Automated Machine Learning on Graphs: A Survey

Ziwei Zhang* , **Xin Wang*** and **Wenwu Zhu†**
Tsinghua University, Beijing, China
zw-zhang16@mails.tsinghua.edu.cn, {xin_wang,wwzhu}@tsinghua.edu.cn

Abstract

Machine learning on graphs has been extensively studied in both academic and industry. However, as the literature on graph learning booms with a vast number of emerging methods and techniques, it becomes increasingly difficult to manually design the optimal machine learning algorithm for different graph-related tasks. To solve this critical challenge, automated machine learning (AutoML) on graphs which combines the strength of graph machine learning and AutoML together, is gaining attention from the research community. Therefore, we comprehensively survey AutoML on graphs in this paper[1], primarily focusing on hyper-parameter optimization (HPO) and neural architecture search (NAS) for graph machine learning. We further overview libraries related to automated graph machine learning and in-depth discuss AutoGL, the first dedicated open-source library for AutoML on graphs. In the end, we share our insights on future research directions for automated graph machine learning. This paper is the first systematic and comprehensive review of automated machine learning on graphs to the best of our knowledge.

1 Introduction

Graph data is ubiquitous in our daily life. We can use graphs to model the complex relationships and dependencies between entities ranging from small molecules in proteins and particles in physical simulations to large national-wide power grids and global airlines. Therefore, machine learning on graphs has long been an important research direction for both academics and industry [1]. In particular, network embedding [2; 3; 4; 5] and graph neural networks (GNNs) [6; 7; 8] have drawn increasing attention in the last decade. They are successfully applied to recommendation systems [9; 10], fraud detection [11], bioinformatics [12; 13], physical simulation [14], traffic forecasting [15; 16], knowledge representation [17], drug re-purposing [18; 19] and pandemic prediction [20] for Covid-19.

Despite the popularity of graph machine learning algorithms, the existing literature heavily relies on manual hyper-parameter or architecture design to achieve the best performance, resulting in costly human efforts when a vast number of models emerge for various graph tasks. Take GNNs as an example. At least one hundred new general-purpose architectures have been published in top-tier machine learning and data mining conferences in the year 2020 alone, not to mention cross-disciplinary researches of task-specific designs. More and more human efforts are inevitably needed if we stick to the manual try-and-error paradigm in designing the optimal algorithms for targeted tasks.

On the other hand, automated machine learning (AutoML) has been extensively studied to reduce human efforts in developing and deploying machine learning models [21; 22; 23]. Complete AutoML pipelines have the potential to automate every step of machine learning, including auto data collection and cleaning, auto feature engineering, and auto model selection and optimization, etc. Due to the popularity of deep learning models, hyper-parameter optimization (HPO) [24; 25; 26] and neural architecture search (NAS) [27] are most widely studied. AutoML has achieved or surpassed human-level performance [28; 29; 30] with little human guidance in areas such as computer vision [31; 32].

Automated machine learning on graphs, combining the advantages of AutoML and graph machine learning, naturally serves as a promising research direction to further boost the model performance, which has attracted an increasing number of interests from the community. In this paper, we provide a comprehensive and systematic review of automated machine learning on graphs, to the best of our knowledge, for the first time. Specifically, we focus on two major topics: HPO and NAS of graph machine learning. For HPO, we focus on how to develop scalable methods. For NAS, we follow the literature and compare different methods from search spaces, search strategies, and performance estimation strategies. How different methods tackle the challenges of AutoML on graphs are discussed along the way. Then, we review libraries related to automated graph machine learning and discuss AutoGL, the first dedicated framework and open-source library for automated machine learning on graphs. We high-

*Equal contributions
†Corresponding author
[1]We provide a paper collection about AutoML on graphs at https://github.com/THUMNLab/awesome-auto-graph-learning.

light the design principles of AutoGL and briefly introduce its usages, which are all specially designed for AutoML on graphs. We believe our review in this paper will significantly facilitate and further promote the studies and applications of automated machine learning on graphs.

The rest of the paper is organized as follows. In Section 2, we point out the challenges for automated graph machine learning and briefly introduce basic formulations of machine learning on graphs and AutoML. We comprehensively review HPO on graph machine learning in Section 3 and NAS for graph machine learning in Section 4, followed by our overview and discussions of related libraries in Section 5. Lastly, we outline future research opportunities in Section 6.

2 Automated Machine Learning on Graphs

Automated machine learning on graphs, which non-trivially combines the strength of AutoML and graph machine learning, faces the following challenges.

- **The uniqueness of graph machine learning:** Unlike audio, image, or text, which has a grid structure, graph data lies in a non-Euclidean space [33]. Thus, graph machine learning usually has unique architectures and designs. For example, typical NAS methods focus on the search space for convolution and recurrent operations, which is distinct from the building blocks of GNNs [34].
- **Complexity and diversity of graph tasks:** As aforementioned, graph tasks per se are complex and diverse, ranging from node-level to graph-level problems, and with different settings, objectives, and constraints [35]. How to impose proper *inductive bias* and integrate *domain knowledge* into a graph AutoML method is indispensable.
- **Scalability:** Many real graphs such as social networks or the Web are incredibly large-scale with billions of nodes and edges [36]. Besides, the nodes in the graph are interconnected and cannot be treated as independent samples. Designing scalable AutoML algorithms for graphs poses significant challenges since both graph machine learning and AutoML are already notorious for being compute-intensive.

