Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 321–326
DOI: 10.15439/2025F0041
ISSN 2300-5963 ACSIS, Vol. 43

# Applying Evolutionary Techniques to Enhance Graph Convolutional Networks for Node Classification: Case Studies

Maciej Krzywda*, Szymon Łukasik*, Amir H. Gandomi†
*Faculty of Physics and Applied Computer Science, AGH University of Krakow,
al. Mickiewicza 30, 30-059 Kraków, Poland
Email: krzywda@agh.edu.pl, slukasik@agh.edu.pl
†Faculty of Engineering and IT, University of Technology Sydney,
5 Broadway, Ultimo NSW 2007, Australia
Email: gandomi@uts.edu.au
‡Systems Research Institute, Polish Academy of Sciences,
ul. Newelska 6, 01-447 Warsaw, Poland
§NASK National Research Institute,
ul. Kolska 12, 01-045 Warsaw, Poland
¶University Research and Innovation Center (EKIK), Óbuda University,
Bécsi út 96/B, Budapest, 1034, Hungary
‖Department of Computer Science, Khazar University,
Baku, Azerbaijan

***Abstract*—In recent years, significant efforts have been made to address graph node classification tasks by applying graph neural networks and methods based on label propagation. Despite the progress achieved by these approaches, their success often hinges on complex architectures and algorithms, sometimes leading to the oversight of crucial technical details. In designing artificial neural networks, one crucial aspect of the innovative approach is suggesting a novel neural architecture. Currently used architectures have mostly been developed manually by human experts, which is a time-consuming and error-prone process. That is why the adoption of more sophisticated semi-automatic methods, such as Neural Architecture Search, has become commonplace. This paper introduces and assesses an evolutionary-based approach for the design of graph convolutional neural networks in the context of node classification. Our approach aims to systematically define the graph convolutional networks parameter space, drawing inspiration from recent research on design principles. By doing so, our method seeks to strike a balance between achieving satisfactory performance and optimizing memory and computation resources, thus offering a more efficient alternative to conventional approaches from the neural architecture search area.**

***Index Terms*—graph neural networks, graph convolutional networks, genetic algorithm , node classification, evolutionary computation**

## I. Introduction

NEURAL Architecture Search (NAS) [1]–[4] is gaining popularity as a fully automatic method for designing Artificial Neural Networks Architectures (ANNs). Neural Architecture Search is a method that allows the creation of comparable and even superior architectures in terms of performance to those that have been designed manually. In essence, NAS streamlines the process of a human adjusting a neural network through trial and error to identify successful configurations and automates it to discover more intricate structures. It comprises a range of techniques and tools that evaluate numerous network architectures within a defined search space using a search strategy and choose the one that accomplishes the goals of a specific problem by maximizing a fitness function.

The manuscript is structured as follows. Firstly, Section II presents a concise overview of Neural Architecture Search approaches applied across various domains of neural networks. Following that, Section II-A provides a comprehensive summary of the state-of-the-art in Graph Neural Networks, encompassing prevalent methods employed in the design of GNN architectures. In Section III, we delve into the description of the evolutionary algorithm utilized in our research, elucidating details such as fitness metrics and the delineation of the search space. Section IV comprehensively covers settings and implementation details, including the datasets employed and the preparation of the computational budget. Section V is dedicated to a thorough discussion of the attained results, while Section VI serves as a conclusive synthesis of our research findings.

## II. Neural Architecture Search

NAS is computationally and time-consuming solution typically involving intensive calculations using graphics processing units (GPUs). Therefore, researchers and research groups are increasingly daring to look for alternative methods to optimize costs and find the most efficient and effective neural network architecture in order to solve their research problem.

Today, the most popular options for NAS are methods based on reinforcement learning [2] (including one-shot methods) [5], evolutionary computing (including multiobjective search), Bayesian optimization [6], and hill-climbing [7].

In article [4], the authors defined NAS as a system composed of three primary elements: search space, search algorithm, and evolutionary strategy.

