Knowledge-Based Systems 247 (2022) 108752

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Genetic-GNN: Evolutionary architecture search for Graph Neural Networks

Min Shi [a], Yufei Tang [a,*], Xingquan Zhu [a], Yu Huang [a], David Wilson [a], Yuan Zhuang [b], Jianxun Liu [c]

[a] Electrical Eng. and Computer Science, Florida Atlantic University, USA
[b] State Key Lab of Information Eng. in Surveying, Mapping and Remote Sensing, Wuhan University, China
[c] School of Computer Science and Eng., Hunan University of Science and Technology, China

ARTICLE INFO

Article history:
Received 8 June 2021
Received in revised form 4 April 2022
Accepted 5 April 2022
Available online 11 April 2022

Keywords:
Graph Neural Networks
Neural Architecture Search
Evolutionary computation
Genetic model
Graph representation learning

ABSTRACT

Neural architecture search (NAS) has seen significant attention throughout the computational intelligence research community and has pushed forward the state-of-the-art of many neural models to address grid-like data such as texts and images. However, little work has been done on Graph Neural Network (GNN) models dedicated to unstructured network data. Given the huge number of choices and combinations of components such as aggregators and activation functions, determining the suitable GNN model for a specific problem normally necessitates tremendous expert knowledge and laborious trials. In addition, the moderate change of hyperparameters such as the learning rate and dropout rate would dramatically impact the learning capacity of a GNN model. In this paper, we propose a novel framework through the evolution of individual models in a large GNN architecture searching space. Instead of simply optimizing the model structures, an alternating evolution process is performed between GNN model structures and hyperparameters to dynamically approach the optimal fit of each other. Experiments and validations demonstrate that evolutionary NAS is capable of matching existing state-of-the-art reinforcement learning methods for both transductive and inductive graph representation learning and node classification.

1. Introduction

Network data and systems are ubiquitous [1,2] in the real world including social networks, document networks, biological networks, and many others. Relationship modeling is important for many network or graph data mining tasks (e.g., link prediction), which naturally desire flexible learning mechanisms to capture the discriminative pairwise node relationships at different levels, *i.e.*, first-order and second-order neighborhoods. Recently, Graph Neural Networks (GNN) [3,4] are developed to directly learn on networks or graphs, where nodes are allowed to incorporate high-order neighborhood relationships to generate node embeddings through the design of multiple graph convolution layers. For example, with a two-layer Graph Convolutional Networks (GCN) [4], the first and second GCN layers are able to respectively preserve the first-order and second-order neighborhood relationships between nodes in the embedding space. Due to the inspiring learning ability for graph-structured data, GNN has recently seen a plethora of successful real-world applications such as image recognition [5], new drug discovery [6], and traffic prediction [7].

As of today, many GNN structures with diverse learning mechanisms have been proposed for node relationships modeling [8,9]. For example, GCN [4] adopts a spectral-based convolution filter by which each node aggregates features from all immediate neighborhoods. Graph Attention Network (GAT) allows each node to aggregate features from all nodes in the network while learning to assign respective importance weights for different nodes. GraphSAGE [10] learns a set of aggregation functions for each node to flexibly aggregate information from neighborhoods of different hops. Yet, developing a tailored learning architecture comprising multiple GNN layers for a specific scenario (e.g., biological and physical network data) remains to be tricky, even for neural network experts, because of two main reasons. First, each of the multiple GNN layers may prefer a different aggregation function (a.k.a. aggregator) to better capture neighborhood relationships of varying orders, *i.e.*, GCN for the first-order while GAT for the second-order neighborhood relationships. Second, each specific aggregator alone may involve a number of structure selections such as activation function and the number of attention heads for GAT [10]. As a result, to identify a superior model from the huge number of combinations of various components

* Corresponding author.
E-mail address: tangy@fau.edu (Y. Tang).

https://doi.org/10.1016/j.knosys.2022.108752
0950-7051/

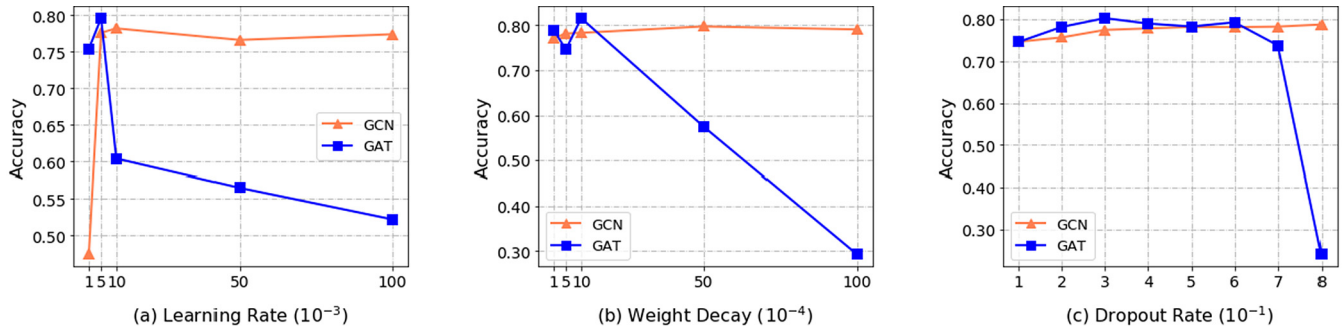


Fig. 1. Node classification accuracy of GCN and GAT models with changing hyperparameters: (a) learning rates, (b) weight decays, and (c) dropout rates, respectively. We can observe that the variation of these hyperparameters can impact or even destroy the learning ability of a GNN model such as GAT.

(e.g., aggregators and activation functions), one usually must apply tedious and laborious efforts for GNN structure selection and optimization.

To automate the model selection process, Neural Architecture Search (NAS) was widely adopted [11–13] and has been a focal point of deep learning research in recent years. NAS seeks to find an optimal combination of architecture components from a well-defined searching space and finally generates an integral model suitable for a target problem under study. To date, massive efforts have been made for optimizing Convolutional Neural Network (CNN) architectures, which pushed forward the state-of-the-art in a number of significant benchmark tasks, *e.g.*, image classification on CIFAR10/100 and ImageNet [14,15]. However, very litter work has been done on the GNN architecture search dedicated to graph-structured data. Two recent relevant works [16,17] mainly focus on the reinforcement learning-based NAS and adopt a Recurrent Neural Network (RNN) as the controller to generate variable-length component strings that describe the GNN structures. Despite the promising results, they are all challenged by the following two realities:

- **Invariable Hyperparameters.** Optimization of GNN structures is normally done based on a set of pre-defined hyperparameters. However, studies [18] show that the convergence rate and final performance of common deep learning models have significantly benefited from heuristics such as learning rate schedules. Sensitivity experiments in Fig. 1 also corroborate that a moderate change of hyperparameters (e.g., learning rate, dropout rate, and weight decay) could impact the performance of a GNN model with already optimized structures. Existing methods optimizing only the structures with fixed hyper-parameter settings are likely to produce a suboptimal GNN model, especially when the model is sensitive to the hyperparameters.
- **High Computation and Low Scalability.** Training RNN adds an additional burden to the searching process. Training the controller (e.g., RNN) and individual GNN models simultaneously demands extensive run-time computation resources. In addition, the controller typically generates candidate GNN structures and evaluates them in a sequential manner, meaning that it is difficult to scale to a large searching space and perform parallel model evaluations in the searching process.

To address the aforementioned problems, this paper proposes a novel NAS framework termed Genetic-GNN through the evolution of individual models in a large GNN architecture space/population. Different from existing methods using fixed hyperparameters, Genetic-GNN provides a mechanism to optimize the hyperparameters dynamically to conform with changing GNN structures throughout the searching process. However, jointly optimizing the GNN structure and parameter is non-trivial since

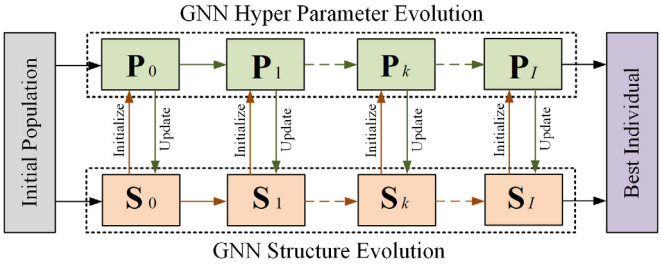


Fig. 2. The proposed Genetic-GNN model for GNN architecture search. First, the population of GNN structures is initialized ($\mathbf{S}_0$), where each individual is a multi-layer GNN with each layer constructed from randomly sampled components, *i.e.*, aggregator, activation function, and hidden embedding size. Then, the population ($\mathbf{P}_0$) of GNN parameters *w.r.t* $\mathbf{S}_0$ is initialized and evolved to identify the optimal parameter setting (e.g., learning rate and dropout rate). Subsequently, the architecture evolution (from $\mathbf{S}_0$ to $\mathbf{S}_1$) is performed to optimize the GNN structures with the best parameter setting selected from $\mathbf{P}_0$. After I rounds of alternating evolution between the structure and parameter, it finally generates a GNN architecture with optimal structure and hyperparameters settings generated from $\mathbf{S}_I$ and $\mathbf{P}_I$, respectively.

the structure and parameter are dependent on each other in that a moderate change of hyperparameters could completely deteriorate the already fine-tuned model structures and vice versa (refer to Fig. 1 as an example). In the proposed model shown in Fig. 2, we adopt an alternating evolution process to dynamically optimize both structures and parameters. In the structure evolution, each individual in the initial population represents a multi-layer GNN with randomly sampled components/parts for each layer. At each state $\mathbf{S}_k$, to determine the optimal model parameters, we hold the GNN structures and meanwhile evolve to find an optimal set of GNN hyperparameters fitting the GNN structure well. Alternatively, to find the optimal GNN structure upon the optimized hyperparameters, we hold the hyperparameters and meanwhile evolve the entire population to optimize structures.