Approaches with HPO or NAS for graph machine learning reviewed in later sections target handling at least one of these three challenges. We briefly introduce basic problem formulations before moving to the next section.

2.1 Machine Learning on Graphs

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \left\{v_1, v_2, ..., v_{|\mathcal{V}|}\right\}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. The neighborhood of node v_i is denoted as $\mathcal{N}(i) = \{v_j : (v_i, v_j) \in \mathcal{E}\}$. The nodes can also have features denoted as $\mathbf{F} \in \mathbb{R}^{|\mathcal{V}| \times f}$, where f is the number of features. We use bold uppercases (e.g., $\mathbf{X}$) and bold lowercases (e.g., $\mathbf{x}$) to represent matrices and vectors, respectively.

Most tasks of graph machine learning can be divided into the following two categories:

- Node-level tasks: the tasks are associated with individual nodes or pairs of nodes. Typical examples include node classification and link prediction.
- Graph-level tasks: the tasks are associated with the whole graph, such as graph classification and graph generation.

For node-level tasks, graph machine learning models usually learn a node representation $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}| \times d}$ and then adopt a classifier or predictor on the node representation to solve the task. For graph-level tasks, a representation for the whole graph is learned and fed into a classifier/predictor.

GNNs are the current state-of-the-art in learning node and graph representations. The message-passing framework of GNNs [37] is formulated as follows.

$$\mathbf{m}_i^{(l)} = \text{AGG}^{(l)}\left(\left\{a_{ij}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_i^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right) \quad (1)$$

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\text{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right), \quad (2)$$

where $\mathbf{h}_i^{(l)}$ denotes the node representation of node v_i in the l^{th} layer, $\mathbf{m}^{(l)}$ is the message for node v_i, $\text{AGG}^{(l)}(\cdot)$ is the aggregation function, $a_{ij}^{(l)}$ denotes the weights from node v_j to node v_i, $\text{COMBINE}^{(l)}(\cdot)$ is the combining function, $\mathbf{W}^{(l)}$ are learnable weights, and $\sigma(\cdot)$ is an activation function. The node representation is usually initialized as node features $\mathbf{H}^{(0)} = \mathbf{F}$, and the final representation is obtained after L message-passing layers $\mathbf{H} = \mathbf{H}^{(L)}$.

For the graph-level representation, pooling methods (also called readout) are applied to the node representations

$$\mathbf{h}_{\mathcal{G}} = \text{POOL}\left(\mathbf{H}\right), \quad (3)$$

i.e., $\mathbf{h}_{\mathcal{G}}$ is the representation of $\mathcal{G}$.

2.2 AutoML

Many AutoML algorithms such as HPO and NAS can be formulated as the following bi-level optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}\left(\mathbf{W}^*(\alpha), \alpha\right) \\ \text{s.t.} \quad &\mathbf{W}^*(\alpha) = \arg\min_{\mathbf{W}}\left(\mathcal{L}_{train}\left(\mathbf{W}, \alpha\right)\right), \end{aligned} \quad (4)$$

where α is the optimization objective of the AutoML algorithm, e.g., hyper-parameters in HPO and neural architectures in NAS, $\mathcal{A}$ is the feasible space for the objective, and $\mathbf{W}(\alpha)$ are trainable weights in the graph machine learning models. Essentially, we aim to optimize the objective in the feasible space so that the model achieves the best results in terms of a validation function, and $\mathbf{W}^*$ indicates that the weights are fully optimized in terms of a training function. Different AutoML methods differ in how the feasible space is designed and how the objective functions are instantiated and optimized since directly optimizing Eq. (4) requires enumerating and training every feasible objective, which is prohibitive.

Typical formulations of AutoML on graphs need to properly integrate the above formulations in Section 2.1 and Section 2.2 to form a new optimization problem.

3 HPO for Graph Machine Learning

In this section, we review HPO for machine learning on graphs. The main challenge here is scalability, i.e., a real graph can have billions of nodes and edges, and each trial on the graph is computationally expensive. Next, we elaborate on how different methods tackle the efficiency challenge. Notice that we omit some straightforward HPO methods such as random search and grid search [24].

AutoNE [38] first tackles the efficiency problem of HPO on graphs by proposing a transfer paradigm that samples subgraphs as proxies for the large graph, which is similar in principle to sampling instances in previous HPO methods[39]. Specifically, AutoNE has three modules: the sampling module, the signature extraction module, and the meta-learning module. In the sampling module, multiple representative subgraphs are sampled from the large graph using a multi-start random walk strategy. Then, AutoNE conducts HPO on the sampled subgraphs using Bayesian optimization [26] and learns representations of subgraphs using the signature extraction module. Finally, the meta-learning module extracts meta-knowledge from HPO results and representations of subgraphs. AutoNE fine-tunes hyper-parameters on the large graph using the meta-knowledge. In this way, AutoNE achieves satisfactory results while maintaining scalability since multiple HPO trials on the sampled subgraphs and a few HPO trails on the large graph are properly integrated.

