Evolutionary Architecture Search for Graph Neural Networks

Min Shi, David A. Wilson, *Student Member, IEEE*, Xingquan Zhu, *Senior Member, IEEE*, Yu Huang, Yuan Zhuang, *Member, IEEE*, Jianxun Liu, and Yufei Tang*, *Member, IEEE*

***Abstract*—Automated machine learning (AutoML) has seen a resurgence in interest with the boom of deep learning over the past decade. In particular, Neural Architecture Search (NAS) has seen significant attention throughout the AutoML research community, and has pushed forward the state-of-the-art in a number of neural models to address grid-like data such as texts and images. However, very litter work has been done about Graph Neural Networks (GNN) learning on unstructured network data. Given the huge number of choices and combinations of components such as aggregator and activation function, determining the suitable GNN structure for a specific problem normally necessitates tremendous expert knowledge and laborious trails. In addition, the slight variation of hyper parameters such as learning rate and dropout rate could dramatically hurt the learning capacity of GNN. In this paper, we propose a novel AutoML framework through the evolution of individual models in a large GNN architecture space involving both neural structures and learning parameters. Instead of optimizing only the model structures with fixed parameter settings as existing work, an alternating evolution process is performed between GNN structures and learning parameters to dynamically find the best fit of each other. To the best of our knowledge, this is the first work to introduce and evaluate evolutionary architecture search for GNN models. Experiments and validations demonstrate that evolutionary NAS is capable of matching existing state-of-the-art reinforcement learning approaches for both the semi-supervised transductive and inductive node representation learning and classification.**

***Index Terms*—Graph Neural Networks, Architecture Search, Evolutionary Computation, Genetic Model**

I. Introduction

NETWORK data and systems are ubiquitous [1], [2] in the real world including social network, document network, and biological network, *etc*. Relationship modeling is of paramount importance 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 in the embedding space. Due to the encouraging learning ability for graph-structured data in many domains, GNN has recently seen a plethora of successful real-world applications such as image recognition [5], new drug discovery [6], and traffic prediction [7], *ect*.

As of today, many GNN structures with diverse learning mechanisms have been proposed for node relationship modeling [8], [9]. For examples, GCN [4] adopts a spectral-based convolution filter by which each node aggregates features from all direct neighborhoods. Graph Attention Network (GAT) makes each node aggregate features from all nodes on 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 consisting of multiple GNN layers for a specific scenario (e.g., biological and physical network data) remains to be tricky, even for the 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 in the respective order, *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 (e.g., aggregators and activation functions), one usually must apply tedious and laborious efforts for GNN structure tuning and optimization.

To automate the model selection process, Neural Architecture Search (NAS) is widespread used [11], [12] and has been a focal point of deep learning research in recent years. It seeks to find an optimal combination of architecture components from a well-defined searching space, where the resulting assembled

This work was supported in part by the U.S. National Science Foundation through Grant Nos. IIS-1763452, CNS-1828181, IIS-2027339, and OAC-2017597, and an Early-Career Research Fellowship from the Gulf Research Program (GRP) of the National Academies of Sciences, Engineering, and Medicine (NASEM).

M. Shi is with the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, USA, and the School of Computer Science and Engineering, Hunan University of Science and Technology, Hunan, China. E-mail: toshimin132@gmail.com.

D. Wilson, Y. Huang, X. Zhu, and Y. Tang are with the Department of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA. E-mails: {davidwilson2016, yhwang2018@fau.edu, xzhu3, tangy}@fau.edu.

Yuan Zhuang is with the State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 129 Luoyu Road, Wuhan 430079, China. E-mail: yuan.zhuang@whu.edu.cn.

Jianxun Liu is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Hunan, China. E-mail: ljx529@gmail.com.

* Corresponding author.

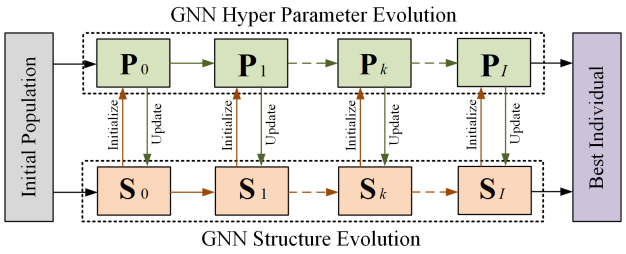

Fig. 1. The proposed Genetic-GNN model for GNN architecture search. First, the population of GNN structures is initialized ($\mathbf{S}_0$), where each individual is a multi-layer GNN with each layer constructed from randomly sampled components, *i.e.*, aggregator, activation function, and hidden embedding 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 hyper parameter settings generated from $\mathbf{S}_I$ and $\mathbf{P}_I$, respectively.