Since both GNN structures and hyperparameters can be evolved to fit each other dynamically, we expect to achieve a GNN architecture with both optimized structure and parameter settings for the target graph learning task (e.g., node classification). Extensive experiments and comparisons demonstrate that Genetic-GNN is capable of matching the state-of-the-art methods for both transductive and inductive node representation learning and classification. An obvious advantage of Genetic-GNN is that it is able to automatically identify suitable hyperparameters settings for the searched GNN structure, which can significantly improve the automation level of the NAS process. In comparison, existing NAS methods typically rely on extensive human efforts to perform careful hyperparameters selection, which is non-trivial, especially for non-expert users. Since Genetic-GNN simply follows the principle of survival of fitness over the evolution, it is

more robust and stable to find a good GNN model compared with existing methods which adopt a more sophisticated optimization mechanism delivering poor generalization ability. In addition, unlike existing reinforcement learning-based methods, our model can be easily scaled to a large searching space since all individual models in every generation of the population are independent and thereby can be evaluated simultaneously.

In summary, we make the following contributions in this work:

1. We formulated a graph neural network architecture search problem under the evolutionary searching framework which aims to optimize both model structures and hyperparameters. The searching process follows a principle of fit of survival, which is robust to gradually optimize the GNN models with iterative evolutionary training.
2. We proposed a novel evolutionary NAS framework called Genetic-GNN to automatically identify the optimal GNN architecture from a well-defined searching space. Different from existing methods that only optimize the GNN structures, an alternating evolution process is performed to simultaneously optimize GNN structures and hyperparameters until they have a good fit for each other.
3. We designed a series of experiments to evaluate Genetic-GNN by conducting both transductive and inductive learning on graphs from three different domains. Compared with existing state-of-the-art methods, experimental results show that Genetic-GNN is able to achieve competitive performance *w.r.t* node classification accuracy, the number of trained parameters, running time, floating-point operations, and inference latency for the searched GNN models.

The rest of the paper is organized as follows. Section 2 outlines related work about GNN and NAS. Section 3 defines the GNN architecture search problem. Section 4 introduces the preliminaries regarding graph neural networks and the genetic algorithm. Section 5 establishes the underlying principle of the proposed Genetic-GNN model. Section 6 evaluates the evolutionary GNN architecture search for both transductive and inductive graph learning tasks. Section 7 presents the comparative results on the benchmark datasets against baseline models. Finally, Section 8 discusses the advantages and drawbacks of the proposed evolutionary architecture search method, and Section 9 concludes the paper while laying out possible directions for future work.

2. Related work

We first survey current research on graph neural networks and neural architecture search. Then, we summarize existing research on graph neural network architecture search and highlight the differences between our work compared with existing methods.

2.1. Graph neural networks

Many real-world systems take the form of graph or network, *i.e.*, social networks [19], citation networks [20], biology molecular networks [21] and others [22,23]. Different from the grid-like data such as texts and images that are regular and sequential, networked data are irregular in that network nodes may have different numbers of unordered neighborhoods [3], making existing neural models such as CNN and RNN cannot be directly used in the graph domain [24,25]. Recently, graph neural networks (GNN) as a family of neural network models were proposed to directly learn on graph-structured data [4,26]. The main idea of GNN is to capture node relationships and features with carefully designed graph convolution kernels or filters [25], where nodes are allowed to aggregate features from their respective neighborhood (e.g., first-order and second-order relationships) nodes iteratively.

Naturally, flexible graph convolution kernels or feature aggregators are desired to efficiently model the complex node relationships in complicated graph systems and learning tasks, *i.e.*, transductive and inductive network representation learning [10,27]. To date, a significant number of graph neural kernel designs have been proposed [9,28]. Gated graph neural network [29] adopts a gated recurrent unit for neighborhood relationship modeling, where the hidden state of each node is updated by its previous hidden states and its neighboring hidden states. Chebyshev Spectral CNN (ChebNet) adapts the traditional CNN to learn on graphs by using the Chebyshev polynomial basis to represent the spectral filters [25]. GCN [4] simplifies ChebNet architecture by using filters operating on 1-hop neighborhoods of the graph, where nodes in each GCN layer only aggregate features from their direct neighbors. Diffusion CNN (DCNN) [30] regards graph convolutions as a diffusion process. It assumes that information is transferred from one node to each of its neighborhoods following a transition probability distribution. In addition to convolution filters that treat neighborhoods equally important, many works demonstrate that attention-based filters could be beneficial. For example, Graph Attention Networks (GAN) [31] introduce an attention mechanism to determine the importance of neighborhoods to the target nodes in the feature aggregation.

Although diverse graph convolution filters and feature aggregators have been proposed to achieve new-record performance in many real-world applications (e.g., node classification and link prediction), it is impossible to identify a GNN model which is suitable for all kinds of networked data and systems [16,17]. In general, nodes build relationships with each other in different granularities, *i.e.*, direct and indirect neighborhood relations, which intuitively demand different graph filters for different feature aggregations and relationship modelings. For example, GraphSAGE [10] tries to train a set of aggregation functions that learn to aggregate features from a node's local neighborhoods, where each aggregation function is responsible for aggregating information from all neighborhoods at a particular hop.

2.2. General neural architecture search

Neural Architecture Search (NAS) is a fundamental step in automating the machine learning process, which has been successfully applied in many real-world applications such as image segmentation [32] and text processing [33]. NAS aims to design an optimal model architecture using limited computing resources in an automated way with litter or no human intervention [34]. Most existing works can be roughly classified into three categories, including reinforcement learning, Bayesian, and evolutionary optimizations [12,35].

Reinforcement Learning (RL), functioning as a model architecture selection controller, has been extensively used in automating CNN model designs [36,37]. Zoph et al. [38] first used an RNN to generate the string description of a CNN model and then trained this RNN with RL to maximize the expected accuracy (e.g., image recognition) of the generated model. Baker et al. [39] proposed an RL-based meta-modeling algorithm called MetaQNN which incorporates a novel Q-learning agent whose goal is to discover CNN architectures that perform well on a given machine learning task with no human intervention. Above methods often design and train each candidate neural model from scratch each time after searching the architecture space. To enable more efficient training, Cai et al. [40] proposed an RL-based method where weights for historical network models can be reused to evaluate the current model. However, a noticeable limitation for RL-based

NAS is the poor scalability, especially when the searching space is very large since the candidate models are sequentially dependent on the progressive optimization process and it is impossible to perform a parallel evaluation on multiple models.

Bayesian Optimization (BO) is a family of algorithms that build a probability model of the objective function determining the best expected neural network architecture [41]. Early work proposed using the tree-based frameworks such as random forest and tree Parzen estimators [42]. For example, Bergstra et al. proposed a non-standard Bayesian-based optimization algorithm TBE, which uses tree-structured Parzen estimators to model the error distribution in a non-parametric way. Hutter et al. [43] proposed SMAC which is a tree-based algorithm that uses random forests to estimate the error density. Gaussian processes are also widely used in Bayesian optimization. Kandasamy et al. [44] proposed a Gaussian Process-based BO framework for searching multi-layer perceptron and convolutional neural network architectures, which is performed sequentially and at each time step all past model evaluation results are viewed as posterior to construct an acquisition function evaluating the current model.

Evolutionary Algorithm (EA) performs an iterative genetic population-based meta-heuristic optimization process, which is a mature global optimization method with high robustness and wide applicability [12,41]. EA-based NAS has been widely used for identifying suitable CNN models for specific tasks such as image denoising, in-painting, and super-resolution [45]. Different EA-based algorithms may use different types of genome encoding methods for the neural model architectures. For example, Genetic-CNN [46] proposed to represent each network structure with a fixed-length binary string, where each element in the string corresponds to a particular kind of unit operation. Masanori et al. [47] proposed to use Cartesian genetic programming to represent the CNN structure and connectivity, which can represent variable-length network structures and skip connections.

2.3. Graph neural network architecture search

Most existing work focuses on the NAS of CNN models for grid-like data such as texts and images. For NAS of GNN models evaluating on graph-structured data, very litter work has been done so far. GraphNAS [16] proposed a graph neural architecture search method based on reinforcement learning. It first uses a recurrent network to generate variable-length strings to describe GNN architectures and then trains the recurrent network with reinforcement learning to maximize the expected accuracy of the generated architectures on a held-out validation dataset. Auto-GNN [17] follows a similar architecture searching paradigm as GraphNAS while proposing a parameter sharing strategy that enables homogeneous architectures to share learning weights. These works all focus on GNN structure (e.g., aggregators and activation functions) optimization with fixed hyperparameters. However, GNN structures and hyperparameters would impact each other in that the moderate change of hyperparameters (e.g., learning rate) could severely degrade the accuracy of the optimal GNN architecture achieved by existing methods.

Instead of focusing on reinforcement learning, we aim to evaluate the effectiveness of the evolutionary method for GNN architecture search and propose a novel framework through the evolution of individual models in a large GNN architecture searching space. In addition, unlike existing methods that only optimize the GNN structures, we propose to evolve and optimize both GNN structures and hyperparameters given that they may impact each other in the searching process. More specifically, we propose a two-phase encoding scheme that uses two strings to respectively represent the GNN structures and hyperparameters. In this way, we are able to evolve and optimize both structures and hyperparameters until they fit each other to produce an optimal model.

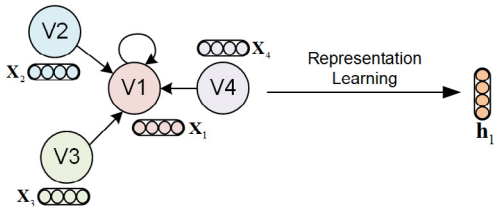


Fig. 3. The representation learning scheme of GNN models.

3. Problem definition

In this paper, the objective is to identify the optimal GNN architecture through NAS for network representation learning (a.k.a network embedding) by doing semi-supervised node classification training. Formally, the tasks of network embedding and graph NAS are defined as follows.