JITuNE [40] proposes to replace the sampling process of AutoNE with graph coarsening to generate a hierarchical graph synopsis. A similarity measurement module is proposed to ensure that the coarsened graph shares sufficient similarity with the large graph. Compared with sampling, such graph synopsis can better preserve graph structural information. Therefore, JITuNE argues that the best hyper-parameters in the graph synopsis can be directly transferred to the large graph. Besides, since the graph synopsis is generated in a hierarchy, the granularity can be more easily adjusted to meet the time constraints of downstream tasks.

HESGA [41] proposes another strategy to improve efficiency using a hierarchical evaluation strategy together with evolutionary algorithms. Specifically, HESGA proposes to evaluate the potential of hyper-parameters by interrupting training after a few epochs and calculating the performance gap with respect to the initial performance with random model weights. This gap is used as a fast score to filter out unpromising hyper-parameters. Then, the standard full evaluation, i.e., training until convergence, is adopted as the final assessor to select the best hyper-parameters to be stored in the population of the evolutionary algorithm.

Besides efficiency, AutoGM [42] focuses on proposing a unified framework for various graph machine learning algorithms. Specifically, AutoGM finds that many popular GNNs can be characterized in a framework similar to Eq. (1) with five hyper-parameters: the number of message-passing neighbors, the number of message-passing steps, the aggregation function, the dimensionality, and the non-linearity. AutoGM adopts Bayesian optimization to optimize these hyper-parameters.

4 NAS for Graph Machine Learning

NAS methods can be compared in three aspects [27]: search space, search strategy, and performance estimation strategy. Next, we review NAS methods for graph machine learning from these three aspects. We mainly review NAS for GNNs fitting Eq. (1) and summarize the characteristics of different methods in Table 1.

4.1 Search Space

The first challenge of NAS on graphs is the search space design since the building blocks of graph machine learning are usually distinct from other deep learning models such as CNNs or RNNs. For GNNs, the search space can be divided into the following four categories.

Micro Search Space

Following the message-passing framework in Eq. (1), the micro search space defines how nodes exchange messages with others in each layer. Commonly adopted micro search spaces [34; 43] compose the following components:

- Aggregation function AGG$(\cdot)$: SUM, MEAN, MAX, and MLP.
- Aggregation weights a_{ij}: common choices are listed in Table 2.
- Number of heads when using attentions: 1, 2, 4, 6, 8, 16, etc.
- Combining function COMBINE$(\cdot)$: CONCAT, ADD, and MLP.
- Dimensionality of $\mathbf{h}^l$: 8, 16, 32, 64, 128, 256, 512, etc.
- Non-linear activation function $\sigma(\cdot)$: Sigmoid, Tanh, ReLU, Identity, Softplus, Leaky ReLU, ReLU6, and ELU.

However, directly searching all these components results in thousands of possible choices in a single message-passing layer. Thus, it may be beneficial to prune the space to focus on a few crucial components depending on applications and domain knowledge [44].

Macro Search Space

Similar to residual connections and dense connections in CNNs, node representations in one layer of GNNs do not necessarily solely depend on the immediate previous layer [62; 63]. These connectivity patterns between layers form the macro search space. Formally, such designs are formulated as

$$\mathbf{H}^{(l)} = \sum\nolimits_{j<l} \mathcal{F}_{jl}\left(\mathbf{H}^{(j)}\right), \tag{5}$$

where $\mathcal{F}_{jl}(\cdot)$ can be the message-passing layer in Eq. (1), ZERO (i.e., not connecting), IDENTITY (e.g., residual connections), or an MLP. Since the dimensionality of $\mathbf{H}^{(j)}$ can vary, IDENTITY can only be adopted if the dimensionality of each layer matches.

Pooling Methods

In order to handle graph-level tasks, information from all the nodes is aggregated to form graph-level representations using the pooling operation in Eq (3). [51] propose a pooling search space including row- or column-wise sum, mean, or maximum, attention pooling, attention sum, and flatten. More advanced methods such as hierarchical pooling [64] could also be added to the search space with careful designs.