neural model is suitable for a target problem. To date, massive efforts has been applied to searching the 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 [13], [14]. In contrast, very litter work has been done about GNN learning on graph-structured data. Two recent relevant works [15], [16] mainly focus on the reinforcement learning-based NAS and adopt a Recurrent Neural Network (RNN) as the controller to generate variable-length strings that describe the GNN structures. Despite the promising results, existing works are consistently challenged by the following two shortcomings:

- **Invariable Hyper Parameters.** In addition to structures of GNN, a slight variation of hyper parameters (e.g., learning rate) could drastically impact the performance of a model with the converged structure. Existing methods optimizing only the structural variables with fixed hyper-parameter settings may end up with a suboptimal model.
- **High Computation and Low Scalability.** Training the recurrent network adds burden to the searching process. Both the training of controller and individual GNN model would demand extensive run-time computation resource. Furthermore, the controller typically generates candidate GNN structures and evaluates them in a sequential manner, which is difficult to scale to a large searching space.

To address 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 considering both model structures and learning parameters. However, jointly optimizing GNN structure and parameter is non-trivial, since the structure and parameter are dependent on each other in that a moderate change of hyper parameters could completely deteriorate the already fine-tuned model structures and vice verse. In the proposed model shown in Fig. 1, 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 population structures and meanwhile evolve to find the optimal parameters fitting the entire population well. Then, to find the optimal GNN structure upon a parameter setting, we hold the parameters and meanwhile evolve the entire population to optimize structures.

Since both GNN structure and parameter can be evolved to fit each other dynamically, we expect to achieve a GNN architecture with both optimal structure and parameter settings for the target graph learning task (e.g., node classification). To the best of our knowledge, this is the first work to introduce and evaluate evolutionary architecture search for GNN models on graph-structured data. Extensive experiments and comparisons demonstrate that Genetic-GNN is capable of matching the state-of-the-art for both transductive and inductive node representation learning and classification. In addition, compared with existing reinforcement learning-based methods, our model can be easily scaled to large searching space since all individual models in each population are independent and thereby can be evaluated simultaneously.

In summary, the contribution of this work is as follows:

1) We formulated a graph neural network architecture search problem under the evolutionary searching framework that seeks to optimize both model structures and hyper parameters.
2) We proposed a novel evolutionary NAS framework called Genetic-GNN to automatically identify the optimal GNN architecture from a well-defined searching space. An alternating evolution process is performed to dynamically optimize GNN structures and parameters to fit each other.
3) We designed extensive experiments to demonstrate the effectiveness of evolutionary framework for GNN architecture search on both transductive and inductive graph representation learning and node classification. The results can provide guidance for other practitioners to select suitable graph neural models for a specific scenario.

The rest of the paper is organized as follows. Section II outlines related work about GNN and NAS. Section III defines the GNN architecture search problem. Section V establishes the underlying principles of the proposed Genetic-GNN model. Section VI evaluates the evolutionary GNN architecture search on both transductive and inductive graph learning tasks, and presents the comparative results on the benchmark datasets against baseline models. Finally, Section VIII concludes the paper while laying out possible directions of future work.

II. Related Work

This section first surveys current research on graph neural networks and the general neural architecture search, and then summarizes existing research targeted at the graph neural network architecture search and highlight their differences with our work in this paper.

A. Graph Neural Networks

Many real-world systems take the form of graph or network, *i.e.*, social networks, citation networks, and biology molecular networks [17]. Different from 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], causing the failure of existing neural models such as CNN and RNN in the graph domain [18], [19]. Recently, graph neural networks (GNN) as a family of neural network models were proposed to directly learn on graph-structured data [4], [20]. The main idea of GNN is to capture node relationships and features with carefully designed graph convolution kernels or filters [19], 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 various graph systems and learning tasks, *i.e.*, transductive and inductive network representation learning [10], [21]. To date, a significant number of graph neural kernel designs have been proposed [9]. Gated graph neural network [22] 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 [19]. 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) [23] regards graph convolutions as a diffusion process. It assumes information is transferred from one node to each of its neighborhoods following a transition probability distribution. In addition to convolution filters that treat neighborhood nodes as equally important, many works demonstrate that attention-based filters could be useful. For example, Graph Attention Networks (GAN) [24] introduce an attention mechanism to determine the importance of neighborhoods to the center node during 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 prohibitive to identify a GNN model which is suitable for all kinds of networked data and systems [15], [16]. In general, nodes build relationships with each other in different granularity, *i.e.*, direct and indirect neighborhood relations, which intuitively demands varying graph filters for different feature aggregations and relationship modeling. For example, GraphSAGE [10] tries to train a set of aggregator functions that learn to aggregate feature information from a nodes local neighborhood, where each aggregator function aggregates information from a different number of hops, or search depth, away from a given node.

