased search strategy is essential when dealing with large search spaces, particularly in the context of deep GNNs. The main limitation of our method lies in its high search cost. In the future, we will continue to explore more effective search spaces and more efficient search strategies in deep GNNs. Moreover, it is still worthwhile to continue exploring GNNs with depths reaching hundreds of layers.

References

[1] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. 29th Conf. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3837–3845.

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

[3] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 30th Conf. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

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

[5] F. Wu, A. H.S. Jr, T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Inst. Conf. Mach. Learn.*, 2019, pp. 6861–6871.

[6] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 1–15.

[7] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 338–348.

[8] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. 37th Inst. Conf. Mach. Learn.*, 2020, pp. 1725–1735.

[9] W. Zhang et al., "Model degradation hinders deep graph neural networks," in *Proc. 28th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2022, pp. 2493–2503.

[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Representations*, 2017, pp. 1–16.

[11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 1–13.

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

[13] C. Liu et al., "Progressive neural architecture search," in *Proc. Eur. Conf. Comp. Vis.*, 2018, pp. 19–35.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Inst. Conf. Mach. Learn.*, 2018, pp. 4092–4101.

[15] Y. Yang, Z. Shen, H. Li, and Z. Lin, "Optimization-inspired manual architecture design and neural architecture search," *Sci. China Inf. Sci.*, vol. 66, 2023, Art. no. 212101.

[16] K. Zhou, X. Huang, Q. Song, R. Chen, and X. Hu, "Auto-GNN: Neural architecture search of graph neural networks," *Front. Big Data*, vol. 5, 2022, Art. no. 1029307.

[17] W. Zhang, Z. Lin, Y. Shen, Y. Li, Z. Yang, and B. Cui, "Deep and flexible graph neural architecture search," in *Proc. 39th Inst. Conf. Mach. Learn.*, 2022, pp. 26362–26374.

[18] Z. Huan, Y. Quanming, and T. Weiwei, "Search to aggregate neighborhood for graph neural network," in *Proc. 37th IEEE Int. Conf. Data Eng.*, 2021, pp. 552–563.

[19] M. Shi et al., "Genetic-GNN: Evolutionary architecture search for graph neural networks," *Knowl. Based Syst.*, vol. 247, pp. 108752, 2022.

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

[21] X. Zheng, M. Zhang, C. Chen, Q. Zhang, C. Zhou, and S. Pan, "Auto-HeG: Automated graph neural network on heterophilic graphs," in *Proc. 32nd Int. Conf. World Wide Web*, 2023, pp. 611–620.

[22] L. Wei, H. Zhao, and Z. He, "Designing the topology of graph neural networks: A novel feature fusion perspective," in *Proc. 31st Int. Conf. World Wide Web*, 2022, pp. 1381–1391.

[23] Z. Zhang, X. Wang, and W. Zhu, "Automated machine learning on graphs: A survey," in *Proc. 30th Int. Joint Conf. Artif. Intell.*, 2021, pp. 4704–4712.

[24] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[25] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. 35th Inst. Conf. Mach. Learn.*, 2018, pp. 5449–5458.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comp. Vis. Patt. Recognit.*, 2016, pp. 770–778.

[27] W. Cao, C. Zheng, Z. Yan, and W. Xie, "Geometric deep learning: Progress, applications and challenges," *Sci. China Inf. Sci.*, vol. 65, no. 2, 2022, Art. no. 126101.

[28] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, no. 3-4, pp. 219–354, 2018.

[29] X. Zhou, H. Wang, W. Peng, B. Ding, and R. Wang, "Solving multi-scenario cardinality constrained optimization problems via multi-objective evolutionary algorithms," *Sci. China Inf. Sci.*, vol. 62, no. 9, pp. 192104:1–192104:18, 2019.

[30] A. Darwish, A. E. Hassanien, and S. Das, "A survey of swarm and evolutionary computing approaches for deep learning," *Artif. Intell. Rev.*, vol. 53, no. 3, pp. 1767–1812, 2020.

[31] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 550–570, Feb. 2023.