- **Search space** - outlines a group of operations, such as convolution, pooling, and fully connected layers, and determines how these operations can be combined to create valid network architectures. Developing the search space often involves human expertise, inevitably introducing biases.
- **Search algorithm** - selects a group of candidates for network architecture and tests their performance. It receives child model performance metrics, such as accuracy and low latency, as rewards and utilizes this information to optimize and produce high-performance architecture candidates. As the possible search algorithms for NAS, one can name, e.g., algorithms like Random Search, Reinforcement Learning, and Bayesian Optimization.
- **Evaluation strategy** - in order to provide feedback for the search algorithm to learn, we must evaluate the performance of numerous proposed child models through measurement, estimation, or prediction. This process of evaluating candidates can be quite costly, prompting the proposal of various new methods to conserve time and computing resources. For these strategies, we can include proxy task performance, low-fidelity performance estimation, weight inheritance, weight sharing, learning curve extrapolation, and network morphism. [8], [9]

Based on the above definition of NAS we can define the NAS using Evolutionary Computation (EC) as a task of exploring architectures search space by generating and evaluating a population of candidate architectures and then evolving this population over multiple generations to find better-performing architectures. As we mentioned, ECs are one of the methods used to realize NAS, but the most promising results were achieved recently using the following approaches:

- **NAS-EA-FA** [10] - evolutionary algorithm based on the fitness approximation to the search for neural architecture. It is designed to accelerate the search process.
- **EF-ENAS** [11] - evolutionary neural architecture search algorithm based on evaluation corrections and fitness sharing. It is designed to improve the efficiency and effectiveness of the search process.

for a more extensive survey of evolutionary NAS one can refer to [8], [9].

### *A. Neural Architecture Search for GNNs*

Remarkably, the field of NAS within GNNs remains relatively unexplored. Notable exceptions include GraphNAS [12]–[14], which uses RL to identify architectures for node classification tasks. The search space defined by GraphNAS includes sampling, aggregation, and gated functions, demonstrating its efficacy in the realm of GNNs. Auto-GNN [15] follows suit by adopting RL and a comparable search space. SNAG framework (Simplified Neural Architecture Search for Graph Neural Networks [16]) addresses drawbacks in existing neural architecture search (NAS) methods, such as GraphNAS and Auto-GNN, by introducing an innovative search space and a reinforcement learning-based search algorithm, as demonstrated through extensive experiments on real-world datasets.

Another novel approach to NAS for GNN is Scalable Graph Neural Architecture Paradigm [17] representing a novel paradigm introduced in the paper ”PaSca.” It serves as a systematic approach to construct and explore the design space for scalable Graph Neural Networks (GNNs), aiming to address the scalability challenges inherent in traditional GNNs designed based on the neural message passing mechanism. SGAP is presented as a key element of the PaSca system, offering a principled framework for designing GNNs that can efficiently handle larger datasets and message passing steps, thus enhancing overall scalability. Neural Architecture Coding (NAC) [18] is a Neural Architecture Search (NAS) method for Graph Neural Networks (GNNs) that addresses computational cost and optimization challenges. It utilizes a sparse coding objective to find optimal architecture parameters without weight updates after random initialization. NAC claims linear-time efficiency, and empirical evaluations on GNN benchmark datasets show that it outperforms strong baselines. The method’s strength lies in leveraging the expressive power of GNNs with randomly-initialized weights for efficient architecture discovery.

[19] Automated Graph Neural Network(Auto-HeG) on Heterophilic Graphs addresses the limitations of existing graph neural architecture search (NAS) methods, which primarily focus on the homophily assumption and neglect heterophily, a crucial graph property in real-world applications. Auto-HeG aims to automatically design powerful graph neural networks with expressive learning abilities for heterophilic graphs, incorporating heterophily into various stages of the automatic heterophilic graph learning process. This includes aspects such as search space design, supernet training, and architecture selection, using a diverse message-passing scheme, progressive supernet training, and a heterophily-aware distance criterion to derive specialized and expressive heterophilic GNN architectures. The proposed method demonstrates its superiority through extensive experiments, showcasing its effectiveness compared to human-designed models and existing graph NAS models.