B. 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 [25] and text processing [26]. NAS aims to design a model architecture with the best performance using limited computing resources in an automated way with litter or no human intervention. Most of existing works can be roughly classified into three categories, including reinforcement learning, Bayesian, and evolutionary optimizations [12], [27].

Reinforcement Learning (RL), functioning as a model architecture selection controller, has been extensively used in automating CNN model designs [28]. Zoph et al. [11] first used a RNN to generate the string description of 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. [29] proposed a 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 network from scratch during the exploration of the architecture space. To enable more efficient training, Cai et al. [30] proposed a RL-based method where weights for historical network models can be reused to train the current model. However, a noticeable limitation for RL-based NAS is the low scalability, especially when the searching space is very large, since the candidate models are sequentially dependent on each other for progressive optimization.

Bayesian Optimization (BO) is a family of algorithms that build a probability model of the objective function determining the best expected neural network architecture [31]. Early work proposed using the tree-based frameworks such as random forest and tree Parzen estimators [32]. 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. [33] proposed SMAC which is a tree-based algorithm that uses random forests to estimate the error density. Gaussian processes are also widely used in the Bayesian optimization. Kandasamy et al. [34] proposed a Gaussian process based BO framework for searching multi-layer perceptron and convolutional neural network architectures, which is performed sequentially where 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], [31]. EA-based NAS has been widely used for identifying suitable CNN model for a specific task such as image denoising, in-painting and super-resolution [35]. Different EA-based algorithms may use different types of genome encoding methods for the neural model architectures. For example, Genetic-CNN [36] proposed to represent each network structure in a fixed-length binary string, where each element in the string corresponds to a specific kind of operation. Masanori et al. [37] proposed to use Cartesian genetic programming to represent the CNN structure and connectivity, which can represent variable-length network structures and skip connections.

C. Graph Neural Network Architecture Search

Most existing work focuses on NAS of CNN models learning on 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 [15] proposed a graph neural architecture search method based on the 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 validation data set. Auto-GNN [16] follows a similar architecture searching paradigm as GraphNAS while proposing a parameter sharing strategy that enables homogeneous architectures to share parameters. These works all focus on the GNN structures (e.g., aggregators and activation functions) optimization with fixed learning parameters. However, GNN structures and hyper parameters would impact each other in that the moderate change of learning parameters (e.g., learning rate) could severely degrade the accuracy of the optimal GNN architecture achieved by existing methods.

In comparison, studying from a different line we aim to evaluate the effectiveness of evolutionary method for GNN architecture search, and propose a novel framework through the evolution of individual models in a large GNN architecture space. In addition, other than optimizing the GNN structures as existing methods, we propose to evolve and optimize both GNN structures and learning parameters given that they may impact each other in the searching process. More specific, we propose a two-phase encoding scheme which uses two strings to respectively represent the GNN structures and hyper parameters. By this way, we are able to evolve and optimize both structures and learning parameters to fit each other.

III. Problem Definition

In this paper, the objective is to identify the optimal GNN architecture by NAS for network representation learning (a.k.a network embedding) by doing a 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 $\mathbf{V} = \{v_i\}_{i=1,\cdots,|\mathbf{V}|}$ is a set of $|\mathbf{V}|$ unique nodes, $\mathbf{E} = \{e_{i,j}\}_{i,j=1,\cdots,|\mathbf{V}|;\ i \neq j}$ 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 $f : G \rightarrow \{\mathbf{h}_i\}_{i=1,\cdots,|\mathbf{V}|}$ 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 learning parameter space $\mathrm{P} \in \mathbb{R}^{|\mathcal{P}_1| \times |\mathcal{P}_2| \times \cdots \times |\mathcal{P}_n|}$ have been given, where $\mathcal{S}_{i=1,2,\cdots,m} \in \mathbb{R}^{|\mathcal{S}_i|}$ is the set of candidate choices for the i^{th} structure component (e.g., GNN aggregator) and $\mathcal{P}_{j=1,2,\cdots,n} \in \mathbb{R}^{|\mathcal{P}_j|}$ is the set of candidate choices for the j^{th} learning parameter. m and n are respectively the numbers of structure components and learning parameters 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 learning parameter $\mathcal{P}_j$, such that the constructed GNN model $f = \{\mathcal{S}_1^{best}, \mathcal{S}_2^{best}, \cdots, \mathcal{S}_m^{best}; \mathcal{P}_1^{best}, \mathcal{P}_2^{best}, \cdots, \mathcal{P}_n^{best}\}$ could achieve the optimal embedding performance learning on the target network G.