Method	Search space					Tasks		Search Strategy	Performance Estimation	Other Characteristics
	Micro	Macro	Pooling	HP	Layers	Node	Graph			
GraphNAS [34]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	-	-
AGNN [43]	✓	✗	✗	✗	Fixed	✓	✗	Self-designed controller + RL	Inherit weights	-
SNAG [44]	✓	✓	✗	✗	Fixed	✓	✗	RNN controller + RL	Inherit weights	Simplify the micro search space
PDNAS [45]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	-
POSE [46]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	Support heterogenous graphs
NAS-GNN [47]	✓	✗	✗	✓	Fixed	✓	✗	Evolutionary algorithm	-	-
AutoGraph [48]	✓	✓	✗	✗	Various	✓	✗	Evolutionary algorithm	-	-
GeneticGNN [49]	✓	✗	✗	✓	Fixed	✓	✗	Evolutionary algorithm	-	-
EGAN [50]	✓	✓	✗	✗	Fixed	✓	✓	Differentiable	One-shot	Sample small graphs for efficiency
NAS-GCN [51]	✓	✓	✓	✗	Fixed	✗	✓	Evolutionary algorithm	-	Handle edge features
LPGNAS [52]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	Single-path one-shot	Search for quantisation options
You *et al.* [53]	✓	✓	✗	✓	Various	✓	✓	Random search	-	Transfer across datasets and tasks
SAGS [54]	✓	✗	✗	✗	Fixed	✓	✓	Self-designed algorithm	-	-
Peng *et al.* [55]	✓	✗	✗	✗	Fixed	✗	✓	CEM-RL [56]	-	Search spatial-temporal modules
GNAS[57]	✓	✓	✗	✗	Various	✓	✓	Differentiable	One-shot	-
AutoSTG[58]	✗	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot+meta learning	Search spatial-temporal modules
DSS[59]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	Dynamically update search space
SANE[60]	✓	✓	✗	✗	Fixed	✓	✗	Differentiable	One-shot	-
AutoAttend[61]	✓	✓	✗	✗	Fixed	✓	✓	Evolutionary algorithm	One-shot	Cross-layer attention

Table 1: A summary of different NAS methods for graph machine learnings.

Type	Formulation
CONST	$a_{ij}^{\text{const}} = 1$
GCN	$a_{ij}^{\text{gcn}} = \frac{1}{\sqrt{\lvert\mathcal{N}(i)\rvert\lvert\mathcal{N}(j)\rvert}}$
GAT	$a_{ij}^{\text{gat}} = \text{LeakyReLU}\left(\text{ATT}\left(\mathbf{W}_a\left[\mathbf{h}_i, \mathbf{h}_j\right]\right)\right)$
SYM-GAT	$a_{ij}^{\text{sym}} = a_{ij}^{\text{gat}} + a_{ji}^{\text{gat}}$
COS	$a_{ij}^{\text{cos}} = \cos\left(\mathbf{W}_a\mathbf{h}_i, \mathbf{W}_a\mathbf{h}_j\right)$
LINEAR	$a_{ij}^{\text{lin}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)$
GENE-LINEAR	$a_{ij}^{\text{gene}} = \tanh\left(\text{sum}\left(\mathbf{W}_a\mathbf{h}_i + \mathbf{W}_a\mathbf{h}_j\right)\right)\mathbf{W}_a'$

Table 2: A typical search space of different aggregation weights.

Hyper-parameters

Besides architectures, other training hyper-parameters can be incorporated into the search space, i.e., similar to jointly conducting NAS and HPO. Typical hyper-parameters include the learning rate, the number of epochs, the batch size, the optimizer, the dropout rate, and the regularization strengths such as the weight decay. These hyper-parameters can be jointly optimized with architectures or separately optimized after the best architectures are found. HPO methods in Section 3 can also be combined here.

Another critical choice is the number of message-passing layers. Unlike CNNs, most currently successful GNNs are shallow, e.g., with no more than three layers, possibly due to the over-smoothing problem [65; 62]. Limited by this problem, the existing NAS methods for GNNs usually preset the number of layers as a small fixed number. How to automatically design deep GNNs while integrating techniques to alleviate over-smoothing remains mostly unexplored. On the other hand, NAS may also bring insights to help to tackle the over-smoothing problem [34].

4.2 Search Strategy

Search strategies can be broadly divided into three categories: architecture controllers trained with reinforcement learning (RL), differentiable methods, and evolutionary algorithms.

Controller + RL

A widely adopted NAS search strategy uses a controller to generate the neural architecture descriptions and train the controller with reinforcement learning to maximize the model performance as rewards. For example, if we consider neural architecture descriptions as a sequence, we can use RNNs as the controller [28]. Such methods can be directly applied to GNNs with a suitable search space and performance evaluation strategy.

Differentiable

Differentiable NAS methods such as DARTS [29] and SNAS [66] have gained popularity in recent years. Instead of optimizing different operations separately, differentiable methods construct a single super-network (known as the *one-shot model*) containing all possible operations. Formally, we denote

$$\mathbf{y} = o^{(x,y)}(\mathbf{x}) = \sum\nolimits_{o\in\mathcal{O}} \frac{\exp(\mathbf{z}_o^{(x,y)})}{\sum_{o'\in\mathcal{O}}\exp(\mathbf{z}_{o'}^{(x,y)})} o(\mathbf{x}), \quad (6)$$

where $o^{(x,y)}(\mathbf{x})$ is an operation in the GNN with input $\mathbf{x}$ and output $\mathbf{y}$, $\mathcal{O}$ are all candidate operations, and $\mathbf{z}^{(x,y)}$ are learnable vectors to control which operation is selected. Briefly speaking, each operation is regarded as a probability distribution of all possible operations. In this way, the architecture and model weights can be jointly optimized via gradient-based algorithms. The main challenges lie in making the NAS algorithm differentiable, where several techniques such as Gumbel-softmax [67] and concrete distribution [68] are resorted to. When applied to GNNs, slight modification may be needed to incorporate the specific operations defined in the search space.

