TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 04/10 pp692–708
DOI: 10.26599/TST.2021.9010057
Volume 27, Number 4, August 2022

Graph Neural Architecture Search: A Survey

Babatounde Moctard Oloulade, Jianliang Gao*, Jiamin Chen, Tengfei Lyu, and Raeed Al-Sabri

Abstract: In academia and industries, graph neural networks (GNNs) have emerged as a powerful approach to graph data processing ranging from node classification and link prediction tasks to graph clustering tasks. GNN models are usually handcrafted. However, building handcrafted GNN models is difficult and requires expert experience because GNN model components are complex and sensitive to variations. The complexity of GNN model components has brought significant challenges to the existing efficiencies of GNNs. Hence, many studies have focused on building automated machine learning frameworks to search for the best GNN models for targeted tasks. In this work, we provide a comprehensive review of automatic GNN model building frameworks to summarize the status of the field to facilitate future progress. We categorize the components of automatic GNN model building frameworks into three dimensions according to the challenges of building them. After reviewing the representative works for each dimension, we discuss promising future research directions in this rapidly growing field.

Key words: graph neural network; neural architecture search; automated machine learning; geometric deep learning

1 Introduction

Deep learning (DL) models have achieved superior performance in fields such as computer vision[1, 2], natural language processing[3–5], and speech recognition[6, 7] because of their ability to capture hidden patterns and leverage the statistical properties of data. DL models, e.g., recurrent neural networks (RNNs) and convolutional neural networks (CNNs), have been widely used by researchers for machine learning (ML) tasks and have solved a wide variety of problems in many cross disciplines, including but not limited to medicine[8, 9], agriculture[10], commerce[11, 12], and finance[13]. However, these models, which are predominantly applied to data represented as a regular grid in the Euclidean space, fail to extract latent representations from graph data. This drawback is due to graphs being non-Euclidean data, which cannot be represented in an n-dimensional linear space. Graphs are nongrid-like structured data with nonfixed size and unordered nodes. Thus, some important operations, such as convolutions computed in Euclidean space, are difficult to implement, such as in the case of transforming computation results into graph data. Graph neural networks (GNNs) successfully tackle this problem and have thus become a very popular approach in academia and in industries. Graph data have recently become ubiquitous in our lives. The omnipresence of graph data has boosted the research on graph pattern recognition and graph mining. Naturally, many GNN models have been proposed for various tasks, ranging from node classification[14, 15] and link prediction[16, 17] to graph clustering[18]. GNN model components include GNN architecture components (ACs), such as attention function, aggregation function, and activation function, and hyperparameters (HPs), such as drop out and learning rate. In traditional GNNs, the selection of these components is a problem that is tackled on the basis of expert experience and unwritten rules of thumb. Hence, achieving optimized combinations of these components is difficult. Building the best GNN models for targeted

• Babatounde Moctard Oloulade, Jianliang Gao, Jiamin Chen, Tengfei Lyu, and Raeed Al-Sabri are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China. E-mail: {oloulademoctard, gaojianliang, chenjiamin, tengfeilyu, alsabriraeed}@csu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2021-05-15; revised: 2021-07-15; accepted: 2021-07-30

tasks requires testing several GNN architectures before selecting the best one. Such a process is time-consuming or impossible to apply to big graph data. Meanwhile, different tasks need different tuning of HPs to obtain optimal results. Motivated by the recent success of neural architecture search (NAS) for RNN and CNN models, many researchers have attempted to apply automated ML (AutoML) approaches to the graph domain. Specifically, many researchers have focused on automatically finding the best GNN models for targeted tasks, thereby raising a new topic in the field. This research topic is commonly referred to as Graph Neural Architecture Search, which we denote as Graph-NAS in this article. The growing interest in GNN models has led to increasing attention to Graph-NAS. In this work, we aim to provide a comprehensive review of Graph-NAS frameworks, which not only achieve promising performance for many applications of GNNs but also show potential as a unanimous approach to constructing GNN models.

Differences with existing surveys. The studies involving Graph-NAS surveys are limited. Xie et al.[19] provided an exhaustive enumeration of optimization strategies in neural architecture search. Although their work slightly tackled Graph-NAS, it is not emphasized on Graph-NAS challenges and was published at the time when there were just a few works on Graph-NAS. Nunes and Pappa[20] analyzed the optimization strategies for existing Graph-NAS frameworks. The authors compared reinforcement-based optimization learning and evolutionary algorithm-based optimization without providing a critical analysis of other Graph-NAS challenges, e.g., search space design. Since their work, other frameworks have been proposed. The current work differs from existing studies as it is aimed at providing a comprehensive and up-to-date survey with a critical analysis of Graph-NAS methods and applications.

Recently, Zhang et al.[21] conducted a similar comprehensive survey of AutoML for graphs. In their work, NAS for GNNs and hyperparameter optimization (HPO) for GNNs were reviewed separately. By contrast, we argue herein that HPO for GNNs is included in Graph-NAS.

Recent frameworks consider HPs and ACs in the search process[22–24]. Instead of a disjoint review of search algorithms for Graph-NAS and HPO for GNNs, such as that by Zhang et al.[21], the current study presents a comprehensive survey of Graph-NAS frameworks from a global view. We categorize Graph-NAS frameworks from the perspective of challenges. We present detailed descriptions of their methods by challenge, analyze their advantages and limitations, and provide necessary comparisons. This article will not only serve as a reference for experts who would like to compare Graph-NAS frameworks but also establish a clear outline for researchers who want to enter this new and interesting field.

Contributions of this article. This article makes significant contributions, which are summarized as follows:

- **Categorization:** We categorize the constituents of existing Graph-NAS frameworks in three dimensions on the basis of the challenges in building them: (1) search space, which defines a set of GNN models; (2) search algorithm, which defines how to find the best GNN model from the search space; and (3) performance evaluation, which decides how good a GNN model is. Our categorization provides a unified representation to present different strategies in different Graph-NAS frameworks. One can easily make a distinction between different frameworks by using our categorization.
- **Comprehensive review:** We provide a comprehensive overview of existing Graph-NAS frameworks from a global view, along with the known challenges. We present detailed descriptions of their methods by challenge, analyze their advantages and limitations, and provide necessary comparisons.
- **Promising research directions:** We discuss various aspects of Graph-NAS frameworks, analyze their limitations, and propose promising future research directions in terms of efficiency and scalability trade-off.

Organization of this article. The remaining part of this work is structured as follows: In Section 2, we provide preliminary definitions, present an overview of GNNs, introduce Graph-NAS, and expose the existing frameworks for Graph-NAS, along with their strengths and weaknesses. In Section 3,we present the categorization of the components of existing Graph-NAS frameworks. In Sections 4, 5, and 6, we provide critical analysis of the existing frameworks according to the dimensions of search space, search algorithm, and performance evaluation, respectively. In Section 7, we present the most common applications of Graph-NAS and provide a comparative picture of exiting frameworks. Finally, we conclude the survey and suggest some promising future research directions.

2 Definitions and Background

In this section, we present graph-related concepts, list common notations, and outline the background of Graph-NAS.

2.1 Definitions and notations

Graph data consist of a set of pairwise relations between objects. The objects are called nodes or vertices, and a relation between two nodes is called an edge. For example, in e-commerce, graph-based representation can be used to show the interaction between a user and a product. In a social network system, nodes are denoted by users, and edges represent relations between users. An edge can be directed if it represents an asymmetric dependency between two nodes; otherwise, it is undirected. An undirected edge can be treated as two directed edges. A graph with only undirected edges is called an undirected graph; otherwise, it is a directed graph. Mathematically, a graph is commonly represented as $G = (V, E)$, where V is a set of nodes, and E is a set of edges. $v \in V$ is used to denote a node and $e_{uv} = (u, v) \in E$ denotes an edge pointing from u to v. The neighborhood of a node v is denoted as $N(v) = \{u \in V | (v, u) \in E\}$. The adjacency matrix of a graph commonly denoted $\boldsymbol{A}$ is an $n \times n$ matrix, where $\boldsymbol{A}_{uv} = \mathbf{1}$ if $e_{uv} \in E$, and $\mathbf{0}$ otherwise; and n is the total number of nodes. A node u and an edge e_{uv} can carry properties, also called features, denoted by vectors $\boldsymbol{x}_u$ and $\boldsymbol{x}^e_{uv}$, respectively. The node features of a graph are represented by a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, where d represents a node feature size. The edge features of a graph are represented by a matrix $\boldsymbol{X}^e \in \mathbb{R}^{m \times c}$, where m and c represent the number of edges and an edge feature size respectively. Furthermore, a graph can contain nodes having different types of edges that denote different types of relationships. A graph with the same types of nodes and the same types of relationships is called a homogeneous graph; otherwise, it is a heterogeneous graph. Take for example, a social network graph in which the nodes only represent users and edges, in such a case, their friendship is a homogeneous graph. For a knowledge graph in which a node represents a player or a game and an edge represents the relationship between players or the dependency between a player and a game, it is described as a heterogeneous graph. Throughout this article, we use bold lowercase characters to represent vectors, and bold uppercase characters to represent matrices. Table 1 shows the notations used in this article.