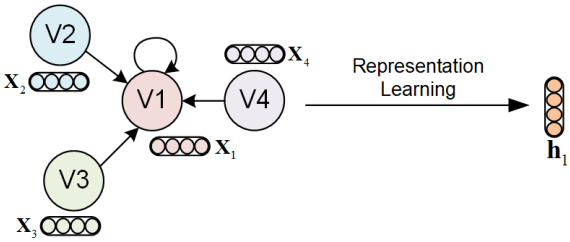

Fig. 2. The representation learning scheme of GNN models.

IV. Preliminaries

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

A. 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. 2, 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 above representation learning scheme:

1. **Attention Function ($\mathcal{S}_1$).** While each node aggregates features from its neighbors, different neighborhood nodes may have different contributions aligned with the affinities between nodes [24]. The attention function aims to learn an importance weight w_{ij} for each edge relationship $e_{i,j}$ between the two corresponding nodes v_i and v_j. $\mathcal{S}_1$ denotes the set of candidate attention functions.
2. **Attention Head ($\mathcal{S}_2$).** Instead of applying the attention function once, studies [38] show that it is beneficial to perform multiple attentions independently. 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 to form the representation $\mathbf{h}_i$. $\mathcal{S}_3$ denotes the set of candidate aggregation functions.

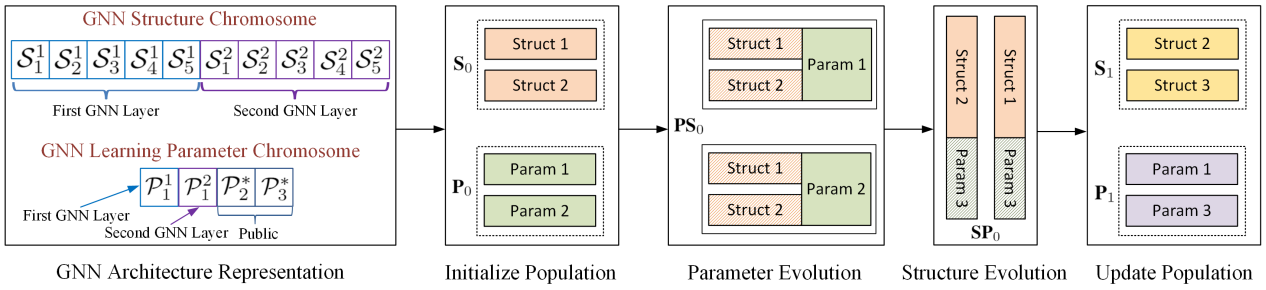

Fig. 3. The proposed Genetic-GNN model for GNN architecture optimization. For simplicity, we demonstrate one evolution step of a two-layer GNN, where both structure and parameter populations have two individuals. **First**, the GNN structure population ($\mathbf{S}_0$) and parameter population ($\mathbf{P}_0$) 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.

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 probability vector for 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. $\mathcal{S}_5$ denotes the set of candidate dimension choices.

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

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

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 l^{th} attention head. One can stack multiple GNN layers to form a multi-layer GNN model. At the k^{th} layer, assume the action string is instantiated as $\mathrm{S}_k = \{\mathcal{S}_1^k, \mathcal{S}_2^k, \mathcal{S}_3^k, \mathcal{S}_4^k, \mathcal{S}_5^k\}$, then the output representation $\mathbf{h}_i^{(k)} \in \mathbb{R}^{\mathcal{S}_5^k}$ is written as:

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

where $\mathbf{h}_i^{(k-1)}$ is the output representation by $(k-1)^{th}$ 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:

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

The weight matrix $\mathbf{W}_l$ at each GNN layer is trained in a semi-supervised manner, *i.e.*, by performing node classification [4], [24] optimized with the gradient decent algorithm.

B. Genetic Algorithm

Genetic Algorithm (GA) is a kind of evolutionary algorithm motivated by the principle of natural selection and genetics [39]. The search 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 form 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 in 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) **Selection**(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 information (e.g., swap the genes) and produce 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 update the population by adding the new individuals.