Evolutionary Algorithms

Evolutionary algorithms are a class of optimization algorithms inspired by biological evolution. For NAS, randomly generated architectures are considered initial individuals in a population. Then, new architectures are generated using mutations and crossover operations on the population. The

architectures are evaluated and selected to form the new population, and the same process is repeated. The best architectures are recorded while updating the population, and the final solutions are obtained after sufficient updating steps.

For GNNs, regularized evolution (RE) NAS [32] has been widely adopted. RE's core idea is an aging mechanism, i.e., in the selection process, the oldest individuals in the population are removed. Genetic-GNN [69] also proposes an evolution process to alternatively update the GNN architecture and the learning hyper-parameters to find the best fit of each other.

Combinations

It is also feasible to combine these three types of search strategies mentioned above. For example, AGNN [43] proposes a reinforced conservative search strategy by adopting both RNNs and evolutionary algorithms in the controller and train the controller with RL. By only generating slightly different architectures, the controller can find well-performing GNNs more efficiently. [55] adopt CEM-RL [56], which combines evolutionary and differentiable methods.

4.3 Performance Estimation Strategy

Due to the large number of possible architectures, it is infeasible to fully train each architecture independently. Next, we review some performance estimation strategies.

A commonly adopted "trick" to speed up performance estimation is to reduce fidelity [27], e.g., by reducing the number of epochs or the number of data points. This strategy can be directly generalized to GNNs.

Another strategy successfully applied to CNNs is sharing weights among different models, known as parameter sharing or weight sharing [30]. For differentiable NAS with a large one-shot model, parameter sharing is naturally achieved since the architectures and weights are jointly trained. However, training the one-shot model may be difficult since it contains all possible operations. To further speed up the training process, single-path one-shot model [70] has been proposed where only one operation between an input and output pair is activated during each pass.

For NAS without a one-shot model, sharing weights among different architecture is more difficult but not entirely impossible. For example, since it is known that some convolutional filters are common feature extractors, inheriting weights from previous architectures is feasible and reasonable in CNNs [71]. However, since there is still a lack of understandings of what weights in GNNs represent, we need to be more cautious about inheriting weights [44]. AGNN [43] proposes three constraints for parameter inheritance: same weight shapes, same attention and activation functions, and no parameter sharing in batch normalization and skip connections.

4.4 Discussions

The Search Space

Besides the basic search space presented in Section 4.1, different graph tasks may require other search spaces. For example, meta-paths are critical for heterogeneous graphs [46], edge features are essential in modeling molecular graphs [51], and spatial-temporal modules are needed in skeleton-based recognition [55]. Sampling mechanisms to accelerate GNNs are also critical, especially for large-scale graphs [34]. A suitable search space usually requires careful designs and domain knowledge.

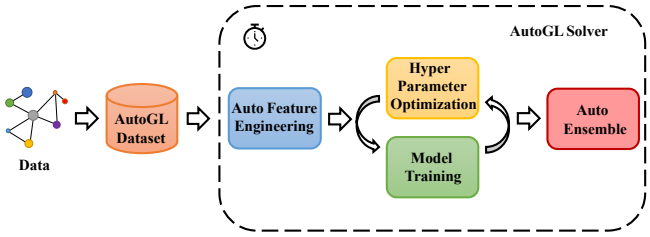

Figure 1: The overall framework of AutoGL.

Transferability

It is non-trivial to transfer GNN architectures across different datasets and tasks due to the complexity and diversity of graph tasks. [72] adopt a fixed set of GNNs as anchors on different tasks and datasets. Then, the rank correlation serves as a metric to measure the similarities between different datasets and tasks. The best-performing GNNs of the most similar tasks are transferred to solve the target task.

The Efficiency Challenge of Large-scale Graphs

Similar to AutoNE introduced in Section 3, EGAN [50] proposes to sample small graphs as proxies and conduct NAS on the sampled subgraphs to improve the efficiency of NAS. While achieving some progress, more advanced and principle approaches are further needed to handle billion-scale graphs.

5 Libraries for AutoML on Graphs

Publicly available libraries are important to facilitate and advance the research and applications of AutoML on graphs. Popular libraries for graph machine learning include PyTorch Geometric [73], Deep Graph Library [74], GraphNets [75], AliGraph [76], Euler [77], PBG [78], Stellar Graph [79], Spektral [80], CodDL [81], OpenNE [82], GEM [83], Karateclub [84], DIG [85], and classical NetworkX [86]. However, these libraries do not support AutoML.