Auto-GNAS (Automatic Graph Neural Architecture Search) [20] addresses the challenge of constructing effective graph neural networks (GNNs) for non-euclidean data by automating the search for optimal GNN architectures. Unlike traditional methods that evaluate architectures serially, Auto-GNAS employs parallel evaluation, utilizing multiple genetic searchers simultaneously. These searchers use evaluation feedback, information entropy, and results from other searchers through a sharing mechanism to enhance efficiency. Auto-GNAS demonstrates competitive model performance and improved search efficiency compared to other algorithms, marking the first

instance of using parallel computing to improve the system efficiency of graph neural architecture search. Our approach allows for a quick and simple (computationally inexpensive) way to limit the spatial hyperparameters of a graph neural network.

## III. Evolutionary algorithm for Designing GCN

Genetic Algorithm (GA) is part of the evolutionary algorithm family, inspired by natural selection and genetics. The exploration space is a crucial element in all GAs, represented as strings called chromosomes or individuals. A population comprises a collection of such strings. Each chromosome consists of a sequence of genes encoding a solution to a target problem. Initially, a diverse population of random individuals is created to represent various potential solutions for the target problem. Each individual is assigned a fitness value that indicates its quality within the population.

During the evaluation step, the fitness of each individual is calculated. The selection step involves choosing certain individuals from the entire population as parents for mating. In the cross-step, parents exchange information, such as swapping genes, to generate new individuals and form the next generations. The mutation step introduces diversity into the population by randomly altering a gene in new individuals based on the probability of mutation. Finally, the population is updated by incorporating new individuals.

In this approach, the individual is a single representation of a graph convolutional network with a random configuration that allows it to compile and run successfully on selected datasets. The genetic algorithm used in research is the mutation and cross-over operators. Mutation involves randomly selecting a new gene value from a uniform distribution within the range of values available for that gene. The crossover function takes a list of parents as an argument, from which it randomly selects two individuals. Each individual then contributes one part of his genes (complementary to the part contributed by the other individual). Subsequently, a mutation is introduced to the obtained gene. The random selection process is repeated until the specified population size is achieved. The algorithm dynamically adjusts the number of layers based on the specified `num_layers` parameter for each agent (more params of individuals are described in Tables **??** and I, providing adaptability in the neural network architecture for each agent in the `deep_GCN` class involves iterating over the specified number of layers (`num_layers`). For every layer, the algorithm determines the specific parameters (described in TableI).

### A. Fitness

In our approach, `fitness_value` serves as a measure of the classifier quality that you aim to minimize. It is the sum of squared F1 errors for each class. The F1 scores for individual classes are numerical metrics that represent the classification quality of each class. This considers both precision and recall for a given class, making them more informative than accurate, especially in the context of unbalanced data sets. `fitness_value` is the sum of squared F1 errors across all classes. This approach provides an overall assessment of the quality of the classifier in multiple classes. Squaring the F1 error emphasizes larger errors, i.e., greater deviations from the ideal F1 score. The goal is to minimize `fitness_value`. Minimizing this value implies an effort to enhance the classifier's quality for all classes simultaneously, offering a more balanced objective than merely maximizing accuracy. Evaluating the classification quality for each class individually allows the identification and improvement of areas where the classifier may perform suboptimally.

In summary, the `fitness_value`-based approach considers more nuanced aspects of classification quality, taking into account both false positives and false negatives, as well as precision and recall. This is a more balanced approach than relying solely on accuracy, especially in scenarios involving imbalanced datasets or multiclass classification.

### B. Search space

In this section, we delve into the details of the search space employed in our approach, elucidating the ranges of various parameters within it. The search space is a crucial aspect of our methodology, and in Table I, we provide a comprehensive overview of the hyperparameter ranges relevant to the training of neural networks. It is essential to note that these ranges remain consistent across all datasets, ensuring a standardized approach to model optimization.

However, the uniqueness of each dataset requires specific considerations for certain parameters, particularly Hidden Channels and Batch Size, as explained in Table **??**. For Hidden Channels, the range is dynamically defined as range from 10 to the number of features within the dataset. This adaptability reflects the intrinsic characteristics of the data and aims to optimize the model architecture accordingly. Similarly, for the batch size parameter, the range ranges from a minimum of 10 to the total count of all training masks present in the dataset, as detailed in Table II.