Table 1 Notations.

Notation	Description
G	A graph
V	Set of nodes in a graph
v	A node $v \in V$
E	Set of edges in a graph
e_{ij}	An edge, $e_{ij} \in E$
$N(v)$	Neighborhood of a node v
n	Number of nodes, $n = \|V\|$
m	Number of edges, $m = \|E\|$
$\boldsymbol{A}$	$A(n \times n)$ graph adjacency matrix
d	Dimension of a node feature vector
c	Dimension of an edge feature vector
$\boldsymbol{X} \in \mathbb{R}^{n \times d}$	Feature matrix of a graph
$\boldsymbol{x}_v \in \mathbb{R}^n$	Feature vector of a node v
$\boldsymbol{X}^e \in \mathbb{R}^{m \times c}$	Edge feature matrix of a graph
$\boldsymbol{x}^e_{uv} \in \mathbb{R}^c$	Feature vector of the edge
$\boldsymbol{H} \in \mathbb{R}^{n \times b}$	Node hidden feature matrix
$\boldsymbol{h}_v \in \mathbb{R}^b$	Hidden feature vector of node v
k	Layer index
$\boldsymbol{W}, w, \theta$	Learnable model parameters
S	A search space
s	Sub-model generated from S
R_s	Reward of a sub-model generated from S
D	Graph dataset
$D_{\text{val}} \subset D$	Validation set

2.2 Background

2.2.1 Background of graph neural networks

GNNs are a kind of DL method applied to graphs to learn graph representation. Early studies on GNNs were motivated by the work of Sperduti and Starita[25] which applied neural networks to directed graphs. The concept of GNNs had been outlined by two works[26, 27] which enhanced existing neural networks to directly process the most practically useful types of graphs. The objective is to learn the representation of a target node or graph by propagating neighborhood information in an iterative way until convergence. These studies fall into the category of recurrent graph neural networks (RecGNNs) and are computationally expensive[28]. Although efforts have been exerted to alleviate these weaknesses, many other researchers have relied on the success of CNNs in computer vision to develop new convolution approaches that are applicable to graphs, which are called convolutional GNNs (ConvGNNs). ConvGNNs have two branches: spectral-based approach models[29, 30] and the spatial-based approach models[31–33]. In the spectral-based approach, models are based on spectral

graph theory, and signals on a graph are filtered using the eigendecomposition of Laplacian graphs[34]. In the spatial-based approach, convolution operations are directly performed on the graphs, and graph convolution is represented as a combination of feature information from node neighborhoods.

Modern GNN models follow a neighborhood aggregation (or message passing) scheme[35], where the representation vector of a node is determined by recursively aggregating and transforming the representation vectors of its neighboring nodes. The goal is to learn a function of features on a graph G. The function takes the following as input: A feature description $\boldsymbol{x}_v$ for each node $v \in V$ summarized in a feature matrix $\boldsymbol{X}$, and a representative description of the graph G structure in matrix form, typically in the form of an Adjacency matrix $\boldsymbol{A}$ (or some functions) and return node-level output or graph level output. In this calculation, node feature representation vectors are chosen according to the application. They can be word vector when working with a knowledge base, or pixel value when working with images, or a combination of image feature and word vector when working with scenes. As the objective is to obtain a good vector representation over time, an aggregation is applied recursively over node neighbors. After k round(s) ($k = 1, 2, \ldots, K$) of aggregation, a node's representation captures the structural information within its k-hop network neighborhood. Formally, given a graph G, the feature vector of its node v at the k-th iteration, representing the k-th layer of a GNN model is defined as follows:

$$\boldsymbol{h}_v^{(k)} = ACT^{(k)}(\boldsymbol{W}_k AGG(\{\boldsymbol{h}_u^{k-1}, \forall u \in N(v)\}), \gamma) \tag{1}$$

$$\gamma = \boldsymbol{B}_k \boldsymbol{h}_v^{k-1},\ \boldsymbol{h}_v^0 = \boldsymbol{x}_v \tag{2}$$

where $k \in [1, K]$ is the number of iteration, $\boldsymbol{W}_k$ is a trainable weight matrix in the k-th layer, $\boldsymbol{B}_k$ is a bias, $\boldsymbol{B}_k \boldsymbol{h}_v^{k-1}$ is a self-loop activation for node v, $ACT^{(k)}(\cdot)$ is a non-linear activation function in the k-th layer, e.g., Rectified Linear Unit (ReLU), sigmoid, and $AGG^{(k)}(\cdot)$ is an aggregation function in the k-th layer, e.g., max-pooling, sum. The output of the last layer is commonly utilized to denote the final representation for each node. For graph-level tasks, pooling is applied to layer representations. The choice of $AGG^{(k)}(\cdot)$ and $ACT^{(k)}(\cdot)$ is primordial and is the key to successful graph representation learning by a GNN model. The main challenge is the selection of the aggregation schema and the order of aggregation for each node. In recent years, many GNN variants with different neighborhood aggregation and graph-level pooling schemes have been developed, and they include graph convolutional network (GCN)[36], graph isomorphism network (GIN)[37], graph attention network (GAT)[38], and local extrema graph neural network (LEConv)[39]. These GNNs have empirically achieved superior performances in many tasks such as semantic segmentation[40], node classification[15], recommendation system[41–43], and so on. However, building the best GNN model might require numerous attempts using many GNN models, and the process is very time-consuming because of the wide array of choices for $AGG^{(k)}(\cdot)$ and $ACT^{(k)}(\cdot)$, which are sensitive to variation[37]. Meanwhile, different tasks need different levels of tuning of the components and HPs of the GNN model architecture to obtain optimal results. To overcome these difficulties, many frameworks have been proposed for Graph-NAS.

2.2.2 Background of graph neural architecture search

A model is composed of ACs and HPs according to a search space. Given a search space S composed of all the possible models and a dataset D, the objective of Graph-NAS is to find the best model $s^* \in S$ that maximizes the estimated performance $\epsilon_{\mathrm{p}}(s)$ on a validation set $D_{\mathrm{val}} \subset D$, i.e.,

$$s^* = \mathrm{argmax}_{(s \in S)} \epsilon_{\mathrm{p}}(s) \tag{3}$$

Many studies have explored Graph-NAS. One of the early works is that on Graph-NAS[44] based on reinforcement learning (RL). Graph-NAS defines a search space comprising five ACs, namely, attention function, activation function, aggregation function, number of attention heads, and hidden units with many possible choices for each function. Graph-NAS describes GNN model ACs with variable length strings generated using a recurrent network controller. Each layer of GNN models is generated from the AC search space comprising five functions. Graph-NAS uses a one-layer long short-term memory (LSTM) recurrent network as a controller to generate architecture descriptions (submodels), as a sequence of tokens, each of which represents one of the functions/actions. The generated submodel has two layers assumed to be independently designed. It is then tested with fixed HPs on the validation set D_{val}, and its accuracy metric is used as feedback of the LSTM recurrent network. In

the learning process, the controller is trained to optimize the distribution of GNNs architecture parameterized by the choice of θ and shared weight w describing the architecture. The controller parameter θ is updated using a policy gradient method with moving average based on G. C. Williams and D. C. Williams[45] defined as follows:

$$\nabla_\theta \epsilon_{\mathrm{p}}(s_{1:T}, \theta) R_s = \sum_{t=1}^{T} \epsilon_{\mathrm{P}(s_{1:T},\theta)} \nabla_\theta \ln[P(s_t | s_{(t-1):1}, \theta)(R_s - b)] \quad (4)$$

where $s_{1:T}$ is a list of functions generated by the controller for an architecture, R_s is the reward signal of the validation accuracy of architecture s, and b is the exponential moving average of the previous architecture rewards. Parameters sharing strategy are used to avoid training each model from scratch to convergence. Later, the same authors proposed a new framework[46] using the same method but performed hyperparameters optimization (HPO) by the use of hyperparameters search space to improve the framework performance. Other reinforcement-based frameworks have been proposed[22, 47], and they use slightly different search space with different mechanism for sub-models evaluation. Methods based on reinforcement learning are computationally expensive and difficult to scale.