On the other hand, AutoML libraries such as NNI [87], AutoKeras [88], AutoSklearn [89], Hyperopt [90], TPOT [91], AutoGluon [92], MLBox [93], and MLJAR [94] are widely adopted. Unfortunately, because of the uniqueness and complexity of graph tasks, they cannot be directly applied to automate graph machine learning.

Recently, some HPO and NAS methods for graphs such as AutoNE [38], AutoGM [42], GraphNAS [34] GraphGym [53] have open-sourced their codes, facilitating reproducibility and promoting AutoML on graphs. Besides, AutoGL [95][2], the first dedicated library for automated graph learning, is developed. Next, we review AutoGL in detail. Figure 1 shows the overall framework of AutoGL. The main characteristics of AutoGL are three-folded:

- Open-source: all the source codes of AutoGL are publicly available under the MIT license.

[2] AutoGL homepage: https://mn.cs.tsinghua.edu.cn/AutoGL

- Easy to use: AutoGL is designed to be easy to use. For example, less than ten lines of codes are needed to conduct some quick experiments of AutoGL.
- Flexible to be extended: AutoGL adopts a modular design with high-level base classes API and extensive documentations, allowing flexible and customized extensions.

We briefly review the dataset management and four core components of AutoGL: Auto Feature Engineering, Model Training, Hyper-Parameters Optimization, and Auto Ensemble. These components are designed in a modular and object-oriented fashion to enable clear logic flows, easy usages, and flexibility in extending.

AutoGL Dataset. inherits from PyTorch Geometric [73], covering common benchmarks for node and graph classification, including the recent Open Graph Benchmark [35]. Users can easily add customized datasets following documentations.

Auto Feature Engineering. module first processes the graph data using three categories of operations: generators, where new node and edge features are constructed; selectors, filtering out and compressing useless and meaningless features; sub-graph generators, generating graph-level features. Convenient wrappers are also provided to support PyTorch Geometric and NetworkX [96].

Model Training. module handles the training and evaluation process of graph tasks with two functional sub-modules: model and trainer. Model handles the construction of graph machine learning models, e.g., GNNs, by defining learnable parameters and the forward pass. Trainer controls the optimization process for the given model. Common optimization methods, training controls, and regularization methods are packaged as high-level APIs.

Hyper-Parameter Optimization. module conducts HPO for a specified model, covering methods presented in Section 3 such as AutoNE and general-purpose algorithms like random search [24] and Tree Parzen Estimator [97]. The model training module can specify the hyper-parameters, their types (e.g., integer, numerical, or categorical), and feasible ranges. Users can also customize HPO algorithms.

Auto Ensemble. module can automatically integrate the optimized individual models to form a more powerful final model. Two kinds of ensemble methods are provided: voting and stacking. Voting is a simple yet powerful ensemble method that directly averages the output of individual models while stacking trains another meta-model that learns to combine the output of models in a more principled way. The general linear model (GLM) and gradient boosting machine (GBM) are supported as meta-models.

AutoGL Solver. On top of the four modules, another high-level API Solver is proposed to control the overall pipeline. In the Solver, the four modules are organized to form the AutoML solution for graph data. The Solver also provides global controls. For example, the time budget can be explicitly set, and the training/evaluation protocols can be selected.

AutoGL is still actively updated. Key features to be released shortly include neural architecture search, large-scale datasets support, and more graph tasks. For the most up-to-date information, please visit the project homepage. All kinds of inputs and suggestions are also warmly welcomed.

6 Future Directions

- **Graph models for AutoML**: In this paper, we mainly focus on how AutoML methods are extended to graphs. The other direction, i.e., using graphs to help AutoML, is also feasible and promising. For example, we can model neural networks as a directed acyclic graph (DAG) to analyze their structures [98; 72] or adopt GNNs to facilitate NAS [99; 100; 69]. Ultimately, we expect graphs and AutoML to form tighter connections and further facilitate each other.
- **Robustness and explainability**: Since many graph applications are risk-sensitive, e.g., finance and healthcare, model robustness and explainability are indispensable for actual usages. Though there exist some initial studies on the robustness [101] and explainability [102] of graph machine learning, how to generalize these techniques into AutoML on graphs remains to be further explored [103].
- **Hardware-aware models**: To further improve the scalability of automated machine learning on graphs, hardware-aware models may be a critical step, especially in real industrial environments. Both hardware-aware graph models [104] and hardware-aware AutoML models [105; 106; 107] have been studied, but integrating these techniques still poses significant challenges.
- **Comprehensive evaluation protocols**: Currently, most AutoML on graphs are tested on small traditional benchmarks such as three citation graphs, i.e., Cora, CiteSeer, and PubMed [108]. However, these benchmarks have been identified as insufficient to compare different graph machine learning models [109], not to mention AutoML on graphs. More comprehensive evaluation protocols are needed, e.g., on recently proposed graph machine learning benchmarks [35; 110] or new dedicated graph AutoML benchmarks similar to the NAS-bench series [111].

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (No.2020AAA0106300, 2020AAA0107800, 2018AAA0102000) and National Natural Science Foundation of China No.62050110.