V. The Proposed Method

This section presents a Genetic Graph Neural Network (Genetic-GNN) NAS framework to evolve the GNN architecture. As shown in Fig. 3, Genetic-GNN can be organized in three main components set to optimize both the GNN structure and learning parameters, including GNN architecture representation & population initialization, GNN learning parameter

evolution, and GNN structure evolution. We elaborate the three components in the following.

A. Architecture Representation & Population Initialization

As discussed in previous sections, the optimizations of GNN structure and learning parameters are dependent on each other. Existing works optimize only GNN structures may end up with a suboptimal searched model since the change of learning parameters could severely degrade the fine-tuned GNN structure. Therefore, we recommend evolving both GNN structure and learning parameters for reliable NAS.

In this paper, we are specifically interested in optimizing the following three types of learning parameter, although Genetic-GNN is a general framework which is flexible to include other significant parameters:

1. **Dropout Rate** ($\mathcal{P}_1$)**.** Overfitting is a common issue when training neural network models. Dropout is a technique for addressing this problem, which meanwhile helps to reduce the training complexity for large networks [40]. 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 rate values.
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 weigh decay rate values.
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 decent 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 rate values.

The weigh 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 learning parameters $\mathcal{P}_2^*$ and $\mathcal{P}_3^*$ are set once and maintain public through multiple layers of a GNN model.

Assume we search the optimal architecture for a two-layer GNN model, as shown in Fig. 3, the GNN structure can be represented by a string/chromosome:

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

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

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

where public 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):

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

TABLE I
THE SEARCH SPACE FOR STRUCTURE COMPONENTS.

Component	Search Space
$\mathcal{S}_1$	listed in Table II
$\mathcal{S}_2$	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

TABLE II
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

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

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

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 [15], the candidate choices or search space for each GNN structure component are summarized in Table I and Table II. Similarly, we define the search space for each learning parameter which is summarized in Table III. Based on the chromosomes (e.g., Eq. (4) and Eq. (5)) and their respective search spaces (e.g., Table I and Table III), the GNN structure population $\mathbf{S}_0 = \{struct_i\}_{i=1,\cdots,N_s}$ and learning parameter population $\mathbf{P}_0 = \{param_j\}_{j=1,\cdots,N_p}$ are respectively created and initialized, *i.e.*, feeding each structure component in Eq. (4) and learning parameter in Eq. (5) with values randomly selected from their respective search 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

TABLE III
THE SEARCH SPACE FOR LEARNING PARAMETERS.

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

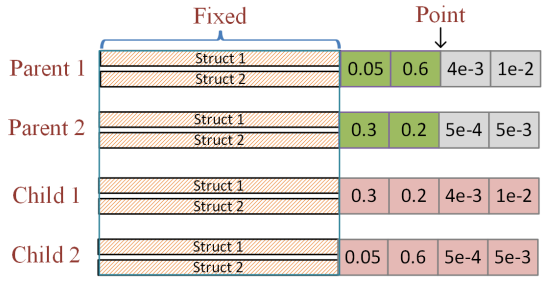

Fig. 4. An example to show the crossover process between two parents, where the point index for learning parameter crossover is 2.

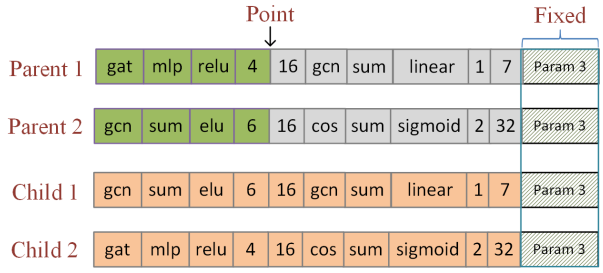

Fig. 5. An example to show the crossover process between two parents, where the point index for structure crossover is 4.

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.

B. GNN Learning 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 should be measured regarding all individuals in the structure population. To this end, as shown in Fig. 3 (e.g., $k = 0$), we combine $\mathbf{S}_0$ and $\mathbf{P}_0$ to create an intermediate population $\mathbf{PS}_0 = \{PS_j\}_{j=1,\cdots,N_p}$, where each individual $PS_j = \{f_{i,j}\}_{i=1,\cdots,N_s}$ encapsulates a set of GNN models with the same learning 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:

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

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 motivation is that we seek to find a parameter setting $param_j$ that fits the entire structure population $\mathbf{S}_0$ while simultaneously considering its fit to the best structure individual $struct_b \in \mathbf{S}_0$. α is a balance parameter which allows us to adjust the importance between these two sides 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. 4 shows the crossover step between two selected parents, where the crossover only happens for the learning parameters with the GNN structures fixed. 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. 3.