Shi et al.[23] proposed genetic GNNs that use a two-step encoding schema with two variable length strings representing GNN architectures and HPs and one evolution step of a two-layer GNN to optimize the architecture and HPs. The architectures of GNNs are generated from an AC search space similar to that of Graph-NAS[46] and the HP search space is represented with four variable length strings representing the dropout of the first layer, the dropout of the second layer, the weight decay rate, and the learning rate. AutoGraph[48] is another evolution-based framework that randomly selects ACs from the search space and changes their component values. The newly generated model is then trained, evaluated, and used to replace the oldest model in the population. HP tuning is subsequently applied to each generated GNN by using the tree-structured Parzen estimator approach[49]. Although these evolution-based algorithms perform well, they converge slowly and are difficult to scale. Moreover, their results mainly depend on the initial population.

AutoGM[50] is a Bayesian framework based on Bayesian optimization (BO) that aims to find the best graph mining algorithm under time or metric constraints. The proponents first built UNIFIEDGM, a framework for a message passing-based graph algorithm that unifies various graph mining algorithms, including PageRank[51] and GCN[36]. Then, AutoGM finds the optimal graph algorithm by maximizing its performance under a computational time budget or minimizing its computational time under a lower bound constraint on the performance.

Recently, differentiable search strategies such as Differentiable Architecture Search (DARTS) [52] and Stochastic Neural Architecture Search (SNAS)[53] have been exploited for Graph-NAS. Zhao et al.[54] proposed a differentiable search-based framework in which the search space for a GNN is formulated as a stack of GNN layers. Every stack comprises four sub-blocks (functions), namely, a linear layer, an attention layer, an aggregation function, and an activation function. A wide variety of functions can be chosen to describe the ACs of GNN models. Probabilistic priors generated from a controller are used to train a superset of functions. They allow the controller to learn the probability distribution of the candidate functions and select the most effective function from the superset. Unlike prior frameworks, which make use of RL to iteratively evaluate architecture from the search space, the proposed framework uses Gumbel-sigmoid[55] to relax discrete architecture decisions and keep them continuous in a macro architecture search. In this way, the architecture can be optimized using gradient descent. This approach can outperform current NAS approaches applied to GNNs in terms of speed and search quality. However, it entails huge computation costs because all operations require forward and backward propagation during gradient descent while only one operation is selected. Zhao et al.[56] performed a follow-up study[52] to search an aggregate neighborhood for automatic architecture search in GNNs. They designed a differentiable search algorithm for expressive search spaces, including node and layer aggregators. Their search algorithm trains a supernet subsuming all candidate architectures, thus greatly reducing the computational cost. Ding et al.[57] proposed another differentiable search-based framework designed with an expressive directed acyclic graph search space to represent candidate meta graphs. Their framework utilizes task-dependent semantic information encoded in heterogeneous information networks and makes the search cost equivalent to training a single GNN at a time.

GraphGym[24] proposed a controlled random search (RS) based framework that uses a GNN search space

with 12 design dimensions, including GNN model ACs and HPs, proposes a task similarity metric to characterize relationships between different tasks, and uses controlled RS evaluation to efficiently analyze the trade-offs of each design dimension over tasks. For each choice of design dimension, s randomly generated GNN models are applied to tasks, and the design choices are ranked by their performance. Finally, the ranking over s GNN models is collected, the ranking distribution is analyzed, and the most important choice is retained. Although the search algorithm seems very simple, the framework performs well because of the search space. Specifically, the search space is big and compact enough as it covers most state-of-the-art GNN model structures, including HPs.

2.2.3 Graph neural architecture search versus hyperparameters optimization

Graph-NAS and HPO are AutoML subtasks. Graph-NAS aims to automate the search for the best neural network models for targeted tasks, including the GNN model ACs and HPs. Meanwhile, HPO involves optimizing the HPs of neural networks. Thus, for HPO, the architecture of the neural network is predefined, and only HP tuning is performed. In summary, HPO is a part of Graph-NAS, which ensures HP tuning for predefined neural networks.

3 Categorization

The differences among frameworks present three essential challenges: (1) designing a generalized structure of the models to be generated by the framework through a search space, (2) building a robust search algorithm to efficiently find the best architecture from the search space, and (3) evaluating architectures generated by the search algorithm. These challenges are commonly solved using three main strategies, namely, search space design, search algorithm design, and performance evaluation (Fig. 1). Generally, GNN models are generated from a search space using a search strategy and are then evaluated using a performance evaluation strategy. In this section, we present the categorization and show the details in Table 2.

3.1 Search space

A search space determines the general structure of the architectures to be chosen and contains all the possible

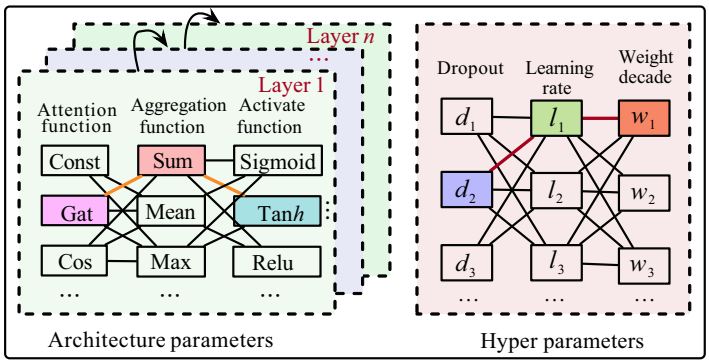

Fig. 1 Abstract illustration of Graph-NAS frameworks. The search algorithm samples an architecture s from a predefined search space S. The performance of s is determined and evaluated according to the performance evaluation strategy, and a performance evaluation is returned to the search algorithm.

Table 2 Categorization of the constituents of existing Graph-NAS frameworks.

Dimension	Method	Publication
Search space	Architecture components search space with fixed hyperparameters	Refs. [22, 44, 47, 48, 54]
	Hyperparameters search space with fixed architecture components	Ref. [58]
	Architecture components search space and hyperparameters search space	Refs. [23, 24, 46, 56, 57, 59, 60]
Search algorithm	Reinforcement learning	Refs. [22, 44, 46, 47]
	Bayesian optimization	Refs. [50, 58]
	Evolution learning	Refs. [22, 23, 48]
	Differentiable search	Refs. [54, 56, 57, 59, 60]
	Random search	Refs. [24, 44, 46, 58]
Performance evaluation	Whole model	Refs. [23, 44, 46–48, 50, 58]
	Model components	Refs. [22, 24, 54, 56, 57, 59, 60]

options of all the components of a GNN model. The larger the search space is, the more options to find the best model will be available, and the higher the computation time will be. Thus, the challenge lies in building a strong and compact search space that should contain all possible best architectures. Many works attempting to overcome the problem have proposed several types of search spaces, which we categorize in three dimensions herein: (1) a search space that comprises only GNN model ACs and uses fixed values for HPs, (2) a search space comprising only HP tuning options for a predefined GNN, and (3) a search space comprising GNN model ACs and HPs.

3.2 Search algorithm

The search algorithm defines how the best GNN model will be found among all possible GNN models in the search space. When defining the search algorithm, the computation time and computation space should be considered. The faster the algorithm is in reaching the optimal solution, the better it is; the less computing resources that the algorithm needs, the better it is. Thus, the main challenge is how to find a trade-off between high performance, low cost, and strengthened productivity through the search algorithm. Many existing search algorithms have been applied, and they include (1) RS, (2) RL, (3) evolution learning, (4) BO algorithm, and (5) differentiable search.