Definition 1 (*Network Embedding*)**.** The target network can be represented by $G = (\mathbf{V}, \mathbf{E}, \mathbf{X})$, where $\mathbf{V} = \{v_i\}_{i=1,\ldots,|\mathbf{V}|}$ is a set of $|\mathbf{V}|$ unique nodes, $\mathbf{E} = \{e_{i,j}\}_{i,j=1,\ldots,|\mathbf{V}|;\ i\neq j}$ is a set of edges that can be represented by a $|\mathbf{V}| \times |\mathbf{V}|$ adjacency matrix $\mathbf{A}$, with $\mathbf{A}_{i,j} = 1$ if $e_{i,j} \in \mathbf{E}$, or $\mathbf{A}_{i,j} = 0$ otherwise. $\mathbf{X}$ is a matrix $\mathbb{R}^{|\mathbf{V}|\times n_f}$ containing all $|\mathbf{V}|$ nodes with their associated features, *i.e.*, $\mathbf{X}_i \in \mathbb{R}^{n_f}$ is the feature vector of node v_i, where n_f is the number of unique node features. The task of network embedding is to learn a mapping $f : G \rightarrow \{\mathbf{h}_i\}_{i=1,\ldots,|\mathbf{V}|}$ by preserving network topology and node features, where $\mathbf{h}_i \in \mathbb{R}^{n_d}$ represents the low-dimensional vector representation of node v_i, and n_d is the embedding vector's dimension. f can be the GNN model identified by NAS. In this paper, the mapping is learned in a semi-supervised manner, *i.e.*, labels for a small part of nodes are known, where the node embedding vectors are trained to predict their respective labels.

Definition 2 (*Graph Neural Architecture Search*)**.** For the graph NAS task in this paper, the GNN structure space $\mathrm{S} \in \mathbb{R}^{|\mathcal{S}_1|\times|\mathcal{S}_2|\times\cdots\times|\mathcal{S}_m|}$ and GNN hyper parameter space $\mathrm{P} \in \mathbb{R}^{|\mathcal{P}_1|\times|\mathcal{P}_2|\times\cdots\times|\mathcal{P}_n|}$ have been given, where $\mathcal{S}_{i=1,2,\ldots,m} \in \mathbb{R}^{|\mathcal{S}_i|}$ is the set of candidate choices for the ith structure component (e.g., GNN aggregator) and $\mathcal{P}_{j=1,2,\ldots,n} \in \mathbb{R}^{|\mathcal{P}_j|}$ is the set of candidate choices for the jth hyper parameter. m and n are respectively the numbers of structure components and hyperparameters required to build a GNN model. The task of graph NAS is to identify the optimal choices or value specifications (e.g., $\mathcal{S}_i^{best}$ and $\mathcal{P}_j^{best}$) for each structure component $\mathcal{S}_i$ and hyper parameter $\mathcal{P}_j$, such that the resulting GNN model $f = \left\{\mathcal{S}_1^{best}, \mathcal{S}_2^{best}, \ldots, \mathcal{S}_m^{best}; \mathcal{P}_1^{best}, \mathcal{P}_2^{best}, \ldots, \mathcal{P}_n^{best}\right\}$ can achieve the optimal embedding performance for the target network G.

4. Preliminaries

To support the proposed graph NAS framework, this section briefly introduces the preliminary knowledge about graph neural networks and the genetic algorithm.

4.1. Graph neural networks

GNN is a family of neural network models that can directly incorporate graph topology and node features for efficient low-dimensional node representation learning. As shown in Fig. 3, the main idea for GNN models is that each node v_i generates the representation $\mathbf{h}_i$ by aggregating features from its neighborhoods (e.g., the first-order neighbors in this paper). Typically, the following five GNN structure components are involved in the convolutional representation learning:

1. **Attention Function** ($\mathcal{S}_1$). When each node aggregates features from its neighbors, different neighborhood nodes may have different contributions since affinities between neighborhood nodes and the target node [31] are different. The attention function aims to learn an importance weight w_{ij} for each edge relationship $e_{i,j}$ linking node v_i to node v_j. $\mathcal{S}_1$ denotes the set of candidate attention functions.
2. **Attention Head** ($\mathcal{S}_2$). Instead of applying single-head attention, studies [48] show that a model may have better robustness if incorporating multiple parallel attention learnings simultaneously. Multi-head attention allows the model to jointly attend to features from different node representation subspaces. The multiple representation outputs by multi-head attention for each node v_i are then concatenated or averaged to generate the final representation $\mathbf{h}_i$. $\mathcal{S}_2$ denotes the set of candidate attention head numbers.
3. **Aggregation Function** ($\mathcal{S}_3$). Each node v_i may have multiple neighborhood nodes, thus an aggregation function (e.g., averaging operation) is required to combine features from multiple neighbors and form the final representation $\mathbf{h}_i$. $\mathcal{S}_3$ denotes the set of candidate aggregation functions.
4. **Activation Function** ($\mathcal{S}_4$). After deriving the representation $\mathbf{h}_i$ for node v_i, a non-linear activation function (e.g., ReLu and Sigmoid) is usually applied to smooth $\mathbf{h}_i$ or transform $\mathbf{h}_i$ as a probability distribution vector for multi-class node classification. $\mathcal{S}_4$ denotes the set of candidate activation functions.
5. **Hidden Unit** ($\mathcal{S}_5$). The hidden unit controls the dimension of the representation $\mathbf{h}_i$ for node v_i. For a multi-layer GNN model, the output dimensions in the hidden or middle convolution layers may impact the representation ability of the entire GNN model. $\mathcal{S}_5$ denotes the set of candidate dimension choices.

Based on above definitions, the first-layer learning structure of a GNN model (e.g., instantiated and indexed by 1) can be represented as an action string by $\mathrm{S}_1 = \left\{\mathcal{S}_1^1, \mathcal{S}_2^1, \mathcal{S}_3^1, \mathcal{S}_4^1, \mathcal{S}_5^1\right\}$. Formally, the representation $\mathbf{h}_i \in \mathbb{R}^{\mathcal{S}_5^1}$ for node v_i can be computed as:

$$\mathbf{h}_i = \Big\Vert_{l=1}^{\mathcal{S}_2^1} \mathcal{S}_4^1 \left(\mathcal{S}_3^1 \left(\mathcal{S}_1^1(\mathbf{X}_i, \mathbf{X}_j, \mathbf{W}^l)\mathbf{W}^l\mathbf{X}_j \right)_{j\in\mathcal{N}_i} \right) \tag{1}$$

where $\Vert$ represents concatenation, $\mathcal{N}_i$ represents the set of direct (e.g., first-order) neighborhoods of node v_i, and $\mathbf{W}^l \in \mathbb{R}^{n_f}$ is the learnable weight matrix for the lth attention head. One can stack multiple GNN layers to form a multi-layer GNN model. At the kth layer, suppose the action string be instantiated as $\mathrm{S}_k = \left\{\mathcal{S}_1^k, \mathcal{S}_2^k, \mathcal{S}_3^k, \mathcal{S}_4^k, \mathcal{S}_5^k\right\}$, then the output representation $\mathbf{h}_i^{(k)} \in \mathbb{R}^{\mathcal{S}_5^k}$ is written as:

$$\mathbf{h}_i^{(k)} = \Big\Vert_{l=1}^{\mathcal{S}_2^k} \mathcal{S}_4^k \left(\mathcal{S}_3^k \left(\mathcal{S}_1^k(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{W}^l)\mathbf{W}^l\mathbf{h}_j^{(k-1)} \right)_{j\in\mathcal{N}_i} \right) \tag{2}$$

where $\mathbf{h}_i^{(k-1)}$ is the output representation by the $(k-1)$th GNN layer and $\mathbf{W}^l \in \mathbb{R}^{\mathcal{S}_5^{(k-1)}}$ represents the corresponding learnable weight matrix. If k indicates the last GNN layer, the aggregation function $\mathcal{S}_3^k$ will be the *averaging* operation, meaning averaging the representations generated by all $\mathcal{S}_2^k$ attention heads. Then, the final representation for node v_i is written as:

$$\mathbf{h}_i^{(k)} = \mathcal{S}_4^k \left(\frac{1}{\mathcal{S}_2^k} \sum_{j\in\mathcal{N}_i} \left(\mathcal{S}_1^k(\mathbf{h}_i^{(k-1)}, \mathbf{h}_j^{(k-1)}, \mathbf{W}^l)\mathbf{W}^l\mathbf{h}_j^{(k-1)} \right) \right) \tag{3}$$

In this paper, the weight matrix $\mathbf{W}_l$ at each GNN layer is trained in a semi-supervised or supervised manner, *i.e.*, by performing end-to-end node classification [4,31] optimized with the gradient decent algorithm.

4.2. Genetic Algorithm

Genetic Algorithm (GA) belongs to the family of evolutionary algorithms motivated by the principle of natural selection and genetics [49]. The searching space is a major ingredient for all GAs, which is encoded in the form of strings known as *chromosomes* or *individuals*, and a collection of such strings constitutes a *population*. A *chromosome* is composed of a sequence of elements called *genes* which encode the solution of a target problem. Initially, a random population is created representing different individual solutions for the target problem. A *fitness* value is associated with each individual to indicate its goodness within the population. GA optimizes the population and tries to find the global optimal solution through a standard evolution procedure as follows:

1. **Initialize**(population)
2. **Evaluate**(population)
3. **While**(stopping condition not satisfied):
   (a) **Select**(population)
   (b) **Crossover**(population)
   (c) **Mutate**(population)
   (d) **Evaluate**(population)
   (e) **Update**(population)
4. **Return** the best individual in the population

where the evaluation step aims to calculate the fitness of each individual, the selection step aims to choose some individuals from the entire population as parents for mating, the crossover step describes how parental individuals switch the information (e.g., swap the genes) and produce the next generations (e.g., new individuals), the mutation step aims to introduce diversity in the population by randomly altering a gene from new individuals conditioned on the mutation probability, and finally, it updates the population by adding new individuals.