C. GNN Structure Evolution

Once the optimal learning parameter $param_j \in \mathbf{P}_{k+1}$ for $\mathbf{S}_k$ has been identified, the structure evolution component aims to evolve $\mathbf{S}_k$ to identify the optimal GNN structure. For this purpose, as the case (e.g., $k = 0$) shown in Fig. 3, an intermediate population $\mathbf{SP}_0 = \{SP_i\}_{i=1,\cdots,N_s}$ 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:

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

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 learning parameter $\mathbf{P}_0$ based on the standard GA algorithm. For example, Fig. 5 shows the crossover process by altering only the structural parts of the two parents while keeping the learning parameters fixed.

D. Algorithm Explanation

The learning parameter evolution and GNN structure evolution proceed in an alternating manner. Before evolving the structure population $\mathbf{S}_k$, the parameter population $\mathbf{P}_k$ is evolved to identify the optimal learning parameters 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 $\mathbf{S}_{k+1}$. Above alternating process is performed iteratively to finally achieve a multi-layer GNN architecture with both optimal structures and learning parameters. The training procedure of Genetic-GNN is summarized in Algorithm 1, where K_s and K_p are number of generations for structure evolution and parameter evolution, respectively. For 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 large search space and individual population.

VI. Experiment

Following literature [15], [16], we test the performance of Genetic-GNN on both transductive and inductive node representation learning by performing the supervised node classification training.

A. Dataset

The four datasets used in the two tasks are summarized in Table IV. Three benchmark citation networks including Cora, Citeseer, and Pubmed are used for transductive node