To provide a concise reference, Table I encapsulates the overarching hyperparameter ranges for training neural networks, emphasizing their uniform applicability across diverse datasets. The following table **??** then details the data set-specific nuances, delineating the distinct ranges for Hidden Channels and Batch Size. This approach ensures that our search space is not only comprehensive but also adaptive to the intricacies of individual datasets, fostering effective model optimization.

## IV. Experimental Settings

### A. Datasets

In Table II, we provide a comprehensive overview of the key parameters associated with the datasets utilized in our experimental efforts. Brief characteristics of each dataset are described below.

- **Planetoid Datasets**, including Cora, CiteSeer, and PubMed, are widely used in graph-based learning tasks.

TABLE I: Neural Network Hyperparameter Ranges

| Parameter | Range |
|---|---|
| **Epochs** | 100 |
| **Learning Rate** | From 0.0001 to 0.1 |
| **Weight Decay** | From 0.0001 to 0.1 |
| **Layers** | From 1 to 10 |
| **Optimizers** | Adadelta, Adagrad, Adam, AdamW, Adamax, ASGD, NAdam, RAdam, RMSprop, Rprop, SGD, QHM, QHAdam, PSO, A2GradExp, A2GradInc, A2GradUni, AccSGD, AdaBelief, AdaBound, AdaMod, Adafactor, Adahessian, AdamP, AggMo, Apollo, DiffGrad, Lamb, MADGRAD, NovoGrad, PID, Ranger, RangerQH, RangerVA, SGDP, SGDW, SWATS, Shampoo, Yogi |
| **Loss Function** | CrossEntropy, NLLLoss, MultiMarginLoss, MultiLabelMarginLoss, SmoothL1Loss |

They feature academic citation networks where nodes represent documents, and edges denote citations.

- **WikipediaNetwork Datasets**, comprising Squirrel and Chameleon, focus on analyzing relationships within Wikipedia. For example, Squirrel captures the network structure of Wikipedia articles, enabling researchers to explore connections and interactions between different topics.
- **WebKB Datasets**, such as Cornell, Texas, and Wisconsin, are tailored for web page classification. These datasets encompass web pages from university domains and are labeled with categories such as student, faculty, project, and course. They serve as benchmarks for text classification and information retrieval tasks.
- **Actor Dataset** is commonly utilized in social network analysis, representing a network of actors and their collaborations in movies or television shows. The nodes correspond to actors, and the edges represent collaborations in specific projects.

The selection of these datasets is grounded in their prevalence in academic research and practical applications. Researchers and practitioners often turn to these datasets as benchmarks due to their well-defined structures, facilitating the rigorous evaluation of diverse graph-based machine learning algorithms.

### *B. Computational Settings*

We have defined a computational budget for our evolutionary algorithm with the following specifications:

- 300 generations
- 100 training epochs
- 10 initial populations (random)
- whole population is 20
- mutation rate is 0.3

The search space is 8-dimensional, denoted as **batch size**,**hidden channels**, **learning rate**, **weight decay**, **num layer**, **optimizer**, **loss function**. In particular, **the epoch number** is held constant throughout the exploration. Using the previously defined **fitness value** enables us to conduct innovative research within this fixed dimensionality.

## V. RESULTS

We observed the mean fitness and accuracy for each generation and, for better visualization, we also tracked the `best fitness` and `best accuracy` achieved throughout the evolutionary process. While analysing the results enclosed in Table II and III, it can be noticed that minimizing the cost in multi-class classification problems is more complex than in the case of binary classification, which can also be observed in our case - the more complex the dataset, the potentially more difficult it is to prepare a model that will allow for the correct classification of all nodes from high effectiveness. Our research shows that using only GCN is not an ideal solution for datasets such as Actor, Squirrel or Chameleon, which are more complex datasets, e.g. heterophily ratio, characterized by more Nodes, average node Degree, or Node Features (in Table II) the less accurate the classification, which can be observed in the table.

In Table III, we present the optimal genes, which represent the best solutions obtained through optimization using the genetic algorithm.

These adjustments ensure that the experimental parameters adhere to the requirement of being integer values. Although presented in the table as floating values for clarity, they are effectively treated as integers during the genetic algorithm optimization process. This approach maintains consistency and allows for a meaningful and applicable exploration of the solution space.