5. The proposed method

This section presents a Genetic Graph Neural Network (Genetic-GNN) NAS framework to evolve the GNN architectures. As shown in Fig. 4, Genetic-GNN can be organized into three main components that aim to optimize both GNN structures and hyperparameters, including GNN architecture representation & population initialization, GNN hyperparameters evolution, and GNN structure evolution. We detail the three components in the following sections.

5.1. Architecture representation & population initialization

As discussed in previous sections, the optimization of GNN structures and hyperparameters are dependent on each other. Existing works that optimize only GNN structures may end up with a suboptimal searched model since the change of hyperparameters could severely degrade the fine-tuned GNN structure. Therefore, we advocate evolving both GNN structures and hyperparameters for reliable NAS. In addition, optimizing the hyperparameters can further increase the automation level of NAS compared to existing methods that only optimize the GNN structures using fixed hyperparameters.

In this paper, we are specifically interested in optimizing the following three types of hyperparameters, though Genetic-GNN is a generic framework that is flexible to include other significant parameters:

1. **Dropout Rate** ($\mathcal{P}_1$). Over-fitting is a common issue while training neural network models. Dropout is a technique for addressing this problem, which meanwhile helps to reduce

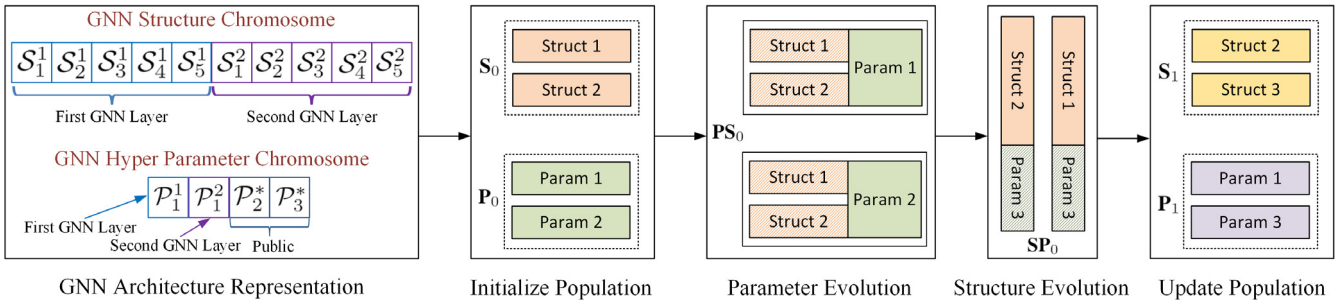


Fig. 4. The proposed Genetic-GNN model for GNN architecture optimization. For simplicity, we demonstrate one evolution step of a two-layer GNN, where both structure and parameter populations have two individuals. **First**, the GNN structure population ($\mathbf{S}_0$) and parameter population ($\mathbf{P}_0$) are randomly initialized, where each structure or parameter individual is represented with a respective string/chromosome. **Second**, an intermediate population $\mathbf{PS}_0$ is constructed to evolve parameters with structures fixed, *i.e.*, assume a better parameter individual *Param 3* is produced to replace *Param 2*. **Then**, another intermediate population $\mathbf{SP}_0$ is constructed to evolve structures with parameters fixed, *i.e.*, assume a better structure individual *Struct 3* is produced to replace *Struct 1*. **Finally**, both structure and parameter populations are updated.

Table 1
The search space for structure components.

Component	Search space
$\mathcal{S}_1$	listed in Table 2
$\mathcal{S}_2$	1, 24, 6, 8, 16
$\mathcal{S}_3$	"sum", "mean-pooling", "max-pooling", "mlp"
$\mathcal{S}_4$	"sigmoid", "tanh", "relu", "linear", "softplus", "leaky_relu", "relu6", "elu"
$\mathcal{S}_5$	4, 8, 16, 32, 64, 128, 256

the training complexity for large networks [50]. The key idea is to randomly drop units (along with their connections) from the neural network during training. $\mathcal{P}_1$ denotes the set of candidate dropout rates.

2. **Weight Decay Rate** ($\mathcal{P}_2^*$). Similar to the dropout, weight decay (e.g., L_2 norm regularization) is a widely used technique to decrease the complexity and meanwhile increase the generalization ability of neural network models by limiting the growth of model weights. $\mathcal{P}_2^*$ denotes the set of candidate weight decay rates.
3. **Learning Rate** ($\mathcal{P}_3^*$). Learning rate determines how fast the loss changes every time while training a neural model based on the gradient descent algorithm. A larger learning rate could cause the model to converge too quickly to a suboptimal solution, whereas a smaller learning rate could cause the optimization to converge too slowly. $\mathcal{P}_3^*$ denotes the set of candidate learning rates.

The weight decay is usually applied to GNN model weights at the first layer and the learning rate is set for training the entire GNN model. Therefore, the hyperparameters $\mathcal{P}_2^*$ and $\mathcal{P}_3^*$ are set once and shared throughout multiple layers of a GNN model.

Assume we search the optimal architecture for a two-layer GNN model, as shown in Fig. 4, the GNN structure can be represented by a string/chromosome:

$$\left\{\mathcal{S}_1^1, \mathcal{S}_2^1, \mathcal{S}_3^1, \mathcal{S}_4^1, \mathcal{S}_5^1, \mathcal{S}_1^2, \mathcal{S}_2^2, \mathcal{S}_3^2, \mathcal{S}_4^2, \mathcal{S}_5^2\right\} \tag{4}$$

where the first and second GNN layers are indexed by 1 and 2, respectively. Similarly, the GNN hyperparameters can be represented by a string/chromosome:

$$\left\{\mathcal{P}_1^1, \mathcal{P}_1^2, \mathcal{P}_2^*, \mathcal{P}_3^*\right\} \tag{5}$$

where shared parameters in the two GNN layers are indexed by the notation *. While evaluating the network embedding performance (e.g., fitness of individuals), the structure and parameter strings need to be combined to form the entire GNN architecture (e.g., the mapping function f):

$$f = \left\{\mathcal{S}_1^1, \mathcal{S}_2^1, \mathcal{S}_3^1, \mathcal{S}_4^1, \mathcal{S}_5^1, \mathcal{S}_1^2, \mathcal{S}_2^2, \mathcal{S}_3^2, \mathcal{S}_4^2, \mathcal{S}_5^2; \mathcal{P}_1^1, \mathcal{P}_1^2, \mathcal{P}_2^*, \mathcal{P}_3^*\right\} \tag{6}$$

Table 2
The search space of structure component $\mathcal{S}_1$.

Search space	Definition
const	$w_{i,j} = 1$
gcn	$w_{i,j} = \frac{1}{\sqrt{\mathcal{N}_i \mathcal{N}_j}}$
gat	$w_{i,j} = leaky\_relu(\mathbf{W}^l * \mathbf{h}_i + \mathbf{W}^l * \mathbf{h}_j)$
sym-gat	$w_{i,j} = w_{i,j} + w_{j,i}$ based on gat
cos	$w_{i,j} = cos(\mathbf{W}^l * \mathbf{h}_i, \mathbf{W}^l * \mathbf{h}_j)$
linear	$w_{i,j} = tanh(sum(\mathbf{W}^l * \mathbf{h}_j))$
gene-linear	$w_{i,j} = \mathbf{W}^a * tanh(\mathbf{W}^l * \mathbf{h}_i + \mathbf{W}^l * \mathbf{h}_j)$, where $\mathbf{W}^a$ is a trainable weight matrix

Table 3
The search space for hyperparameters.

Parameter	Search space
$\mathcal{P}_1$	0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
$\mathcal{P}_2^*$	5e−4, 8e−4, 1e−3, 4e−3
$\mathcal{P}_3^*$	5e−4, 1e−3, 5e−3, 1e−2

For a given graph G, two datasets are created, including a training node set D_{train} and a held-out validation node set D_{val}. The candidate GNN model f is trained on D_{train} by minimizing the semi-supervised node classification loss:

$$\mathcal{L}_f = -\sum_{v_i \in D_{train}} \mathbf{Y}_i \ln \mathbf{h}_i \tag{7}$$

where $\mathbf{Y}_i$ is the one-hot label indicator vector for node v_i. Then, the classification accuracy of f is computed on D_{val}.

Following the literature [16], the candidate choices or searching space for each GNN structure component are summarized in Tables 1 and 2. Similarly, we define the searching space for each hyper parameter summarized in Table 3. Based on the chromosomes (e.g., Eqs. (4) and (5)) and their respective searching spaces (e.g., Tables 1 and 3), the GNN structure population $\mathbf{S}_0 = \{struct_i\}_{i=1,\dots,N_s}$ and hyper parameter population $\mathbf{P}_0 = \{param_j\}_{j=1,\dots,N_p}$ are respectively created and initialized, *i.e.*, feeding each structure component in Eq. (4) and hyper parameter in Eq. (5) with values randomly selected from their respective searching spaces, where N_s and N_p are the population sizes (e.g., number of individuals). We use $f_{i,j} = \{struct_i; param_j\}$ to represent a candidate GNN model and its classification accuracy is $Acc(f_{i,j})$, where $struct_i \in \mathbf{S}_0$ and $param_j \in \mathbf{P}_0$. In the following sections, we adopt an alternating evolution procedure to evolve $\mathbf{S}_0$ and $\mathbf{P}_0$, aiming to identify the optimal f for learning a target graph.

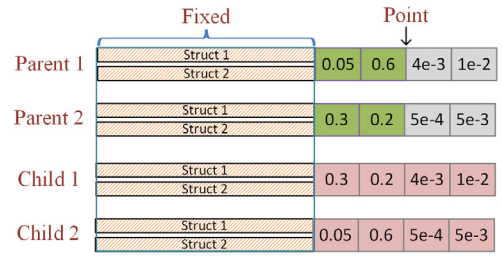


Fig. 5. An example to show the crossover process between two parents, where the point index for learning parameter crossover is 2.