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 S_0
Initialize the learning parameter population P_0
procedure ARCHITECTEVOLVING(D_train, D_val, S_0, P_0, K_s, K_p)
    for i ← 1 to K_s do
        Construct intermediate GNN model population PS_i
        for j ← 1 to K_p do
            Selection(PS_i)
            Crossover(PS_i)
            Mutate(PS_i)
            Evaluate(PS_i)
            Update(PS_i)
        end for
        Construct intermediate GNN model population SP_i
        Selection(SP_i)
        Crossover(SP_i)
        Mutate(SP_i)
        Evaluate(SP_i)
        Update(SP_i)
    end for
end procedure
```

TABLE IV
BENCHMARK NETWORK DATASETS CHARACTERISTICS.

Items	Cora	Citeseer	PubMed	PPI
# Nodes	2,708	3,327	19,717	56,944
# Features	1,433	3,703	500	50
# Classes	7	6	3	121
# Training Nodes	140	120	60	44,906
# Validation Nodes	500	500	500	6,514
# Testing Nodes	1,000	1,000	1,000	5,524

representation learning, where 20 nodes per lass for training (e.g., the train set D_{train}), 500 nodes for validation (e.g., the validation D_{val}) and 1,000 nodes for testing.

We use a protein-protein interaction (PPI) dataset for inductive node representation learning, which contains 20 graphs for training, 2 graphs for validation and 2 graphs for testing. Each graph have 2,372 nodes on average, and each node has 50 features including positional gene sets, motif gene sets and immunological signatures. Each node corresponds to multiple labels from the total of 121 classes.

B. Baseline

We compare Genetic-GNN with the following state-of-the-art methods which adopt either handcrafted GNN architecture or GNN architecture search.

Methods based on Handcrafted GNN Architecture:

- **Chebyshev** [19] adapts the traditional CNN to learn on graphs by using the Chebyshev polynomial basis to represent the spectral CNNs 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 nodes local neighborhood.
- **GAT** [24] is a method built on the GCN model. It introduces an attention mechanism at the node level, which allows each node specifies different weights to different nodes in a neighborhood.
- **LGCN** [41] 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 on the transformed graph.

Methods based on GNN Architecture Search

- **GraphNAS** [15] 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**[16] 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.

Following literature [15], Chebyshev and GCN are for transductive learning since they require the whole graph structure and nodes to be available in 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 and Auto-GNN are used for both transductive and inductive embedding learning.

C. 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, the balance parameter α in Eq. (8) between 0.2 and 1.0. For comparison, the default hyper parameters 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 parameter evolving generations K_p as 10, the balance ratio α as 0.6, the numbers of parents for structure and parameter genetics are respectively 10 and 4, the numbers of child for structure and parameter genetics are respectively 4 and 2, the mutation probability for both structure and parameter evolution as 0.02.

For the transductive learning, we aim to identify the optimal architecture of a two-layer GNN model within the search space, while for the inductive learning a three-layer GNN

TABLE V
THE TRANSDUCTIVE NODE CLASSIFICATION RESULTS ON CITATION NETWORKS.

Categories	Methods	# Layers	Cora	Citeseer	Pubmed
			Accuracy		
Handcrafted GNN Architecture	Chebyshev	2	81.2%	69.8%	74.4%
	GCN	2	81.5%	70.3%	79.0%
	GAT	2	$83.0 \pm 0.7\%$	$72.5 \pm 0.7\%$	$79.0 \pm 0.3\%$
	LGCN	2	$83.3 \pm 0.5\%$	$73.0 \pm 0.6\%$	$79.5 \pm 0.2\%$
GNN Neural Architecture Search	GraphNAS	2	$84.2 \pm 1.0\%$	$73.1 \pm 0.9\%$	$79.6 \pm 0.4\%$
	Auto-GNN	2	$83.6 \pm 0.3\%$	$73.8 \pm 0.7\%$	$79.7 \pm 0.4\%$
	Genetic-GNN	2	$83.8 \pm 0.5\%$	$73.5 \pm 0.8\%$	$79.2 \pm 0.6\%$

TABLE VI
THE INDUCTIVE NODE CLASSIFICATION RESULTS ON THE PPI NETWORK.

Categories	Methods	# Layers	PPI
			Micro-F1
Handcrafted	GraphSAGE (lstm)	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	$99.2 \pm 0.1\%$
	Genetic-GNN	3	$98.6 \pm 0.4\%$

model is optimized in this paper. We train 200 epochs for each specific GNN model, where the accuracy and Micro-F1 are used as metrics for the transductive and inductive embedding learning tasks, respectively.

VII. 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.

A. Graph NAS-based Embedding Learning Performance

Table V shows the comparative results of all baselines. We can have two main conclusions:

- The NAS-based methods including GraphNAS, Auto-GNN and Genetic-GNN can achieve better results than the handcraft-based GNN models on all three 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 hyper parameters, which has a very low chance to obtain an optimal model. In comparison, the NAS-based methods are able to search and validate the performance of the candidate GNN models automatically given the graph data and learning task, which can gradually optimize the model performance with litter or even no human intervention.
- For the category of NAS-based methods, 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, which demonstrate effectiveness of the evolutionary algorithm for GNN architecture search. Compared with the GraphNAS and Auto-GNN, our model is able to optimize both GNN structure and learning parameters, which greatly increased the automation of GNN NAS.

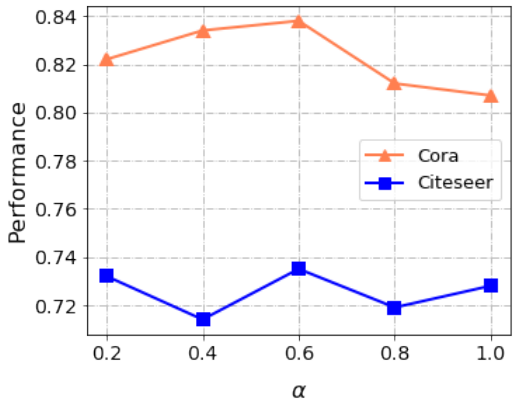

Fig. 6. Influence of the imbalance parameter α.

Table VI shows the node classification results of inductive embedding learning on the PPI data. Similar conclusions can be made as the transductive learning. First, the category of automated methods perform generally better than the handcrafted methods. Second, performance of our Genetic-GNN model is as good as the state-of-the-art model GraphNAS. In this paper, due to the limitation of computation resource, we only set the maximum population size as 50 and the evolution generations as 50. However, with the increase of population size and evolution generations, our model has a potential to achieve better performance.