3.3 Performance evaluation

Performance evaluation defines how to estimate the performance of a GNN architecture generated by the search algorithm. It can be considered as an element of comparison between the various architectures generated by the search algorithm. Its role is to guide the search to the optimal solution. The challenge related to performance evaluation is to learn the submodel performance distribution to guide the progress of the search efficiently. Two common strategies are used: (1) evaluating the whole submodel or (2) evaluating its component importance.

4 Search Space

Several studies have adopted the so-called search space option to find the best combination of different components for determining the best performance metric. The general idea behind the use of a search space is undoubtedly useful, and the challenge lies in the search space design. With a search space containing nothing but average solutions, even the most efficient search algorithm can only find average solutions. The search space in the Graph-NAS literature can be classified into three broad classes: GNN model AC search space, HP search space, and the combination of the AC and HP search spaces. In the following, we analyze each class.

4.1 Search space for architecture components

Herein, a search space is designed for GNN model ACs, and fixed HPs are used[44, 47]. The search space for GNN model ACs could be a micro search space in which nodes' passing of messages in each layer is defined[46, 61] or a macro search space in which node representation in one layer might not depend on the previous layer[62]. The former is commonly adopted. Most existing studies on micro and macro search spaces support this choice by providing empirical evidence that successful performance can be achieved using the micro search space[63]. However, such search spaces are problematic, and they significantly influence search efficiency because they ignore HPs even though HPs are fully integrated model components. Thus, optimizing only the GNN structure may lead to a suboptimal model because a moderate change in learning parameters may severely degrade the fine-tuned GNN structure[37].

4.2 Search space for hyperparameters

This search space uses fixed and predefined GNN model ACs and mainly defines a search space for HPs, thus only optimizing the HPs for a predefined GNN model[58]. This topic falls under the umbrella of HPO, in which the main target is to fine-tune the HPs for a predefined neural network, which will lead to suboptimal solution.

4.3 Search space for architecture components and hyperparameters

Here, the search spaces for GNN model ACs and HPs are defined[23, 48]. The influence of a moderate change in HPs on a model is also considered. However, even if this configuration guarantees good results, it might have a supplementary computation cost because of the search space scale.

4.4 Comparative analysis

Although superior performance has been achieved using the aforementioned search spaces, most of them lack generalization because they only focus on a specific GNN design. This drawback limits the discovery of successful GNN models. Table 3 identifies the types of functions or HPs selected by existing works, and Table 4

Table 3 Details of search space.

Model	Architecture component								Hyper-parameter						
	F1	F2	F3	F4	F5	F6	F7	F8	H1	H2	H3	H4	H5	H6	H7
GraphNAS[44]	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
GraphNAS b[46]	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
SNAG[47]	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
AutoGraph[48]	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
AGNN[22]	✓	✓	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
Genetic GNN[23]	✓	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
AutoGM[50]	✓	✓	✓	✗	✓	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗
GraphGym[24]	✓	✓	✓	✗	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓
AutoNE[58]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✓
PDNAS[54]	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗
DiffMG[57]	✗	✗	✗	✗	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗
SANE[56]	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✗	✗	✗	✗
DSS[59]	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗
GNAS[60]	✗	✗	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗

Note: F1 = attention function, F2 = activation function, F3 = aggregation function, F4 = combine function, F5 = attention head, F6 = skip connection, F7 = number of layers, F8 = hidden size, H1 = learning rate, H2 = drop out, H3 = weight decay, H4 = batch normalization, H5 = batch size, H6 = optimizer, and H7 = training epochs.

Table 4 Commonly used values for functions and hyperparameters.

Component	Function	Value
Architecture component	Attention function	const, gcn, gat, sym-gat, cos, linear, gene-linear
	Activation function	sigmoid, than, relu, linear, softplus, leaky-relu, relu6, elu
	Aggregation function	sum, mean-pooling, max-pooling, mlp
	Attention head	1, 2, 4, 6, 8, 16
	Combine function	avg, concat
	Skip connection	stack, skip-sum, skip-cat
	Number of layers	1, 2, 3, 4, 5, 6
	Hidden size	2 to 64, 128, 256, 512
Hyper parameter	Learning rate	0.0001 to 0.1000
	Dropout	0.1 to 0.9
	Weight decay	0.000 01 to 0.001 00
	Batch normalization	true, false
	Batch size	16, 32, 64
	Optimizer	sgd, adam
	Training epoch	2 to 300

shows a summary of commonly used values for ACs and HPs by Graph-NAS frameworks.

We summarize the comparative analysis of the existing methods for designing search spaces in Table 5. We note that the larger the search space is, the more options to find a better model will be available, and the higher the computation time will be. This feature poses a scalability issue. Many search strategies have attempted to solve this problem. We discuss these strategies in the following sections.

5 Search Algorithm

The search algorithm describes the methodology used to find the best solution among all possible solutions from the search space. Optimization concerns the effectiveness and efficiency of the framework. To solve this optimization problem, several strategies have been proposed for different frameworks, and they include RL, BO, evolution learning, differentiable search, and RS. These search strategies can be combined. Zhou et al.[22] proposed conservative RL using reinforcement learning and evolution learning to generate slightly different GNN architectures. In this way, the controller is able to make highly efficient searches. In this section, we present the aforementioned search algorithms and discuss their strengths and weaknesses.

Table 5 Comparative analysis of search space design methods.

Method	Advantage	Disadvantage
Architecture components search space with fixed hyperparameters	Small search space and less computation	Do not consider hyper-parameters optimization, thus can lead to a sub-optimal or unstable model
Hyperparameters search space with fixed architecture components	Small search space and less computation	Only optimize hyperparameters, can lead to sub-optimal model
Architecture components search space and hyperparameters search space	Lead to stable solution	Supplementary computation cost because of the search space scale

5.1 Reinforcement learning

RL is an ML algorithm that describes a dynamic learning process in which an agent learns to achieve a goal through a sequence of decisions on the basis of continuous feedback to maximize the reward. It has three main components: The agent, which is the learner and decision-maker; the environment where the agent learns and decides what actions to perform; and the reward for each action selected by the agent.

In RL-based methods for Graph-NAS, the environment is denoted by the search space, the agent is represented by a controller defined as a neural network that aims to generate good architecture (called submodel) from the search space over time, and the reward is represented by the evaluation of the performance of the generated model. Basically, frameworks based on RL use recurrent networks as controllers to generate the descriptions of GNN models and then compute the rewards of the models as feedback to maximize the expected performance of the generated GNN models. This method has been used in many frameworks for Graph-NAS[22, 46, 47] with some differences and can be divided into two main classes. The first class consists of a complete modification and the second class consists of a partial modification. Frameworks belonging to the first class assume that the current good sub-model will be replaced in its entirety even if the next sub-model to be generated may have similarities to the old one[44, 46]. That is, at each step, a generated model is evaluated for its global performance and is completely renewed if the goal is not reached yet. The problem with this approach is that the controller cannot effectively learn which specific part of a submodel brings a change in performance relative to the previous submodel; thus, the controller is unable to efficiently guide the search. The second class includes the frameworks that change only a part of the old submodel to have a new submodel assuming that the changed part has contributed minimally to the improvement of the result obtained by the previous submodel[22, 23, 48]. In this case, the controller may consist of many neural networks, the number of which is equal to the number of model components. At each iteration, entropy is calculated for each component to assign a value to the importance of the component. Thus, this strategy can guide the search space exploration through slight architecture modifications. However, the expressive capacity of these models stills suffers from drawbacks because of their search spaces.

5.2 Bayesian optimization

BO is a probabilistic method based on the calculation of posterior probabilities by combining a prior probability with a likelihood function. The entire process is adaptive in the sense that the predictions and uncertainty estimates are updated as new observations are made. BO involves two main components: a Bayesian statistical model, also called a surrogate function, to model the objective function; and an acquisition function for deciding where to sample next. In BO, a probability distribution is built over sampled candidates tested by the surrogate function, and the acquisition function is used to interpret the response from the surrogate function and efficiently search for candidate samples before selecting them to evaluate the actual objective function. The acquisition function is optimized at each step to determine the best sample for the next evaluation. Then, the model is updated, and the process is repeated until convergence. Although BO has achieved superior performance in NAS[64], only Yoon et al.[50] have used BO to overcome the Graph-NAS problem, but their results are not promising. Specifically, BO is computationally expensive and almost impossible to conduct efficiently for large-scale data.