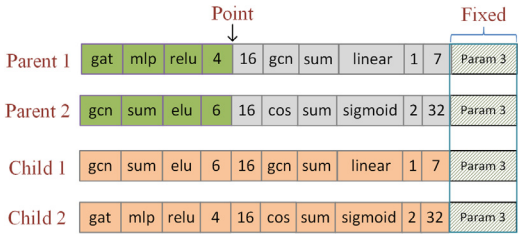


Fig. 6. An example to show the crossover process between two parents, where the point index for structure crossover is 4.

Algorithm 1 Training procedure of the Genetic-GNN model

Require: The target graph G, train set D_{train} and validation set D_{val}
Ensure: The optimal GNN architecture and the learned embedding vector $\mathbf{h}_{v_i}$ for each node $v_i \in \mathbf{V}$

Initialize the structure population $\mathbf{S}_0$
Initialize the hyper parameter population $\mathbf{P}_0$
procedure ArchitectEvolving(D_{train}, D_{val}, $\mathbf{S}_0$, $\mathbf{P}_0$, K_s, K_p)
  for $i \leftarrow 1$ to K_s **do**
    Construct intermediate GNN model population $\mathbf{PS}_i$
    for $j \leftarrow 1$ to K_p **do**
      Select($\mathbf{PS}_i$)
      Crossover($\mathbf{PS}_i$)
      Mutate($\mathbf{PS}_i$)
      Evaluate($\mathbf{PS}_i$)
      Update($\mathbf{PS}_i$)
    end for
    Construct intermediate GNN model population $\mathbf{SP}_i$
    Select($\mathbf{SP}_i$)
    Crossover($\mathbf{SP}_i$)
    Mutate($\mathbf{SP}_i$)
    Evaluate($\mathbf{SP}_i$)
    Update($\mathbf{SP}_i$)
  end for
end procedure

5.2. GNN hyper parameter evolution

The parameter evolution component aims to evolve $\mathbf{P}_k$ to optimize and identify the optimal parameter setting for the corresponding GNN structure population $\mathbf{S}_k$ before evolving to the next structure population $\mathbf{S}_{k+1}$, where the goodness/fitness of a particular parameter setting need to be measured regarding all individuals in the structure population. To this end, as shown in Fig. 4 (e.g., $k = 0$), we combine $\mathbf{S}_0$ and $\mathbf{P}_0$ to create an intermediate population $\mathbf{PS}_0 = \{PS_j\}_{j=1,\dots,N_p}$, where each individual $PS_j = \{f_{i,j}\}_{i=1,\dots,N_s}$ packages a set of GNN models with the common hyper parameter setting $param_j \in \mathbf{P}_0$, and different individuals have the same GNN structure settings in $\mathbf{S}_0$. The fitness of PS_j is computed as:

$$fitness(PS_j) = \alpha Acc(f_{b,j}) + (1-\alpha)\frac{1}{N_s}\sum_{i=1}^{N_s} Acc(f_{i,j}) \tag{8}$$

where $f_{b,j} = \{struct_b; param_j\} \leftarrow \text{argmax}_{f_{i,j} \in PS_j} Acc(f_{i,j})$ is the best individual with highest classification accuracy $Acc(f_{b,j})$ in the population. The last term of Eq. (8) calculates the average classification accuracy over all individuals. The idea is that we seek to find a parameter setting $param_j$ that fits the entire structure population $\mathbf{S}_0$ well while simultaneously considering its fit to the best structure individual $struct_b \in \mathbf{S}_0$. α is a balance parameter to adjust the importance between the global and local structures for flexible fitness calculation.

Then, based on the standard GA algorithm, we evolve $\mathbf{PS}_0$ to optimize the learning parameter $\mathbf{P}_0$ by holding the GNN structure $\mathbf{S}_0$, which includes the selection, crossover, mutation, evaluation, and updating steps. For example, Fig. 5 shows the crossover step between two selected parents, where the crossover only happens to the hyperparameters with fixed GNN structures. The parameter evolution (e.g., $\mathbf{P}_0$ evolves to $\mathbf{P}_1$) finally identifies and outputs the parameter individual with highest fitness calculated by Eq. (8), *i.e.*, the $param_3$ shown in Fig. 4.

5.3. GNN structure evolution

Once the optimal hyper parameter $param_j \in \mathbf{P}_{k+1}$ for $\mathbf{S}_k$ has been identified, the structure evolution component proceeds to evolve $\mathbf{S}_k$ to identify the optimal GNN structure. For this purpose, as the case (e.g., $k = 0$) shown in Fig. 4, an intermediate population $\mathbf{SP}_0 = \{SP_i\}_{i=1,\dots,N_s}$ is created by concatenating the optimal parameter individual with each structure individual $struct_j \in \mathbf{P}_0$, where $SP_i = f_{i,j}$ indicates an individual GNN model with its fitness calculated as:

$$fitness(SP_i) = Acc(f_{i,j}) \tag{9}$$

Similarly, we evolve $\mathbf{SP}_0$ to optimize the GNN structure $\mathbf{S}_0$ (e.g., $\mathbf{S}_0$ evolves to $\mathbf{S}_1$) by holding the hyperparameter $\mathbf{P}_0$ based on the standard GA algorithm. For example, Fig. 6 shows the crossover process by altering only the structural parts of the two parents while keeping the hyperparameters fixed.

5.4. Algorithm explanation

The hyperparameters evolution and GNN structure evolution continue in an alternating manner until an optimal model is generated. Before evolving the structure population $\mathbf{S}_k$, the parameter population $\mathbf{P}_k$ is evolved to identify the optimal hyperparameters fitting $\mathbf{S}_k$ and meanwhile $\mathbf{P}_k$ evolves to $\mathbf{P}_{k+1}$. Subsequently, the structure population $\mathbf{S}_k$ is evolved to optimize the GNN structures and meanwhile $\mathbf{S}_k$ evolves to $\mathbf{S}_{k+1}$. The above alternating process is performed iteratively to finally achieve a multi-layer GNN architecture with both optimal structures and hyperparameters. The training procedure of Genetic-GNN is summarized in Algorithm 1, where K_s and K_p are numbers of generations for structure evolution and parameter evolution, respectively. In the evolution of intermediate populations $\mathbf{PS}_k$ and $\mathbf{SP}_k$, since all individual GNN models are independent and can be evaluated simultaneously, Genetic-GNN is able to scale to a large searching space without increasing the evaluation time exponentially like the RL-based NAS methods, i.e., RL-based NAS method can only evaluate one candidate model at a time.

6. Experiment

6.1. Evaluation task and dataset

Following literature [16,17], we test the performance of Genetic-GNN for both transductive and inductive graph learning

Table 4
Statistics information of various networks used in the experiments.

Domains	Document networks			Image networks		Biology network
Items	Cora	Citeseer	PubMed	MIR	ImageCLEF	PPI
# Nodes	2708	3327	19,717	5892	3461	56,944
# Edges	5429	4732	44,338	380,808	221,185	818,716
# Features	1433	3703	500	500 × 375	500 × 375	50
# Classes	7	6	3	152	134	121
# Training nodes	140	120	60	500	500	44,906 (20 graphs)
# Validation nodes	500	500	500	1000	1000	6514 (2 graphs)
# Testing nodes	1000	1000	1000	2000	2000	5524 (2 graphs)

and node classification on graphs from three different domains shown in Table 4.

Transductive Learning. The transductive learning is performed on the same graph, where a subset of nodes with labels are used for training and other nodes are used for validation and test. Two different classification tasks are examined for transductive learning:

- **Document classification:** Three benchmark citation networks including Cora, Citeseer, and Pubmed [4] are used for transductive node representation learning. **Cora** contains 2708 research papers grouped into 7 machine learning classes such as *Reinforce Learning* and *Genetic Algorithms*. There are 5429 edges between them and each paper node is described with a feature vector of 1433 dimensions. **Citeseer** contains 3327 research papers in 6 classes with 4732 links between them, where each paper node has a feature vector of 3703 dimensions. **Pubmed** contains 19,717 literature nodes and 44,338 edges. Each node belongs to one of the 3 classes and has a feature vector of 500 dimensions. For these three datasets, 20 nodes per class are used for training (e.g., the train set D_{train}), 500 nodes are used for validation (e.g., the validation D_{val}), and 1000 nodes are used for testing the model performance.
- **Image classification:** We adopt two multi-class image networks MIR and ImageCLEF for performing the image classification. MIR contains 5892 nodes from 152 classes. Each node is a 500 × 375 RGB color image and there are 380,808 edges among images for this network. ImageCLEF contains 3461 nodes and 221,185 edges. Each node is also a 500 × 375 RGB image that corresponds to one or more classes out of a total of 134 classes. For each image in these two datasets, we extract a CNN feature descriptor and the feature dimensions for MIR and ImageCLEF are transformed to 152 and 134, respectively. For the MIR and ImageCLEF datasets, we use 500 labeled image nodes for training the model. For other unlabeled nodes, we select 1000 and 2000 images for validation and test respectively.

Inductive Learning. For inductive learning, the graphs for training and testing are different. Therefore, different from transductive learning, inductive learning involves the embedding learning of multiple different subgraphs. We use a protein–protein interaction (PPI) dataset [31] which contains 24 subgraphs for inductive node representation learning. **PPI** consists of graphs corresponding to different human tissues, which has a total number of 56,944 nodes and 818,716 edges. Each graph has 2372 nodes on average, and each node has 50 features including positional gene sets, motif gene sets, and immunological signatures. Each node corresponds to multiple classes (labels) from a total of 121 classes. For this dataset, we use 20 graphs for training, 2 graphs for validation, and 2 graphs for testing, where both validation and test graphs have no connections with graphs in the training set.

6.2. Baseline

We compare Genetic-GNN with the following state-of-the-art methods which adopt either handcrafted GNN architectures or automatically searched GNN architectures.