B. 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. 6. We can observe the best setting for both data is 0.6. Fig. 7 shows the influence of

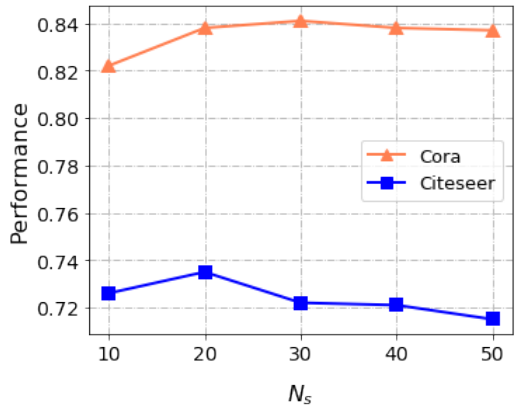

Fig. 7. Influence of the number of initial structure population size K_s.

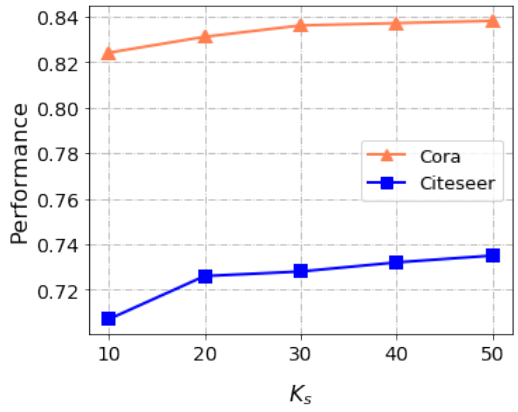

Fig. 8. Influence of the number of evolving generations K_s for structure population.

Fig. 9. Validation and test performance on Cora change over the training iterations.

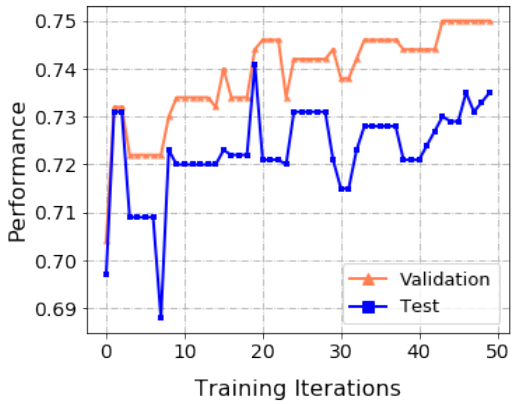

Fig. 10. Validation and test performance on Citeseer change over the training iterations.

the number of structure population size N_s. We can observe with the increase of N_s, the performance first goes up and then decrease on both citation networks. Generally, larger population size means larger search space which tends to generate better resulting solution. However, the large search space normally requires more evolution generations to finally identify an optimal model, which probably explains the performance decrease as population size increases in Fig. 7. Fig. 8 shows the influence of evolution generation for the structure population, where the performance gradually increases over the generations.

Fig. 9 and Fig. 10 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, through they have some turbulence. The reason for the unstable curves is that both validation and test sets are unseen, and the best performance on the train set cannot guarantee the best performances on the validation and test sets.

VIII. 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 model with a well-defined search space. Instead of only optimizing the GNN structures with fixed learning parameters, Genetic-GNN is able to evolve and optimize both structure and parameter to fit each other. The experimental results and parameter sensitivity tests demonstrated 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 a another parameter individual, their model structures remain the same, thereby the model weight parameters can be shared between the two models.

References

[1] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Transactions on Big Data*, 2018.

[2] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[4] T. N. Kipf and 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, and 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, and F. Wang, "Graph convolutional networks for computational drug development and discovery," *Brief. in bioinformatics*, 2019.

[7] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[8] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[9] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[10] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[12] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[13] X. Dong and 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.

[14] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, and X. Wang, "Renas: Reinforced evolutionary neural architecture search," in *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 2019, pp. 4787–4796.

[15] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," 2019.

[16] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," 2019.

[17] C. Su, J. Tong, Y. Zhu, P. Cui, and F. Wang, "Network embedding in biomedical data science," *Briefings in bioinformatics*, vol. 21, no. 1, pp. 182–197, 2020.

[18] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[19] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[20] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[21] X. Zhang, W. Chen, and H. Yan, "Tline: Scalable transductive network embedding," in *Asia Information Retrieval Symposium*. Springer, 2016, pp. 98–110.

[22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[23] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in neural information processing systems*, 2016, pp. 1993–2001.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[25] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and 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.

[26] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "Textnas: A neural architecture search space tailored for text representation." in *AAAI*, 2020, pp. 9242–9249.

[27] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[28] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "Reinforcement learning for neural architecture search: A review," *Image and Vision Computing*, vol. 89, pp. 57–66, 2019.

[29] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[30] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[31] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *arXiv preprint arXiv:1908.00709*, 2019.

[32] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[33] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*. Springer, 2011, pp. 507–523.

[34] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in neural information processing systems*, 2018, pp. 2016–2025.

[35] K. Ho, A. Gilbert, H. Jin, and J. Collomosse, "Neural architecture search for deep image prior," *arXiv preprint arXiv:2001.04776*, 2020.

[36] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[37] M. Suganuma, S. Shirakawa, and 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.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[39] R. H. Sheikh, M. M. Raghuwanshi, and 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.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[41] H. Gao, Z. Wang, and 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.

[42] C. Shang, Q. Liu, K.-S. Chen, J. Sun, J. Lu, J. Yi, and J. Bi, "Edge attention-based multi-relational graph convolutional networks," *arXiv*, pp. arXiv–1802, 2018.