5.3 Evolution learning

Evolution learning is a generic population based metaheuristic optimization algorithm inspired by the mechanics of natural selection and natural genetics. This algorithm comes in various forms, among which the genetic algorithm (GA) is the most commonly used for

NAS frameworks. The most important component of the GA is the population, which is often called individuals. An individual consists of solution components or genes. The GA is an iterative process that involves evaluating individuals from an initialized population according to a fitness function and generating a new population using the best individual(s) of the previous generation. On one hand, generating a new population involves crossover and mutation operations; on the other hand, only the mutation is applied. Thus, only the best individual is selected and altered to obtain a new individual. In the Graph-NAS problem, the population is generated from the search space, and the genes are the description of a GNN architecture. However, because of the large size of the search space, determining the fitness of all individuals is almost impossible, as training one GNN is time-consuming, especially with a large graph dataset. Shi et al.[23] and Li and King[48] randomly initialize the population with a fixed size. The main problem with this method is the slow convergence.

5.4 Differentiable search

The key idea of the differentiable search method is to couple architecture sampling and training into a supernet to reduce the resource overhead. This idea implies defining a continuous search space instead of searching over a discrete set of candidates. Relative to previous search algorithms, a continuous search space helps to define a differentiable training target and leads to efficient optimization and fast convergence. Originally proposed for the NAS problem[52, 63], the differentiable search focuses on finding repetitive structures in a network called units. Modern neural networks consist of one or more computational blocks stacked together to form the final structure of the networks[65]. Although the network can have hundreds of layers, the structure of each layer is repeated. Therefore, learning one or two structures expressed as cells is sufficient to design entire networks. Different differentiable search methods vary in terms of how supernets are trained. Efficient Neural Architecture Search (ENAS)[63] trains a one-shot model on the basis of the approximate gradient obtained through REINFORCE after sampling the architecture from the search space using an RNN controller. DARTS[52] uses the joint optimization of all weights of one-shot models with a continuous relaxation of the search space, while SNAS[53] optimizes a distribution over candidate operations. In Graph-NAS frameworks based on differentiable search, a block is generally represented as a direct acyclic graph[54, 56, 57, 59, 60] consisting of an ordered sequence of nodes.

Suppose we would like to choose an operation for each layer of a five-layer network. The basic idea of the differentiable search for Graph-NAS is to create a network (supernet) by stacking mixed operations. In each mix operation, we apply all the candidates' operations in each layer and linearly combine their outputs. The mixed coefficients are designed to act as selection parameters. We can then perform an architectural search by minimizing a loss function, such as cross-entropy loss. The final architecture is then created by choosing the operation that obtains the largest coefficient in each layer. The continuous relaxation scheme relies on a parameterization trick that is less noisy and easier to train than those in previous search algorithms. Moreover, this search strategy can outperform previous search strategies in terms of search quality and speed. However, because of the construction of the supernet, the strategy results in huge computational costs and requires a differentiable approximation of latency.

5.5 Random search

RS involves generating random submodels from the search space. RS has been tested by some researchers[24, 46, 66] for Graph-NAS. Unfortunately, even if this method can provide good results in terms of efficiency, it is not widely adopted because of its hazardous results, which are often less optimal than those generated by other methods. To alleviate the hazardous nature of this approach, You et al.[24] used a controlled RS to evaluate the importance of a specific choice relative to other choices of the same component. The algorithm tests a fixed number of GNN models with the choice to be evaluated and randomly selects the other components. For each choice, the algorithm ranks the selected GNN models by their performance and analyzes the ranking distribution. The most important choice is selected for the final GNNs. Even though this method is computationally expensive, it achieves good results. Thus, investigating it further offers an interesting research direction.

5.6 Comparative analysis

Although most of the research methods cited previously have yielded good performance, each of them has drawbacks that are worth noting. Herein, we aim to

provide a comparison of the aforementioned search strategies by referring to their advantages and limitations. The comparison is shown in Table 6.

6 Performance Evaluation

The role of the evaluation strategy is to help the search algorithm optimize its choice of a submodel. It basically depends on how the new submodel is generated. On one hand, if generating a new submodel involves overwriting the old submodel[44, 46], then the evaluation will reflect the global performance of the submodel; on the other hand, if generating a new submodel simply requires changing a few components of the old model[22, 23], then the evaluation will be related to the components to determine their importance in the variance of submodel performances. Metric-based performance is considered depending on the nature of a task. For node classification, accuracy is used to evaluate a whole model, while entropy is calculated for component evaluation. Thus, the main challenge is how to speed up the submodel evaluation and improve the efficiency of the evaluation process. Several strategies have been proposed for different frameworks, and they can be applied individually and in combination. The very first strategy that quickly gained consensus is the sharing of the weights of the parameters, that is, weight sharing or parameter sharing[46, 48]. Weight sharing helps prevent a newly generated submodel from being trained from scratch to convergence, thus allowing it to benefit from the weights of previously trained models. However, this strategy can be harmful, especially when the weights are shared between two models that do not have the same dimensions. To correct this negative aspect of the strategy, many studies have opted for sharing with a constraint to avoid weight sharing between two models that do not share a certain number of similarities. Zhou et al.[22] proved through experiments that parameter sharing is not empirically useful. Another strategy is the single-path one-shot model proposed by Guo et al.[67], in which, at each iteration, only one operation between the input and output pair is activated.

Another way to speed up the submodel evaluation that has not been explored yet for Graph-NAS is to use performance prediction[68–70] instead of training all generated models to obtain the performance metrics. However, this method entails the use of hundreds of GNN performance distribution and graph data characteristics to build a neural predictor. One can also use the buffer mechanism for graph representation learning[66].

7 Application

Graph-NAS has various applications. In this section, we present the benchmark graph datasets used for Graph-NAS applications, evaluation methods, and open-source implementation.

7.1 Datasets

Six benchmark datasets, namely, the Cora dataset, CiteSeer dataset, PubMed dataset, PPI dataset, Wiki dataset, and BlogCatalog dataset, are commonly used for experiments. We divide these datasets into the following dimensions: Citation networks, biographs, and social networks. Table 7 shows the statistic of selected benchmarks dataset descriptions.

7.2 Evaluation and open-source implementations

7.2.1 Evaluation

Node classification, graph classification, and link prediction are the most widely used applications for evaluating Graph-NAS performance.

- **Node classification:** Node classification is one of the most popular and widely adopted applications of GNNs. The target is to learn the embedding state of nodes to predict the ground-truth label of unlabeled nodes. It usually takes place in semi-supervised learning, where the ground truths of some nodes are known, but those of others are unknown. Most of the studied Graph-NAS frameworks follow the standard division of *training*/*validation*/*test* for benchmark datasets. The

Table 6 Comparison of search algorithm strategies.

Search strategy	Advantage	Limitation
Reinforcement learning	Maximize performance	Time-consuming
Bayesian optimization	Use a probabilistic model	Sequential model-based optimization; computationally expensive
Evolution learning	Inherently parallel; easily distributed	Depends on the initial population; easy to fall into premature convergence; slow convergence
Differentiable search	Couple architecture sampling and training	Expensive computation cost
Random search	Not expensive in terms of computation	Hazardous result

Table 7 Benchmark datasets.

Category	Dataset	Source	#Graphs	#Classes	#Nodes (Avg)	#Edges (Avg)	#Features	#Training Nodes	#Validation Nodes	#Testing Nodes	Publication
Citation networks	Citeseer	Ref. [71]	1	6	3327	4732	3703	120	500	1000	Refs. [22–24, 44, 46–48, 50]
	Cora	Ref. [71]	1	7	2708	5429	1433	140	500	1000	Refs. [22–24, 44, 46–48, 50, 54, 56, 59]
	Pubmed	Ref. [71]	1	3	19 717	44 338	500	60	500	1000	Refs. [22–24, 44, 46–48, 50, 54, 56, 58, 59]
Social networks	BlogCatalog	Ref. [72]	1	39	10 312	333 983	–	–	–	–	Ref. [58]
	Wikipedia	Ref. [73]	1	10	11 701	216 123	300	–	–	–	Ref. [58]
Bio graphs	Proteinprotein interaction	Ref. [74]	24	121	56 944	818 716	50	–	–	–	Refs. [22–24, 44, 47, 48, 50, 54, 56, 58, 59]
	ENZYMES	Refs. [75, 76]	600	2	39.06	72.82	–	–	–	–	Ref. [24]
	PROTEINS	Refs. [75, 77]	1113	6	32.5	62.14	–	–	–	–	Ref. [24]

commonly used datasets for node classification are the Cora dataset, CiteSeer dataset, PubMed dataset, and protein-protein interaction dataset.