Methods based on Handcrafted GNN Architecture:

- **Chebyshev** [25] adapts the traditional CNN to learn on graphs by using the Chebyshev polynomial basis to represent the spectral CNN's filters.
- **GCN** [4] is a two-layer GCN architecture, where each node generates representation by adopting a spectral-based convolutional filter to recursively aggregate information from all its direct neighbors.
- **GraphSAGE** [10] is a general inductive framework that leverages node features to generate node embeddings for previously unseen data. It learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood.
- **GAT** [31] is a method built on the GCN model. It introduces an attention mechanism at the node level, which allows each node to specify different weights to different nodes in a neighborhood.
- **LGCN** [51] selects a fixed number of neighboring nodes for each feature based on value ranking in order to transform graph data into grid-like structures. Then, the traditional CNN model is directly applied to learn from the transformed graph.

Methods based on GNN Architecture Search

- **GraphNAS** [16] first uses a recurrent network to generate variable-length strings that describe the architectures of graph neural networks, and then trains the recurrent network with reinforcement learning to maximize the embedding accuracy of the generated architectures.
- **Auto-GNN**[17] is a reinforcement learning-based method similar to GraphNAS, which adopts a parameter sharing strategy that enables homogeneous architectures to share parameters during the training.
- **SNAG** [52] is a simplified version of GraphNAS for GNN architecture search. It extends from GraphNAS by designing a novel search space and adopts a reinforcement learning-based search algorithm with a weight sharing mechanism.

Following the literature [16], Chebyshev and GCN are for transductive learning since they require the whole graph structure and nodes to be available during the training. GraphSAGE is used for inductive learning which is able to predict embeddings of unseen graphs based on the trained model. Other baselines including GAT, LGCN, GraphNAS, Auto-GNN, and SANG are used for both transductive and inductive embedding learning. In the experiment, all baseline methods adopt a semi-supervised training, meaning only a handful of labeled nodes are used for training the GNN model, while others are used for validation and testing.

Table 5
The transductive node classification results on citation networks. The first and second best results are boldfaced and underlined respectively.

Categories	Methods	Layers	Accuracy				
			Cora	Citeseer	Pubmed	MIR	ImageCLEF
Handcrafted	Chebyshev	2	81.2%	69.8%	74.4%	54.3 ± 0.3%	58.8 ± 0.5%
	GCN	2	81.5%	70.3%	79.0%	56.1 ± 0.4%	61.0 ± 0.5%
	GAT	2	83.0 ± 0.7%	72.5 ± 0.7%	79.0 ± 0.3%	57.3 ± 0.8%	61.5 ± 0.3%
	LGCN	2	83.3 ± 0.5%	73.0 ± 0.6%	79.5 ± 0.2%	57.8 ± 0.4%	61.9 ± 0.5%
GNN NAS	GraphNAS	2	**84.2 ± 1.0%**	73.1 ± 0.9%	79.6 ± 0.4%	58.1 ± 0.3%	**62.4 ± 0.6%**
	Auto-GNN	2	83.6 ± 0.3%	**73.8 ± 0.7%**	**79.7 ± 0.4%**	58.0 ± 0.5%	62.1 ± 0.4%
	SNAG	3	83.5 ± 0.5%	72.6 ± 0.5%	79.3 ± 0.4%	57.9 ± 0.6%	62.0 ± 0.7%
	Genetic-GNN	2	83.8 ± 0.5%	73.5 ± 0.8%	79.2 ± 0.6%	**58.7 ± 0.4%**	62.3 ± 0.5%

6.3. Experimental setting

We set the number (N_s) of initial structure individuals between 10 and 50, the number (K_s) of evolving generations of structure population between 10 and 50, and the balance parameter α in Eq. (8) between 0.2 and 1.0. For comparison, the default hyperparameters for Genetic-GNN are set as follows. We set the number of initial structure individuals N_s as 20, the number of initial parameter individuals N_p as 6, the number of structure evolving generations K_s as 50, the number of parameters evolving generations K_p as 10, the balance ratio α as 0.6, the numbers of parents for structure and parameter genetics as 10 and 4, the numbers of children for structure and parameter genetics as 4 and 2, the mutation probability for both structure and parameter evolution as 0.02. For transductive learning, we aim to identify the optimal architecture of a two-layer GNN model within the searching space, while for inductive learning a three-layer GNN model is optimized in this paper. We also evaluate the performance of Genetic-GNN for searching GNN architectures with deeper hidden convolution layers ranging from 2 to 6. We train 200 epochs for each specific GNN model, where the accuracy and Micro-F1 are used as evaluation metrics for the transductive and inductive embedding learning tasks, respectively. We compare the transductive and inductive embedding learning performance of various baseline methods on the respective datasets (refer to Section 6.1). In the experiments to examine the computational cost and test sensitivities of various parameters such as the number of hidden convolution layers, we adopt the Cora and Citeseer since they are relatively smaller and easier to perform the parameter permutation than other datasets. The detailed implementation and code can be available by the GitHub link: https://github.com/codeshareabc/Genetic-GNN.

7. Results

This section demonstrates the node classification performance of both transductive and inductive graph embedding learning. Then, some important parameters are empirically examined through their impacts on the Cora and Citeseer datasets, respectively.

7.1. Graph NAS-based embedding learning performance

Table 5 shows the document classification results on Cora, Citeseer, and Pubmed, as well as the image classification results on MIR and ImageCLEF. From the comparative results, we conclude three main observations as follows:

- The NAS-based methods including GraphNAS, Auto-GNN and Genetic-GNN can achieve better results than handcraft-based GNN models on all five datasets, which verified the effectiveness of NAS to identify good GNN models for the given graph-structure data. This is because the handcrafted models are usually determined by several manual trails, *i.e.*, tuning the number of GNN layers and hyperparameters, which has a very low chance to obtain an optimal model. In comparison, the NAS-based methods are able to evaluate a large number of candidate models from a well-defined searching space automatically. For example, we can observe from Table 1 that there are 7840 ($7 \times 5 \times 4 \times 8 \times 7$) possible GNN models from different combinations of structural components in the searching space. The problem becomes even more complicated after considering the hyperparameters (Table 3), which results in a total of 878,080 candidate models. However, we can use NAS to efficiently find a well-performed GNN model for the target graph data and learning task. It is helpful for non-expert users who have litter knowledge about the application domains and the neural network models.
- In the category of NAS-based methods for document classification on Cora, Citeseer, and PubMed, the performance of our model Genetic-GNN is able to match those of the reinforcement learning-based methods GraphNAS and Auto-GNN. A t-student significant test is performed between them with the p value equals to 0.05. It shows that Genetic-GNN is not significantly different from other state-of-the-art methods such as GraphNAS, which verified the effectiveness of Genetic-GNN for GNN architecture search to achieve state-of-the-art results. However, there are two major advantages for Genetic-GNN compared to other NAS methods including GraphNAS, Auto-GNN, and SANG. First, Genetic-GNN follows a principle of survival of fitness to optimize the GNN architecture, which is robust to generate a good model as the evolutionary process continues. In comparison, other NAS methods normally need to manage an extra controller such as the RNN model to iteratively optimize the GNN architecture. Training the RNN-based controller will cause extra computation resources and suffer from the overfitting problem in the semi-supervised training [52]. For example, in addition to training and evaluating the GNN models, the number of trained parameters for the RNN controller alone in GraphNAS reaches up to 87.2K, and needs an extra time of about 20 s per GNN architecture to optimize the RNN controller through the reinforcement learning feedback. The second advantage is that, unlike GraphNAS and Auto-GNN which still need users to manually select the hyperparameters while optimizing the GNN structures, Genetic-GNN is able to optimize both GNN structures and hyperparameters until they reach a good fit for each other, which has greatly increased the automation level of GNN NAS (as we demonstrate in Fig. 1 that optimizing model parameters is also important to contribute to an optimal GNN model).
- In addition, NAS-based methods have shown promising results in the multi-label node classification on image networks, *i.e.*, each image may have multiple labels. We can

Table 6
The inductive node classification results on the PPI network. The first and second best results are boldfaced and underlined respectively.

Categories	Methods	# Layers	PPI Micro-F1
Handcrafted	GraphSAGE (RNN)	2	61.2%
	GAT	3	$97.3 \pm 0.2\%$
	LGCN	–	$77.2 \pm 0.2\%$
GNN NAS	GraphNAS	3	$98.6 \pm 0.1\%$
	Auto-GNN	3	$\mathbf{99.2 \pm 0.1}\%$
	SNAG	3	$\underline{98.8 \pm 0.1\%}$
	Genetic-GNN	3	$98.6 \pm 0.4\%$

further observe that Genetic-GNN can achieve competing and even better performance than existing state-of-the-art NAS methods. For example, Genetic-GNN has an improvement of 0.6% over GraphNAS, 0.7% over Auto-GNN and 0.8% over SNAG, respectively. This verified the effectiveness of Genetic-GNN for graph neural network architecture search applied in the multi-label node classification task. Since Genetic-GNN directly evaluates the candidate GNN models and retains the improved models at every generation over the evolutionary process, it can always end up with finding a well-performed GNN model. Therefore, Genetic-GNN tends to be more robust to find an optimal GNN model for various graph learning tasks, compared to other NAS methods relying on the RNN controller to optimize the GNN models which is sometimes unstable, *i.e.,* returning a locally optimal model or an over-fitting model.

We also compare various baseline methods for the inductive graph embedding learning on the PPI data and Table 6 shows the inductive node classification results. Similar conclusions can be drawn in transductive learning on the document and image networks. First, automated methods perform generally better than handcrafted methods. Second, the performance of Genetic-GNN is as good as the state-of-the-art model GraphNAS. The reasons why Genetic-GNN does not show an obvious performance advantage are in two aspects. First, existing methods such as GraphNSA and Auto-GNN applied a manually sophisticated parameter fine-tuning process, meaning the hyperparameters adopted by these methods can well support the searched GNN structures to perform well. Therefore, although our method can automatically identify suitable hyperparameters, it does not mean it must show much superiority using the automatically identified hyperparameters. The advantage is that GNN-Genetics has its unique mechanism, as an alternative to existing methods, to find good hyperparameter settings with limited human efforts. Second, genetics-based algorithms favor a larger search space, a larger population, and a larger number of evolving generations in order to discover a near-optimal solution. In this paper, due to the limitation of computation resources, we only set the maximum population size as 50 and the evolution generations as 50. However, we argue that with the increase of population size and evolution generations, our model has the potential to further achieve improved performance which is guaranteed by its adopted principle of survival of fitness (e.g., the best models survive).