Analysis of the experimental results reveals key patterns in hyperparameter configurations for Graph Convolutional Networks (GCN) across various datasets (presented in III. The most prevalent configuration associated with optimal performance includes the Adamax optimizer, the cross-entropy loss function, and 1 to 3 layers of GCN. In particular, configurations with a single or two layers of GCN emerged more frequently, suggesting their effectiveness in achieving high fitness values and classification accuracy. These findings offer practical guidance for researchers and practitioners seeking robust hyperparameter choices for GCN models [21], [22] for Node Classification.

TABLE II: Dataset Characteristics

| **Parameter** | **Wisconsin** | **Actor** | **Texas** | **Squirrel** | **PubMed** | **Cornell** | **Cora** | **CiteSeer** | **Chameleon** |
|---|---|---|---|---|---|---|---|---|---|
| Nodes | 251 | 7600 | 183 | 5201 | 19717 | 183 | 2708 | 3327 | 2277 |
| Edges | 515 | 30019 | 325 | 217073 | 88648 | 298 | 10556 | 9104 | 36101 |
| Avg. Node Degree | 2.05 | 3.95 | 1.78 | 41.74 | 4.50 | 1.63 | 3.90 | 2.74 | 15.85 |
| Training Nodes | 1200 | 36480 | 870 | 24960 | 60 | 870 | 140 | 120 | 10920 |
| Isolated Nodes | False | False | False | False | False | False | False | True | False |
| Self-Loops | True | True | True | True | False | True | False | False | True |
| Is Undirected | False | False | False | False | True | False | True | True | False |
| Features | 1703 | 932 | 1703 | 2089 | 500 | 1703 | 1433 | 3703 | 2325 |
| Classes | 5 | 5 | 5 | 5 | 3 | 5 | 7 | 6 | 5 |
| Node Features | 1703 | 932 | 1703 | 2089 | 500 | 1703 | 1433 | 3703 | 2325 |
| Homophily ratio (h) | 0.21 | 0.22 | 0.11 | 0.22 | 0.8 | 0.3 | 0.81 | 0.74 | 0.23 |

TABLE III: Best results achieved for each dataset

| Param | Wisconsin | Texas | Squirrel | PubMed | Cornell | Cora | CiteSeer | Chameleon | Actor |
|---|---|---|---|---|---|---|---|---|---|
| Batch Size | 630 | 500 | 837 | 280 | 210 | 605 | 358 | 509 | 87 |
| Hidden Channels | 267 | 1633 | 1384 | 338 | 348 | 1245 | 230 | 623 | 1647 |
| Epochs | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| Learning Rate | 0.0087 | 0.0118 | 0.0143 | 0.0848 | 0.0073 | 0.0022 | 0.0054 | 0.0826 | 0.0674 |
| Weight Decay | 0.0312 | 0.0023 | 0.0169 | 0.0019 | 0.0011 | 0.0009 | 0.0083 | 0.0039 | 0.0151 |
| Layers | 1 | 1 | 1 | 3 | 2 | 4 | 2 | 4 | 1 |
| Loss Function | SmoothL1Loss | Cross Entropy | Cross Entropy | Cross Entropy | Cross Entropy | Cross Entropy | Cross Entropy | Cross Entropy | Cross Entropy |
| Optimizer | Adam | NAdam | Adadelta | Adam | Adadelta | Adadelta | PSO | Adam | Adagrad |
| Minimum Fitness | 1.4624 | 1.7088 | 1.6277 | 0.3627 | 1.4322 | 0.4845 | 0.8082 | 1.2427 | 1.6963 |
| Accuracy | 0.5882 | 0.6486 | 0.3121 | 0.7960 | 0.5135 | 0.8290 | 0.7320 | 0.4496 | 0.2539 |

It is surprising that NAdam and QHAdam [23] along with Particle Swarm Optimization [24] are algorithms that have achieved high Accuracy values and low fitness. Moreover, the dominance of Adagrad over the most popular Adam is noteworthy. Due to fewer options for loss functions, Categorical Cross Entropy dominates here and possibly achieves better results for other loss functions, which are not included in our research. We compare our methods with baselines methods [25], [26], which are presented in Table IV.