• **Graph classification:** The task here is to classify an entire graph into different classes. The graph classification task is similar to other classification tasks with a wide range of applications, from determining the enzymatic properties of proteins in bioinformatics to determining small communities in social networks. Biochemical graphs are usually used for graph classification experiments. You et al.[24] proposed a Graph-NAS framework with a task space comprising six graph classification tasks over multiple datasets, including the ENZYMES dataset and PROTEINS dataset.

• **Link prediction:** In the link prediction task, a GNN model needs to understand the relationship between nodes in graphs and identify pairs of nodes that will form a link or not in the future. In social networks, social interactions should be inferred, or products should be suggested to users. For link prediction, most existing Graph-NAS frameworks have been evaluated using the Wikipedia dataset and the BlogCatalog dataset.

Table 8 presents a general comparison of Graph-NAS frameworks.

7.2.2 Open source implementations

An open-source implementation of a framework is a system in which the source code of the framework is available to allow the developers of a program to modify the elements of the underlying framework to meet their own specific needs. Thus, complex new programs can be written easily without having to start from scratch. Several works cited in this article have published the source codes of their implementations. For GNNs, the most popular open source is the work of Fey et al.[78] which contains a geometric learning library in PyTorch called PyTorch Geometric. It contains several GNN models including GCN[36], GAT[38], GIN[37], GraphConv[79], LEConv[39], ARMAConv[80], and so on. Wang et al.[81] recently published the Deep Graph Library (DGL) which enables rapid implementation of many GNNs in addition to popular DL models. These two open source implementations are used by GraphNAS† which is an open source implementation of the framework described

†https://github.com/GraphNAS/GraphNAS-simple

Table 8 Summary of different Graph-NAS frameworks.

Model	Search space		Search strategy					Evaluation optimization			Task		
	ACs	HPs	RL	EL	BO	DS	RS	CI	PS	SO	NC	GC	LP
GraphNAS a[44]	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
	✓	✗	✓	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗
GraphNAS b[46]	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
	✓	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗
SNAG[47]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
	✓	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
AGNN[22]	✓	✗	✓	✓	✗	✗	✗	✓	✗	✗	✓	✗	✗
	✓	✗	✓	✓	✗	✗	✗	✓	✓	✗	✓	✗	✗
AutoGraph[48]	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗
Genetic GNN[23]	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗
AutoGM[50]	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
GraphGym[24]	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✓
AutoNE[58]	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✓
PDNAS[54]	✓	✗	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗
DiffMG[57]	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗
SANE[56]	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗
DSS[59]	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗
GNAS[60]	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✓	✗

Note: ACs = architecture components, HPs = hyperparameters, RL = reinforcement learning, EL = evolution learning, BO = bayesian optimization, RS = random search, CI = component importance, PS = parameter sharing, SO = single-path one-shot, NC = node classification, GC = graph classification, and LP = link prediction.

in Ref. [46]. Many other open source implementations for Graph-NAS exist, and they include AutoNE[‡], AutoGM[§], GraphGym[¶], and AutoGL[‖].

8 Conclusion and Future Directions

Graph-NAS has become a significant research topic because of its ability to circumvent multiple difficulties encountered when building GNN models manually. As shown in Tables 5 and 6, although Graph-NAS frameworks have proved their power in finding great GNN models, a robust trade-off between high performance, low cost, and scalability still need to be found because existing solutions fail to maintain low costs for high scalability with high performance. In summary, the current solutions for Graph-NAS outperform handcrafted GNN models but still suffer from some problems, especially in terms of adaptability, scalability, and automation. Moreover, current solutions show great room for improvement. We suggest future research directions from the aspects of efficiency, scalability, adaptability, and automation.

Efficiency. A search algorithm cannot find the best solution for a target task outside the solutions included in the search space. Thus, a good search space should consider all state-of-the-art GNN model structures. You et al.[24] proposed a general search space of GNNs with 12 functions that is a Cartesian product of different design functions and found that 7 out of the 12 design functions exert a significant influence. Given the diversity of the best GNN model architectures, how to standardize the search space with only influential functions while considering all the best model architectures could be an interesting future research direction. Meanwhile, learning the distribution of existing frameworks' performance would be a good research direction that will highlight the most influential factors of the effectiveness and efficiency of automated GNN frameworks.

Scalability. For a practical algorithm implementation, running time and performance are the most relevant factors. ML models are expected to work at the same quality observed when applied to large data. However, for most of the publications studied, the execution time increases quickly when the search space size is slightly increased, and the quality of the solution decreases as the size of the graph data increases. A study on the application of parallel computation to existing search algorithms might help reduce the linearity

[‡]https://github.com/tadpole/AutoNE
[§]https://github.com/minjiyoon/ICDM20-AutoGM
[¶]https://github.com/snap-stanford/GraphGym
[‖]https://github.com/THUMNLab/AutoGL

between the data size and computation time of GraphNAS. Meanwhile, one can use a model's performance predictor to predict a child model performance instead of conducting a full evaluation of a child model.

Adaptability. Adaptability to other problems of the same family is a desirable characteristic of ML models because it eliminates the need for ML practitioners to change or repeat patterns when the problem changes. You et al.[24] studied a task space with 32 different tasks to transfer the best models across different tasks. However, the task space combined with the search space leads to over 10 million possible combinations, thus raising the problem of how we can study search and task spaces to improve framework adaptability.

Automation. Automation is a worthwhile feature of ML models because it allows ML models to automatically scale as data size increases without the need to rethink or manually change parameters, adapt automatically to new changes in the definition of the problem, automatically generalize to unseen instances without the need to retrain the model, and automatically improve performance with experience. Thus, a joint study of efficiency scalability and adaptability can be an interesting research direction.

Acknowledgment

The work was supported by the National Natural Science Foundation of China (No. 61873288), and the CAAI-Huawei MindSpore Open Fund**.

References

[1] J. C. Cheng, Y. L. Li, J. L. Wang, L. Yu, and S. J. Wang, Exploiting effective facial patches for robust gender recognition, *Tsinghua Sci. Technol.*, vol. 24, no. 3, pp. 333–345, 2019.

[2] F. K. Gustafsson, M. Danelljan, and T. B. Schon, Evaluating scalable Bayesian deep learning methods for robust computer vision, presented at the 2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 318–319.

[3] B. Liu, S. J. Tang, X. G. Sun, Q. Y. Chen, J. X. Cao, J. Z. Luo, and S. S. Zhao, Context-aware social media user sentiment analysis, *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 528–541, 2020.

[4] Z. Q. Peng, H. Z. Song, B. H. Kang, O. B. Moctard, M. He, and X. H. Zheng, Automatic textual knowledge extraction based on paragraph constitutive relations, presented at the 6th Int. Conf. Systems and Informatics (ICSAI), Shanghai, China, 2019, pp. 527–532.

**https://www.mindspore.cn/

[5] M. Al-Ayyoub, A. Nuseir, K. Alsmearat, Y. Jararweh, and B. Gupta, Deep learning for Arabic NLP: A survey, *J. Comput. Sci.*, vol. 26, pp. 522–531, 2018.

[6] M. Farooq, F. Hussain, N. K. Baloch, F. R. Raja, H. Yu, and Y. B. Zikria, Impact of feature selection algorithm on speech emotion recognition using deep convolutional neural network, *Sensors*, vol. 20, no. 21, p. 6008, 2020.

[7] Y. Bengio, Y. LeCun, and U. de Montréal, Scaling learning algorithms towards AI, in *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds. Cambridge, MA, USA: MIT Press, 2007, pp. 1–41.

[8] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P. M. Agapow, M. Zietz, M. M. Hoffman, et al., Opportunities and obstacles for deep learning in biology and medicine, *J. Roy. Soc. Interface*, vol. 15, no. 141, p. 20170387, 2018.