7.2. Computational cost of the GNN architecture

It is important to consider the efficiency of the searched GNN architectures to be applied in practical scenarios [37], especially for some resource and response — constraint deployments and applications, *i.e.*, mobile devices and Web-based services. Therefore, we design a series of experiments to compare the relative computational cost of the GNN models identified by various methods. We measure the computational cost with several critical indicators including the number of trainable parameters, running time per epoch, floating-point operations (FLOPs), and inference latency. All these computational costs are tested using a machine with an Intel Core i7 and 16 GB CPU.

Similar to literature [16], we collect the top 5 GNN architectures that achieve the best validation accuracy and calculate their average computational cost. Table 7 compares different methods regarding the number of trainable parameters, running time per epoch, FLOPs, and latency. We can observe for all the NAS methods including GraphNAS, Auto-GNN, SNAG, and Genetic-GNN, our model Genetic-GNN needs generally less running time per epoch, fewer FLOPs, and shorter inference latency compared with other NAS methods, which demonstrates that the Genetic-GNN can find a computationally efficient GNN model. Moreover, Genetic-GNN does not rely on training an additional controller during the GNN architecture searching process, whereas SAGN, GraphNAS, and Auto-GNN adopt RNN as the controller to generate the GNN structures which will bring additional computational costs such as the number of training parameters and running time. For example, beyond evaluating the candidate GNN models, for GraphNAS the number of trained parameters of the RNN controller alone reaches up to 87.2K, which needs an extra 20 s for each GNN model to optimize the RNN controller with reinforcement learning. We can also observe that on the Cora and Citeseer networks Genetic-GNN generates more parameter-optimized GNN models than SANG and GraphNAS.

In addition, we also compare the performance and computational cost (*w.r.t* the number of trainable parameters and the running time per epoch) between Genetic-GNN and GraphNAS for searching GNN architectures with deeper hidden convolution layers. Since deeper GNN models often suffer from the over-fitting problem in the semi-supervised training, we adopt the supervised training instead following a previous work [53], where the training, validation, and testing splits are 60%, 20% and 20% respectively. The comparison results on Cora and Citeseer with the convolution layers ranging from 2 and 6 are summarized in Table 8. We can conclude that Genetic-GNN is able to find GNN models with similar or even better performance as the GraphNAS. The number of parameters and training time for the searched GNN models are generally smaller for Genetic-GNN compared to GraphNAS, which again verified the effectiveness of the evolutionary architecture search for GNNs.

7.3. Parameter influence

We empirically demonstrate the impacts of some important parameters used in Genetic-GNN on Cora and Citeseer data. α is a balance parameter used in the fitness calculation in Eq. (8) and its impact is shown in Fig. 7(a). We can observe that the best setting for both data is 0.6. Fig. 7(b) shows the influence of the number of structure population size N_s. We can observe that with the increase of N_s, the performance first goes up and then decreases on both citation networks. Generally, a larger population size means a larger search space which tends to generate a better solution. However, the large search space normally requires more evolution generations to finally identify an optimal model, which probably explains that the performance decreases as the population size increases in Fig. 7(b). Fig. 7(c) shows the influence of evolution generation on the structure population, where the performance gradually increases over the evolution generations.

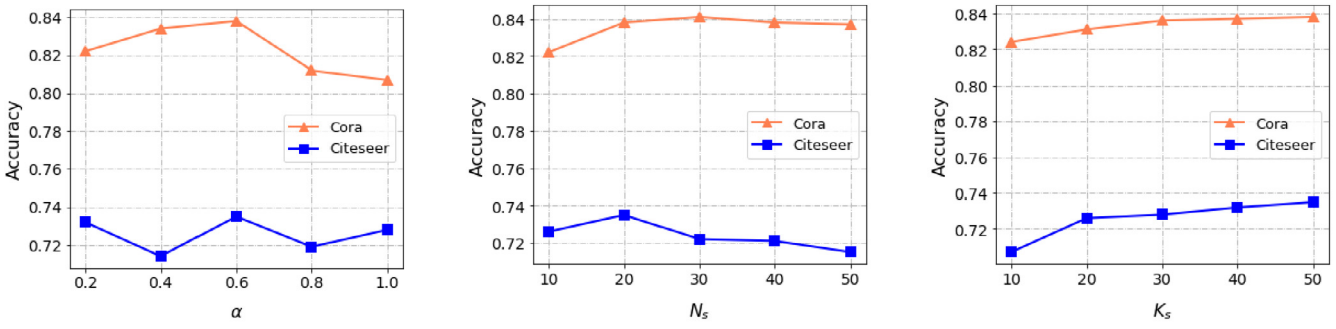
Fig. 8(a) and (b) present the validation and test performances change with the training iterations on Cora and Citeseer, respectively. We can observe the performances have a tendency to improve with the training, though they have some turbulence. The reason for the unstable curves is that both validation and

Table 7
The computational cost on Cora and Citeseer networks. (Time: seconds per epoch)

Methods	Layers	Cora				Citeseer			
		Params	Time	FLOPs	Latency	Params	Time	FLOPs	Latency
Chebyshev	2	0.09M	11.2 s	0.48M	0.34S	0.09M	11.4 s	1.04M	0.28S
GC N	2	0.02M	0.8 s	0.21M	0.15S	0.05M	0.7 s	0.36M	0.16S
GAT	2	0.09M	1.8 s	258M	0.51S	0.23M	2.7 s	525M	0.73S
LGCN	2	0.05M	1.2 s	195M	0.24S	0.05M	1.7 s	376M	0.25S
GraphNAS	2	0.09M	0.9 s	185M	0.22S	0.23M	0.8 s	269M	0.26S
Auto-GNN	2	0.05M	–	–	–	0.71M	–	–	–
SANG	2	0.09M	0.9 s	171M	0.21S	0.25M	0.9 s	254M	0.23S
Genetic-GNN	2	0.07M	0.6 s	163M	0.13S	0.18M	0.8 s	173M	0.18S

Table 8
Comparison of NAS-based GNN models *w.r.t* various GNN hidden layers (train/validation/test data splits are 60%, 20% and 20% respectively).

Methods	# GNN Layers	Cora			Citeseer		
		Accuracy	Params	Time	Accuracy	Params	Time
GraphNAS	2	$89.6 \pm 0.8\%$	91.8K	1.2 s	$77.8 \pm 0.7\%$	234.3K	0.8 s
	3	$88.4 \pm 0.6\%$	94.6K	1.7 s	$77.6 \pm 0.6\%$	238.3K	1.3 s
	4	$88.4 \pm 0.4\%$	97.5K	2.0 s	$75.1 \pm 0.6\%$	242.2K	1.8 s
	5	$87.9 \pm 0.5\%$	101.2K	2.4 s	$75.2 \pm 0.5\%$	246.3K	2.2 s
	6	$87.5 \pm 0.4\%$	105.1K	3.0 s	$74.4 \pm 0.8\%$	250.3K	2.5 s
Genetic-GNN	2	$89.5 \pm 0.7\%$	74.0K	0.8 s	$78.2 \pm 0.5\%$	182.4K	0.8 s
	3	$89.1 \pm 0.5\%$	77.9K	1.3 s	$77.8 \pm 0.8\%$	186.5K	1.2 s
	4	$88.5 \pm 0.6\%$	81.7K	1.6 s	$76.4 \pm 0.6\%$	190.5K	1.7 s
	5	$88.2 \pm 0.4\%$	85.5K	1.9 s	$75.8 \pm 0.5\%$	193.1K	1.9 s
	6	$87.7 \pm 0.6\%$	89.4K	2.4 s	$74.9 \pm 0.7\%$	198.9K	2.1 s



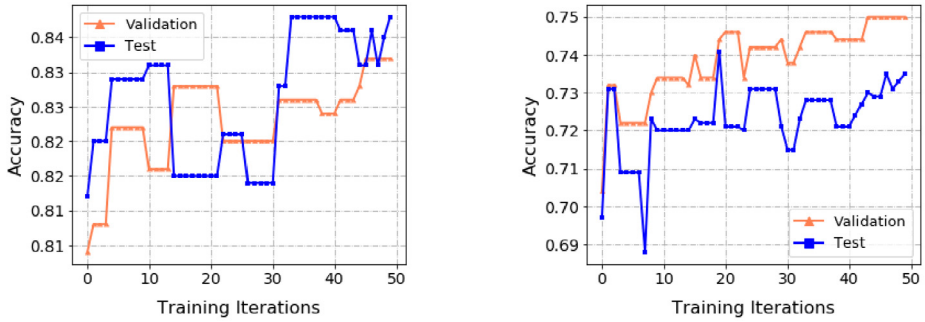
(a) Influence of the parameter α (b) Influence of parameter N_s (c) Influence of the parameter K_s

Fig. 7. (a) Influence of the imbalance parameter α. (b) Influence of the number of initial structure population size K_s. (c) Influence of the number of evolving generations K_s for structure population.

test sets are unseen, and the best performance on the train set cannot guarantee the best performance on the validation and test sets. In addition, we can observe that although the validation and test accuracies are not consistent locally, they both have a trend to improve overall with more training iterations. This is because the validation and test node sets are different, making their evaluation performances have some degree of discrepancy when applied to the GNN model trained on the training set. However, the difference is minor and they both have an increasing tendency, which means the GNN model is optimized with the iterative training.