References

[1] Mark Newman. *Networks*. Oxford university press, 2018.

[2] Peng Cui et al. A survey on network embedding. *TKDE*, 2018.

[3] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE DEBU*, 2017.

[4] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *KBS*, 2018.

[5] Hongyun Cai et al. A comprehensive survey of graph embedding: Problems, techniques, and applications. *TKDE*, 2018.

[6] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *TKDE*, 2020.

[7] Zonghan Wu et al. A comprehensive survey on graph neural networks. *TNNLS*, 2020.

[8] Jie Zhou et al. Graph neural networks: A review of methods and applications. *arXiv:1812.08434*, 2018.

[9] Rex Ying et al. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

[10] Jianxin Ma et al. Learning disentangled representations for recommendation. In *NeurIPS*, 2019.

[11] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *DMKD*, 2015.

[12] Chang Su et al. Network embedding in biomedical data science. *Briefings in bioinformatics*, 2020.

[13] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 2017.

[14] T Kipf et al. Neural relational inference for interacting systems. *ICML*, 2018.

[15] Yaguang Li et al. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*, 2018.

[16] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*, 2018.

[17] Quan Wang et al. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 2017.

[18] Vassilis N Ioannidis, Da Zheng, and George Karypis. Few-shot link prediction via graph neural networks for covid-19 drug-repurposing. *arXiv:2007.10261*, 2020.

[19] Deisy Morselli Gysi et al. Network medicine framework for identifying drug-repurposing opportunities for covid-19. *PNAS*, 2020.

[20] Amol Kapoor et al. Examining covid-19 forecasting using spatio-temporal graph neural networks. *KDD workshop*, 2020.

[21] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *KBS*, 2020.

[22] Quanming Yao et al. Taking human out of learning applications: A survey on automated machine learning. *arXiv:1810.13306*, 2018.

[23] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv:1906.02287*, 2019.

[24] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012.

[25] James Bergstra et al. Algorithms for hyper-parameter optimization. *NeurIPS*, 2011.

[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *NeurIPS*, 2012.

[27] Thomas Elsken et al. Neural architecture search: A survey. *JMLR*, 2019.

[28] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2018.

[30] Hieu Pham et al. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

[31] Barret Zoph et al. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

[32] Esteban Real et al. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

[33] Michael M Bronstein et al. Geometric deep learning: going beyond euclidean data. *IEEE SPM*, 2017.

[34] Yang Gao et al. Graph neural architecture search. In *IJCAI*, 2020.

[35] Weihua Hu et al. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS*, 2020.

[36] Chengxi Zang et al. On power law growth of social networks. *TKDE*, 2018.

[37] Justin Gilmer et al. Neural message passing for quantum chemistry. In *ICML*, 2017.

[38] Ke Tu et al. Autone: Hyperparameter optimization for massive network embedding. In *KDD*, 2019.

[39] Frank Hutter et al. Paramils: an automatic algorithm configuration framework. *JAIR*, 2009.

[40] Mengying Guo et al. Jitune: Just-in-time hyperparameter tuning for network embedding algorithms. *arXiv:2101.06427*, 2021.

[41] Yingfang Yuan et al. A novel genetic algorithm with hierarchical evaluation strategy for hyperparameter optimisation of graph neural networks. *arXiv:2101.09300*, 2021.

[42] Minji Yoon et al. Autonomous graph mining algorithm search with best speed/accuracy trade-off. In *IEEE ICDM*, 2020.

[43] Kaixiong Zhou et al. Auto-gnn: Neural architecture search of graph neural networks. *arXiv:1909.03184*, 2019.

[44] Huan Zhao, Lanning Wei, and Quanming Yao. Simplifying architecture search for graph neural network. *arXiv:2008.11652*, 2020.

[45] Yiren Zhao et al. Probabilistic dual network architecture search on graphs. *arXiv:2003.09676*, 2020.

[46] Yuhui Ding, Quanming Yao, and Tong Zhang. Propagation model search for graph neural networks. *arXiv:2010.03250*, 2020.

[47] Matheus Nunes et al. Neural architecture search in graph neural networks. In *BRACIS*, 2020.

[48] Yaoman Li and Irwin King. Autograph: Automated graph neural network. In *ICONIP*, 2020.

[49] Min Shi et al. Evolutionary architecture search for graph neural networks. *arXiv:2009.10199*, 2020.

[50] Huan Zhao et al. Efficient graph neural architecture search. *OpenReview*, 2021.

[51] Shengli Jiang and Prasanna Balaprakash. Graph neural network architecture search for molecular property prediction. In *IEEE Big Data*, 2020.

[52] Yiren Zhao et al. Learned low precision graph neural networks. *arXiv:2009.09232*, 2020.

[53] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *NeurIPS*, 2020.

[54] Guohao Li et al. Sgas: Sequential greedy architecture search. In *CVPR*, 2020.

[55] Wei Peng et al. Learning graph convolutional network for skeleton-based human action recognition by neural searching. *AAAI*, 2020.