[9] Y. Gurovich, Y. Hanani, O. Bar, G. Nadav, N. Fleischer, D. Gelbman, L. Basel-Salmon, P. M. Krawitz, S. B. Kamphausen, M. Zenker, et al., Identifying facial phenotypes of genetic disorders using deep learning, *Nat. Med.*, vol. 25, no. 1, pp. 60–64, 2019.

[10] W. J. Liu, G. Q. Wu, F. J. Ren, and X. Kang, DFF-ResNet: An insect pest recognition model based on residual networks, *Big Data Mining and Analytics*, vol. 3, no. 4, pp. 300–310, 2020.

[11] S. H. Wang, C. C. Liu, X. Gao, H. T. Qu, and W. Xu, Session-based fraud detection in online e-commerce transactions using recurrent neural networks, presented at the European Conf. Machine Learning and Knowledge Discovery in Databases, Skopje, Macedonia, 2017, pp. 241–252.

[12] A. S. Sheikh, R. Guigourès, E. Koriagin, Y. K. Ho, R. Shirvany, R. Vollgraf, and U. Bergmann, A deep learning system for predicting size and fit in fashion e-commerce, in *Proc. 13th ACM Conf. Recommender Systems*, Copenhagen, Denmark, 2019, pp. 110–118.

[13] Z. Yang, Y. S. Zhang, B. H. Guo, B. Y. Zhao, and Y. F. Dai, Deepcredit: Exploiting user cickstream for loan risk prediction in p2p lending, in *Proc. 12th Int. Conf. Web and Social Media, ICWSM 2018*, Stanford, CA, USA, 2018, pp. 444–453.

[14] Y. J. Wang, Y. Yao, H. H. Tong, F. Xu, and J. Lu, A brief review of network embedding, *Big Mining and Analytics*, vol. 2, no. 1, pp. 35–47, 2019.

[15] Z. Q. Liu, C. C. Chen, L. F. Li, J. Zhou, X. L. Li, L. Song, and Y. Qi, GeniePath: Graph neural networks with adaptive receptive paths, *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, pp. 4424–4431, 2019.

[16] Z. Q. Zhang, J. Y. Cai, Y. D. Zhang, and J. Wang, Learning hierarchy-aware knowledge graph embeddings for link prediction, *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 3, pp. 3065–3072, 2020.

[17] W. W. Gu, F. Gao, R. Q. Li, and J. Zhang, Learning universal network representation via link prediction by graph convolutional neural network, *Journal of Social Computing*, vol. 2, no. 1, pp. 43–51, 2021.

[18] K. H. Zhang, T. P. Li, S. W. Shen, B. Liu, J. Chen, and Q. S. Liu, Adaptive graph convolutional network with attention graph clustering for co-saliency detection, presented at the 2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 9047–9056.

[19] L. X. Xie, X. Chen, K. F. Bi, L. H. Wei, Y. H. Xu, Z. S. Chen, L. F. Wang, A. Xiao, J. L. Chang, X. P. Zhang, et al., Weight-sharing neural architecture search: A battle to shrink the optimization gap, arXiv preprint arXiv: 2008.01475, 2020.

[20] M. Nunes and G. L. Pappa, Neural architecture search in graph neural networks, presented at the 9th Brazilian Conf. Intelligent Systems, Rio Grande, Brazil, 2020, pp. 302–317.

[21] Z. W. Zhang, X. Wang, and W. W. Zhu, Automated machine learning on graphs: A survey, arXiv preprint arXiv: 2103.00742, 2021.

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

[23] M. Shi, D. A. Wilson, X. Q. Zhu, Y. Huang, Y. Zhuang, J. X. Liu, and Y. F. Tang, Evolutionary architecture search for graph neural networks, arXiv preprint arXiv: 2009.10199, 2020.

[24] J. X. You, Z. T. Ying, and J. Leskovec, Design space for graph neural networks, in *Proc. 34th Conf. Neural Information Processing Systems*, Vancouver, Canada, 2020, pp. 17009–17021.

[25] A. Sperduti and A. Starita, Supervised neural networks for the classification of structures, *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, 1997.

[26] M. Gori, G. Monfardini, and F. Scarselli, A new model for learning in graph domains, in *Proc. IEEE Int. Joint Conf. Neural Networks*, Montreal, Canada, 2005, pp. 729–734.

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, The graph neural network model, *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.

[28] Z. H. Wu, S. R. Pan, F. W. Chen, G. D. Long, C. Q. Zhang, and P. S. Yu, A comprehensive survey on graph neural networks, *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2021.

[29] Y. L. Zhang, B. Wu, Y. Liu, and J. N. Lv, Local community detection based on network motifs, *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 716–727, 2019.

[30] F. M. Bianchi, D. Grattarola, and C. Alippi, Spectral clustering with graph neural networks for graph pooling, in *Proc. 37th Int. Conf. Machine Learning*, Vienna, Austria, 2020, pp. 874–883.

[31] A. Micheli, Neural network for graphs: A contextual constructive approach, *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, 2009.

[32] H. Peng, H. F. Wang, B. W. Du, M. Z. A. Bhuiyan, H. Y. Ma, J. W. Liu, L. H. Wang, Z. Y. Yang, L. F. Du, S. Z. Wang, et al., Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting, *Inform. Sci.*, vol. 521, pp. 277–290, 2020.

[33] M. Niepert, M. Ahmed, and K. Kutzkov, Learning convolutional neural networks for graphs, in *Proc. 33rd Int. Conf. Machine Learning*, New York, NY, USA, 2016, pp. 2014–2023.

[34] B. Mohar, The laplacian spectrum of graphs, in *Graph Theory, Combinatorics, and Applications*, Y. Alavi, G. Chartrand, O. R. Oellermann, and A. J. Schwenk, eds. New York, NY, USA: John Wiley and Sons, Inc., 1991, pp. 871–898.

[35] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, Neural message passing for quantum chemistry, in *Proc. 34th Int. Conf. Machine Learning*, Sydney, Australia, 2017, pp. 1263–1272.

[36] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, presented at the 5th Int. Conf. Learning Representations, Toulon, France, 2017, pp. 1263–1272.

[37] K. Xu, W. H. Hu, J. Leskovec, and S. Jegelka, How powerful are graph neural networks? presented at the 7th Int. Conf. Learning Representations, New Orleans, LA, USA, 2019, pp. 826–842.

[38] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, Graph attention networks, presented at the 6th Int. Conf. Learning Representations, Vancouver, Canada, 2018, pp. 10903–10914.

[39] E. Ranjan, S. Sanyal, and P. Talukdar, ASAP: Adaptive structure aware pooling for learning hierarchical graph representations, *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, pp. 5470–5477, 2020.

[40] X. J. Qi, R. J. Liao, J. Y. Jia, S. Fidler, and R. Urtasun, 3D graph neural networks for RGBD semantic segmentation, presented at the 2017 IEEE Int. Conf. Computer Vision (ICCV), Venice, Italy, 2017, pp. 5209–5218.

[41] C. Huang, H. C. Xu, Y. Xu, P. Dai, L. H. Xia, M. Y. Lu, L. F. Bo, H. Xing, X. P. Lai, and Y. F. Ye, Knowledge-aware coupled graph neural network for social recommendation, *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 5, pp. 4115–4122, 2021.

[42] W. P. Song, Z. P. Xiao, Y. F. Wang, L. Charlin, M. Zhang, and J. Tang, Session-based social recommendation via dynamic graph attention networks, in *Proc. 12th ACM Int. Conf. Web Search and Data Mining*, Melbourne, Australia, 2019, pp. 555–563.

[43] J. Zhang, Y. F. Wang, Z. Y. Yuan, and Q. Jin, Personalized real-time movie recommendation system: Practical prototype and evaluation, *Tsinghua Science and Technology*, vol. 25, no. 2, pp. 180–191, 2020.

[44] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, GraphNAS: Graph neural architecture search with reinforcement learning, arXiv preprint arXiv: 1904.09981, 2019.

[45] G. C. Williams and D. C. Williams, Natural selection of individually harmful social adaptations among sibs with special reference to social insects, *Evolution*, vol. 11, no. 1, pp. 32–39, 1957.

[46] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, Graph neural architecture search, in *Proc. 29th Int. Joint Conf. Artificial Intelligence*, Yokohama, Japan, 2020, pp. 1403–1409.