8. Discussion

Nowadays, graph neural networks have become the de facto solution for learning feature representations of graph-structured data. However, it is challenging, even for expert users, to identify a GNN model with optimal architectures in the real-scenario graph learning task due to the various choices of feature aggregators and hyperparameters. Neural architecture search is a promising technique for automatically identifying a well-performed GNN model and has attracted increasing attention in the community [54,55]. In this paper, the proposed evolutionary architecture search for graph neural networks has four obvious advantages: (1) It follows a principle of survival of fitness to optimize the individual GNN models, making it an easily understandable and conceptually interpretable architecture search method for the general users; (2) Due to the principle of natural selection, the evolutionary architecture search method has a potential to return a globally optimal GNN model; (3) Genetic-GNN is more robust and stable to find a good GNN model mainly because it is less likely to encounter the overfitting problem in the semi-supervised training, compared to many reinforcement learning-based methods which need to train an additional burdensome controller; (4) Both the GNN structures and hyperparameters can be optimized, which has increased the automation level of NAS compared to existing methods which only optimize the structures. From the comparisons of Genetic-GNN with existing methods regarding both the graph representation learning performance and the computational cost, we observe that Genetic-GNN shows superiority to some extent, especially in the computational costs. One major advantage for Genetic-GNN is to automatically identify suitable hyperparameters for a searched GNN model.



(a) Training in Cora Network (b) Training in Citeseer Network

Fig. 8. (a) Validation and test performance on Cora change over the training iterations. (b) Validation and test performance on Citeseer change over the training iterations.

Although the automatically identified hyperparameters are not necessarily better than those identified from the manually fine-tuning process, it is indeed desirable especially for non-expert users to avoid the tedious hyper-parameter fine-tuning process.

Although the above merits make Genetic-GNN a promising approach, Genetic-GNN favors large population size and the number of generations in order to find a near-optimal GNN model. This would cause heavy overhead for evaluating tremendous and even duplicate candidate GNN models. In this work, we adopt a relatively small size of population and generation, which is the main reason why the performance of Genetic-GNN is not better than some state-of-the-art NAS methods. Nevertheless, in most cases, one can still quickly find a satisfactory GNN model by managing a relatively small population and evolving a limited number of generations. Such an advantageous point cannot be seen in the reinforcement learning-based NAS methods since they often need to train the entire architecture search process with the gradient descent algorithm and end until it is converged.

9. Conclusion

In this paper, we aim to demonstrate the effectiveness of evolutionary neural architecture search for optimizing graph neural network models on graph-structured data. We proposed a novel genetic-based approach called Genetic-GNN for automatically identifying the optimal GNN models with a well-defined search space. Instead of only optimizing the GNN structures with fixed hyperparameters, Genetic-GNN is able to evolve and optimize both structure and parameter to fit each other. The experimental results and parameter sensitivity tests demonstrated that our model is able to match the state-of-the-art reinforcement learning-based methods.

Since the evolutionary algorithms tend to achieve better solutions with larger population size and evolution generations, it is a future work to test on larger search space and evolution generations. In addition, parameter sharing between individual models is also an interesting direction, *i.e.*, when a parameter individual evolves to another parameter individual, their model structures remain the same, thereby the model weight parameters can be shared between the two models.

CRediT authorship contribution statement

Min Shi: Methodology, Data curation, Writing – original draft. **Yufei Tang:** Conceptualization, Methodology, Supervision, Funding acquisition. **Xingquan Zhu:** Methodology, Writing – review & editing, Funding acquisition. **Yu Huang:** Data curation, Visualization. **David Wilson:** Investigation, Validation, Writing – original draft. **Yuan Zhuang:** Writing – review & editing. **Jianxun Liu:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the National Science Foundation through Grant Nos. IIS-1763452, CMMI-2145571, and OAC-2017597.

References

[1] D. Zhang, J. Yin, X. Zhu, C. Zhang, Network representation learning: A survey, IEEE Trans. Big Data (2018).
[2] P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding, IEEE Trans. Knowl. Data Eng. 31 (5) (2018) 833–852.
[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[4] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations (ICLR), 2017.
[5] Z.-M. Chen, X.-S. Wei, P. Wang, Y. Guo, Multi-label image recognition with graph convolutional networks, in: Proc. of IEEE CVPR, 2019, pp. 5177–5186.
[6] M. Sun, S. Zhao, C. Gilvary, O. Elemento, J. Zhou, F. Wang, Graph convolutional networks for computational drug development and discovery, Brief Bioinform. (2019).
[7] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, H. Li, T-gcn: A temporal graph convolutional network for traffic prediction, IEEE Trans. Intell. Transp. Syst. (2019).
[8] Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: A survey, IEEE Trans. Knowl. Data Eng. (2020).
[9] N.M. Kriege, F.D. Johansson, C. Morris, A survey on graph kernels, Appl. Netw. Sci. 5 (1) (2020) 1–42.
[10] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in Neural Information Processing Systems, 2017, pp. 1024–1034.
[11] X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, Knowl.-Based Syst. 212 (2021) 106622.
[12] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, 2018, arXiv preprint arXiv:1808.05377.
[13] Y. Ding, Q. Yao, T. Zhang, Propagation model search for graph neural networks, 2020, arXiv preprint arXiv:2010.03250.
[14] X. Dong, Y. Yang, One-shot neural architecture search via self-evaluated template network, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 3681–3690.
[15] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, X. Wang, Renas: Reinforced evolutionary neural architecture search, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4787–4796.
[16] Y. Gao, H. Yang, P. Zhang, C. Zhou, Y. Hu, GraphNAS: Graph neural architecture search with reinforcement learning, 2019, arXiv:1904.09981.
[17] K. Zhou, Q. Song, X. Huang, X. Hu, Auto-GNN: Neural architecture search of graph neural networks, 2019, arXiv:1909.03184.

[18] A. Gotmare, N.S. Keskar, C. Xiong, R. Socher, A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation, in: International Conference on Learning Representations, 2018.
[19] S. Min, Z. Gao, J. Peng, L. Wang, K. Qin, B. Fang, STGSN—A Spatial–Temporal Graph Neural Network framework for time-evolving social networks, Knowl.-Based Syst. 214 (2021) 106746.
[20] S. Molaei, H. Zare, H. Veisi, Deep learning approach on information diffusion in heterogeneous networks, Knowl.-Based Syst. 189 (2020) 105153.
[21] C. Su, J. Tong, Y. Zhu, P. Cui, F. Wang, Network embedding in biomedical data science, Brief. Bioinform. 21 (1) (2020) 182–197.
[22] C. Chen, K. Li, W. Wei, J.T. Zhou, Z. Zeng, Hierarchical graph neural networks for few-shot learning, IEEE Trans. Circuits Syst. Video Technol. (2021).
[23] C. Chen, K. Li, S.G. Teo, X. Zou, K. Wang, J. Wang, Z. Zeng, Gated residual recurrent graph neural networks for traffic prediction, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 485–492.
[24] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Trans. Neural Netw. 20 (3) (2009) 498–511.
[25] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.
[26] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016, pp. 2014–2023.
[27] X. Zhang, W. Chen, H. Yan, TLINE: Scalable transductive network embedding, in: Asia Information Retrieval Symposium, Springer, 2016, pp. 98–110.
[28] Z. Yang, S. Dong, HAGERec: hierarchical attention graph convolutional network incorporating knowledge graph for explainable recommendation, Knowl.-Based Syst. 204 (2020) 106194.
[29] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, 2015, arXiv preprint arXiv:1511.05493.
[30] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1993–2001.
[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, 2017, arXiv preprint arXiv:1710.10903.
[32] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A.L. Yuille, L. Fei-Fei, Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 82–92.
[33] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, L. Zhou, TextNAS: A neural architecture search space tailored for text representation, in: AAAI, 2020, pp. 9242–9249.
[34] S. Li, W. Li, S. Wen, K. Shi, Y. Yang, P. Zhou, T. Huang, Auto-FERNet: A facial expression recognition network with architecture search, IEEE Trans. Netw. Sci. Eng. (2021).
[35] M. Wistuba, A. Rawat, T. Pedapati, A survey on neural architecture search, 2019, arXiv preprint arXiv:1905.01392.
[36] Y. Jaafra, J.L. Laurent, A. Deruyver, M.S. Naceur, Reinforcement learning for neural architecture search: A review, Image Vis. Comput. 89 (2019) 57–66.
[37] B. Lyu, S. Wen, K. Shi, T. Huang, Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing, IEEE Trans. Cybern. (2021).
[38] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.
[39] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, 2016, arXiv preprint arXiv:1611.02167.
[40] H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[41] X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, 2019, arXiv preprint arXiv:1908.00709.
[42] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.
[43] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507–523.
[44] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, E.P. Xing, Neural architecture search with bayesian optimisation and optimal transport, in: Advances in Neural Information Processing Systems, 2018, pp. 2016–2025.
[45] K. Ho, A. Gilbert, H. Jin, J. Collomosse, Neural architecture search for deep image prior, 2020, arXiv preprint arXiv:2001.04776.
[46] L. Xie, A. Yuille, Genetic cnn, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1379–1388.
[47] M. Suganuma, S. Shirakawa, T. Nagao, A genetic programming approach to designing convolutional neural network architectures, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 497–504.
[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[49] R.H. Sheikh, M.M. Raghuwanshi, A.N. Jaiswal, Genetic algorithm based clustering: a survey, in: 2008 First International Conference on Emerging Trends in Engineering and Technology, IEEE, 2008, pp. 314–319.
[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[51] H. Gao, Z. Wang, S. Ji, Large-scale learnable graph convolutional networks, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1416–1424.
[52] H. Zhao, L. Wei, Q. Yao, Simplifying architecture search for graph neural network, 2020, arXiv preprint arXiv:2008.11652.
[53] Y. Zhao, D. Wang, X. Gao, R. Mullins, P. Lio, M. Jamnik, Probabilistic dual network architecture search on graphs, 2020, arXiv preprint arXiv:2003.09676.
[54] Y. Li, I. King, AutoGraph: Automated graph neural network, in: International Conference on Neural Information Processing, Springer, 2020, pp. 189–201.
[55] M. Nunes, G.L. Pappa, Neural architecture search in graph neural networks, in: Brazilian Conference on Intelligent Systems, Springer, 2020, pp. 302–317.