[56] Pourchot and Sigaud. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *ICLR*, 2019.

[57] Shaofei Cai et al. Rethinking graph neural network search from message-passing. *CVPR*, 2021.

[58] Zheyi Pan et al. Autostg: Neural architecture search for predictions of spatio-temporal graphs. *WWW*, 2021.

[59] Yanxi Li et al. One-shot graph neural architecture search with dynamic search space. *AAAI*, 2021.

[60] Huan Zhao, Quanming Yao, and Weiwei Tu. Search to aggregate neighborhood for graph neural network. *ICDE*, 2021.

[61] Chaoyu Guan, Xin Wang, and Wenwu Zhu. Autoattend: Automated attention representation search. In *ICML*, 2021.

[62] Guohao Li et al. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[63] Keyulu Xu et al. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

[64] Rex Ying et al. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[65] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[66] Sirui Xie et al. Snas: stochastic neural architecture search. *ICLR*, 2019.

[67] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[68] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[69] Han Shi et al. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *NeurIPS*, 2020.

[70] Zichao Guo et al. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020.

[71] Esteban Real et al. Large-scale evolution of image classifiers. In *ICML*, 2017.

[72] Jiaxuan You et al. Graph structure of neural networks. In *ICML*, 2020.

[73] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop*, 2019.

[74] Minjie Wang et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv:1909.01315*, 2019.

[75] Peter W Battaglia et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

[76] Rong Zhu et al. Aligraph: A comprehensive graph neural network platform. *VLDB*, 2019.

[77] Alibaba. Euler: A distributed graph deep learning framework. https://github.com/alibaba/euler, 2019. [Online; accessed 25-May-2021].

[78] Adam Lerer et al. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *SysML*, 2019.

[79] CSIRO's Data61. Stellargraph machine learning library. https://github.com/stellargraph/stellargraph, 2018. [Online; accessed 25-May-2021].

[80] Daniele Grattarola and Cesare Alippi. Graph neural networks in tensorflow and keras with spektral. *ICML workshop*, 2020.

[81] Yukuo Cen et al. Cogdl: An extensive toolkit for deep learning on graphs. *arXiv:2103.00959*, 2021.

[82] Tsinghua University Natural Language Processing Lab. Openne: An open source toolkit for network embedding. https://github.com/thunlp/OpenNE, 2018. [Online; accessed 25-May-2021].

[83] Palash Goyal and Emilio Ferrara. Gem: A python package for graph embedding methods. *JOSS*, 2018.

[84] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *CIKM*, 2020.

[85] Meng Liu et al. Dig: A turnkey library for diving into graph deep learning research. *arXiv:2103.12608*, 2021.

[86] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Scipy*, 2008.

[87] Quanlu Zhang et al. Retiarii: A deep learning exploratory-training framework. In *OSDI*, 2020.

[88] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *KDD*, 2019.

[89] Matthias Feurer et al. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*. 2019.

[90] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, 2013.

[91] Randal S. Olson et al. Evaluation of a tree-based pipeline optimization tool for automating data science. In *GECCO*, 2016.

[92] Nick Erickson et al. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505*, 2020.

[93] Axel de Romblay et al. Mlbox, machine learning box. https://github.com/AxeldeRomblay/MLBox, 2018. [Online; accessed 25-May-2021].

[94] MLJAR. Mljar automated machine learning. https://github.com/mljar/mljar-supervised, 2019. [Online; accessed 25-May-2021].

[95] Chaoyu Guan et al. AutoGL: A library for automated graph learning. In *ICLR 2021 GTRL Workshop*, 2021.

[96] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab, 2008.

[97] James Bergstra et al. Algorithms for hyper-parameter optimization. In *NeurIPS*, 2011.

[98] Saining Xie et al. Exploring randomly wired neural networks for image recognition. In *ICCV*, 2019.

[99] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *ICLR*, 2018.

[100] Lukasz Dudziak et al. Brp-nas: Prediction-based nas using gcns. *NeurIPS*, 2020.

[101] Lichao Sun et al. Adversarial attack and defense on graph data: A survey. *arXiv:1812.10528*, 2018.

[102] Hao Yuan et al. Explainability in graph neural networks: A taxonomic survey. *arXiv:2012.15445*, 2020.

[103] Xin Wang et al. Explainable automated graph representation learning with hyperparameter importance. In *ICML*, 2021.

[104] Adam Auten, Matthew Tomei, and Rakesh Kumar. Hardware acceleration of graph neural networks. In *DAC*, 2020.

[105] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2018.

[106] Mingxing Tan et al. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[107] Yuhang Jiang, Xin Wang, and Wenwu Zhu. Hardware-aware transformable architecture search with efficient search space. In *ICME*, 2020.

[108] Prithviraj Sen et al. Collective classification in network data. *AI magazine*, 2008.

[109] Oleksandr Shchur et al. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[110] Vijay Prakash Dwivedi et al. Benchmarking graph neural networks. *arXiv:2003.00982*, 2020.

[111] Chris Ying et al. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, 2019.