[47] H. Zhao, L. N. Wei, and Q. M. Yao, Simplifying architecture search for graph neural network, arXiv preprint arXiv: 2008.11652, 2020.

[48] Y. M. Li and I. King, Autograph: Automated graph neural

network, presented at the 27th Int. Conf. Neural Information Processing, Bangkok, Thailand, 2020, pp. 189–201.

[49] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, Algorithms for hyper-parameter optimization, in *Proc. 24th Int. Conf. Neural Information Processing Systems*, Granada, Spain, 2011, pp. 2546–2554.

[50] M. Yoon, T. Gervet, B. Hooi, and C. Faloutsos, Autonomous graph mining algorithm search with best speed/accuracy trade-off, presented at the 2020 IEEE Int. Conf. Data Mining (ICDM), Sorrento, Italy, 2020, pp. 751–760.

[51] L. Page, S. Brin, R. Motwani, and T. Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*. Palo Alto, CA, USA: Stanford InfoLab, 1999.

[52] H. X. Liu, K. Simonyan, and Y. M. Yang, DARTS: Differentiable architecture search, presented at the 7th Int. Conf. Learning Representations, New Orleans, LA, USA, 2019, pp. 9055–9067.

[53] S. R. Xie, H. H. Zheng, C. X. Liu, and L. Lin, SNAS: Stochastic neural architecture search, presented at the 7th Int. Conf. Learning Representations, New Orleans, LA, USA, 2019.

[54] Y. R. Zhao, D. Wang, X. T. Gao, R. D. Mullins, P. Liò, and M. Jamnik, Probabilistic dual network architecture search on graphs, arXiv preprint arXiv: 2003.09676, 2020.

[55] E. Jang, S. X. Gu, and B. Poole, Categorical reparameterization with gumbel-softmax, presented at the 5th Int. Conf. Learning Representations, Toulon, France, 2017, pp. 1144–1156.

[56] H. Zhao, Q. M. Yao, and W. W. Tu, Search to aggregate neighborhood for graph neural network, presented at the IEEE 37th Int. Conf. Data Engineering (ICDE), Chania, Greece, 2021, pp. 552–563.

[57] Y. H. Ding, Q. M. Yao, and T. Zhang, Propagation model search for graph neural networks, arXiv preprint arXiv: 2010.03250, 2020.

[58] K. Tu, J. X. Ma, P. Cui, J. Pei, and W. W. Zhu, AutoNE: Hyperparameter optimization for massive network embedding, in *Proc. 25th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, Anchorage, AK, USA, 2019, pp. 216–225.

[59] Y. X. Li, Z. A. Wen, Y. H. Wang, and C. Xu, One-shot graph neural architecture search with dynamic search space, *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 10, pp. 8510–8517, 2021.

[60] S. F. Cai, L. Li, J. C. Deng, B. C. Zhang, Z. J. Zha, L. Su, and Q. M. Huang, Rethinking graph neural architecture search from message-passing, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition* (*CVPR*), Nashville, TN, USA, 2021, pp. 6657–6666.

[61] J. Zhou, G. Q. Cui, S. D. Hu, Z. Y. Zhang, C. Yang, Z. Y. Liu, L. F. Wang, C. C. Li, and M. S. Sun, Graph neural networks: A review of methods and applications, *AI Open*, vol. 1, pp. 57–81, 2020.

[62] K. Xu, C. T. Li, Y. L. Tian, T. Sonobe, K. I. Kawarabayashi, and S. Jegelka, Representation learning on graphs with jumping knowledge networks, in *Proc. 35th Int. Conf. Machine Learning*, Stockholm, Sweden, 2018, pp. 5453–5462.

[63] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, Efficient neural architecture search via parameters sharing, in *Proc. 35th Int. Conf. Machine Learning*, Stockholm, Sweden, 2018, pp. 4095–4104.

[64] H. P. Zhou, M. H. Yang, J. Wang, and W. Pan, BayesNAS: A Bayesian approach for neural architecture search, in *Proc. 36th Int. Conf. Machine Learning*, Long Beach, CA, USA, 2019, pp. 7603–7613.

[65] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik, ASAP: Architecture search, anneal and prune, in *Proc. 23rd Int. Conf. Artificial Intelligence and Statistics*, Palermo, Italy, 2020, pp. 493–503.

[66] K. H. Lai, D. C. Zha, K. X. Zhou, and X. Hu, Policy-GNN: Aggregation optimization for graph neural networks, in *Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, Virtual Event, CA, USA, 2020, pp. 461–471.

[67] Z. C. Guo, X. Y. Zhang, H. Y. Mu, W. Heng, Z. C. Liu, Y. C. Wei, and J. Sun, Single path one-shot neural architecture search with uniform sampling, in *Proc. 16th European Conf. Computer Vision*, Glasgow, UK, 2020, pp. 544–560.

[68] W. Wen, H. X. Liu, Y. R. Chen, H. Li, G. Bender, and P. J. Kindermans, Neural predictor for neural architecture search, in *Proc. 16th European Conf. Computer Vision*, Glasgow, UK, 2020, pp. 660–676.

[69] X. W. Zheng, R. R. Ji, Q. Wang, Q. X. Ye, Z. G. Li, Y. H. Tian, and Q. Tian, Rethinking performance estimation in neural architecture search, in *Proc. 2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition* (*CVPR*), Seattle, WA, USA, 2020, pp. 11353–11362.

[70] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, Learning curve prediction with Bayesian neural networks, in *Proc. 5th Int. Conf. Learning Representations*, Toulon, France, 2017, pp. 2001–2016.

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

[72] L. Tang and H. Liu, Relational learning via latent social dimensions, in *Proc. 15th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Paris, France, 2009, pp. 817–826.

[73] P. Mernyei and C. Cangea, Wiki-CS: A wikipedia-based benchmark for graph neural networks, arXiv preprint arXiv: 2007.02901, 2020.

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

[75] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel, Protein function prediction via graph kernels, *Bioinformatics*, vol. 21, no. Suppl 1, pp. i47–i56, 2005.

[76] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, Brenda, the enzyme database: Updates and major new developments, *Nucleic Acids Res.*, vol. 32, pp. D431–D433, 2004.

[77] P. D. Dobson and A. J. Doig, Distinguishing enzyme structures from non-enzymes without alignments, *J. Mol. Biol.*, vol. 330, no. 4, pp. 771–783, 2003.

[78] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller,

SplineCNN: Fast geometric deep learning with continuous B-spline kernels, in *Proc. 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition* (*CPVR*), Salt Lake City, UT, USA, 2018, pp. 869–877.

[79] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, pp. 4602–4609, 2019.

[80] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, Graph neural networks with convolutional ARMA filters, arXiv preprint arXiv: 1901.01343, 2021.

[81] M. J. Wang, D. Zheng, Z. H. Ye, Q. Gan, M. F. Li, X. Song, J. J. Zhou, C. Ma, L. F. Yu, Y. Gai, et al., Deep graph library: A graph-centric, highly-performant package for graph neural networks, arXiv preprint arXiv: 1909.01315, 2020.

Babatounde Moctard Oloulade received the BS degree in computer science from University of Abomey-Calavi, Cotonou, Benin, in 2012, and the MS degree in computer science and technology from Wuhan University of Technology, Wuhan, China, in 2020. He is currently a PhD student at the School of Computer Science and Engineering, Central South University, Changsha, China. His current research interests include big data processing and graph mining.

Jianliang Gao received the PhD degree in computer science from Chinese Academy of Sciences, Beijing, China, in 2011. Currently, he is a professor at the School of Computer Science and Engineering, Central South University, Changsha, China. His current research interests include big data processing and graph mining.

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

Tengfei Lyu received the BS degree from Bohai University, Jinzhou, China, in 2019. He is currently pursuing the MS degree at the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include bioinformatics, machine learning, and graph neural network.

Raeed Al-Sabri received the BS degree in information technology from Thamar University, Thamar, Yemen, in 2011, and the MS degree in computer science and technology from Nanjing University of Information Science and Technology, Nanjing, China, in 2020. He is currently a PhD candidate at the School of Computer Science and Engineering, Central South University, Changsha, China. His research interests include natural language processing, machine learning, and graph neural network.