[32] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. 32nd Inst. Conf. Mach. Learn.*, 2015, pp. 2113–2122.

[33] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. 24th Conf. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[34] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Inst. Conf. Mach. Learn.*, 2013, pp. 115–123.

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

[36] H. Zhao, L. Wei, and Q. Yao, "Simplifying architecture search for graph neural network," 2020, *arXiv: 2008.11652*.

[37] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3438–3445.

[38] E. Chien, J. Peng, P. Li, and O. Milenkovic, "Adaptive universal generalized pagerank graph neural network," in *Proc. 9th Int. Conf. Learn. Representations*, 2021, pp. 1–24.
[39] W. Feng et al., "Graph random neural networks for semi-supervised learning on graphs," in *Proc. 33rd Conf. Adv. Neural Inf. Process. Syst.*, 2020, pp. 22092–22103.
[40] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–17.
[41] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in GNNs," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–17.
[42] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 9266–9275.
[43] W. Zhang et al., "PaSca: A graph neural architecture search system under the scalable paradigm," in *Proc. 31st Int. Conf. World Wide Web*, 2022, pp. 1817–1828.
[44] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "GraphNAS: Graph neural architecture search with reinforcement learning," 2019, *arXiv: 1904.09981*.
[45] M. Nunes and G. L. Pappa, "Neural architecture search in graph neural networks," in *Proc. 9th Braz. Conf. Intel. Syst.*, 2020, pp. 302–317.
[46] Y. Li and I. King, "Autograph: Automated graph neural network," in *Proc. 27th Int. Conf. Neur. Inf. Process.*, 2020, pp. 189–201.
[47] S. Jiang and P. Balaprakash, "Graph neural network architecture search for molecular property prediction," in *Proc. IEEE Int. Conf. Big Data*, 2020, pp. 1346–1353.
[48] Y. Li, Z. Wen, Y. Wang, and C. Xu, "One-shot graph neural architecture search with dynamic search space," in *Proc. Conf. AAAI*, 2021, pp. 8510–8517.
[49] C. Guan, X. Wang, and W. Zhu, "AutoAttend: Automated attention representation search," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 3864–3874.
[50] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.
[51] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "EENA: Efficient evolution of neural architecture," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2019, pp. 1891–1899.
[52] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Proc. 25th Conf. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2960–2968.
[53] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.
[54] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. 33rd Inst. Conf. Mach. Learn.*, 2016, pp. 40–48.

Yukang Dong received the BS degree from the Hefei University of Technology, in 2017. He is currently working toward the postgraduate degree with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). His research interests include model and system optimization for graph neural networks and sparse computing.

Fanxing Pan received the BS degree from the Wuhan University of Technology, in 2021. She is currently working toward the postgraduate degree with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). Her research interests include data flow diagrams and graph processing.

Yi Gui received the master's degree from the Huazhong University of Science and Technology, in 2022. He is currently working toward the PhD degree with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). His research interests include code intelligence and large language models.

Wenbin Jiang (Member, IEEE) received the PhD degree from the Huazhong University of Science and Technology, in 2004. He is a professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). He was a visiting scholar with UCLA (University of California, Los Angeles), in 2014, and with the NUS (National University of Singapore), in 2017. His research interests include deep learning systems, parallel and distributed computing, and graph processing.

Yao Wan received the PhD degree from Zhejiang University, in 2019. He is currently an associate professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). His research interests include code intelligence, natural language processing, programming languages, and artificial intelligence.

Ran Zheng received the PhD degree from the Huazhong University of Science and Technology, in 2006. She is currently an associate professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). Her research interests include high-performance computing and distributed computing.

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, in 1994. He is a chair professor of computer science and engineering with the Huazhong University of Science and Technology (HUST), in China. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked with the University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He was awarded the Excellent Youth Award from the National Science Foundation of China in 2001. He is a fellow of CCF, and a life member of ACM. He has co-authored more than 20 books and published over 900 research papers. His research interests include computer architecture, parallel and distributed computing, Big Data processing, data storage, and system security.