Based on the results presented above in Table IV, we observe that the results achieved for GCN based on the most popular implementation of GCN [26] are very close to the best accuracy. Our solution enhances the standard GCN models, though it does not quite reach the level of the most advanced ones [27], [28]. Nevertheless, it can serve as an inspiring starting point for future research in this field. Homophily ratio [29] in datasets with high values (Actor, Wisconsin, Squrrel) refers to the tendency of nodes with similar features or attributes to be more likely connected to each other. In datasets with a high homophily ratio (CiteSeer, Cora, PubMed), the relationships between similar nodes become more pronounced, potentially improving classification performance in tasks that rely on node similarities.

## VI. Conlusions

In the last few years, studies on designing Artificial Neural Networks (ANNs) have become an active research field, mainly due to the advanced cost training and prototyping of underlying deep learning architectures. The proposed study focuses on developing the first stage of a method for the design and optimization of graph-convolutional neural networks.Our method enables avoiding mindless reliance on closed-box approaches, where one often waits for the final outcome without considering what transpires throughout the entire process. Assuming that the parameter search space of a graph convolutional neural network is multidimensional (each dimension defined by a network parameter), we conclude that by increasing the number of dimensions and their size, such as utilizing various types of layers (not just GCNConv, as in our study, but also GATConv, GIN, SGCNN etc.), we can construct the best possible neural network. This is achieved without the need to specify in advance whether our network should be GCN or GAT-based. Our study provides answers that GCN is not always the ideal solution. By extending our proposed mechanism, we can achieve an ideal architecture for solving a specific node classification problem as a result, our method allows for flexibility and adaptation to the specific requirements of a given situation.Plans for this research include the research of another search optimization framework based on Genetic Programming (including Cartesian Genetic Programming, the utility of which was previously demonstrated by designing CNNs [30]), Evolutionary Strategies, and Bio-Inspired optimization methods [31]. This approach will allow greater flexibility in designing and optimizing not only graph coefficient neural networks but all available Graph Neural Networks (GAT, GIN, Graph Transformer etc.). We anticipate that the results of this study might be useful to researchers in the field of Neural Architecture Search because the results obtained in the experiment are very promising.

TABLE IV: Comparison of accuracy of our approaches with state-of-the-art methods and GNN baselines, and performance of different models on various datasets

| Method | Wisconsin | Texas | Squirrel | PubMed | Cornell | Cora | CiteSeer | Chameleon | Actor |
|---|---|---|---|---|---|---|---|---|---|
| GCN [26] | 45.88 | 52.16 | 23.96 | 88.13 | 52.70 | 85.77 | 73.68 | 28.18 | 26.86 |
| GAT [25] | 49.41 | 58.38 | 30.03 | 87.62 | 54.32 | 86.37 | 74.32 | 42.93 | 28.45 |
| Geom-GCN-P [27] | 64.12 | 67.57 | 38.14 | 88.09 | 60.81 | 84.93 | 75.14 | 60.90 | 31.63 |
| GGCN [28] | 86.86 | 84.86 | 55.17 | 89.15 | 85.68 | 87.95 | 77.14 | 71.14 | 37.54 |
| Our | 58.82 | 64.86 | 31.21 | 79.60 | 51.35 | 82.90 | 73.20 | 44.96 | 25.39 |

## ACKNOWLEDGMENTS

The work was supported by statutory tasks of the AGH UST Faculty of Physics and Applied Computer Science within the MEiN grant. We gratefully acknowledge the Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2022/015677 and no. PLG/2023/016643.

## REFERENCES

[1] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. P. Murphy, "Progressive neural architecture search," in *European Conference on Computer Vision*, 2017.

[2] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *ArXiv*, vol. abs/1611.01578, 2016.

[3] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International conference on machine learning*. PMLR, 2018, pp. 4095–4104.

[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *ArXiv*, vol. abs/1808.05377, 2018.

[5] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*, 2018.

[6] C. White, W. Neiswanger, and Y. Savani, "Bananas: Bayesian optimization with neural architectures for neural architecture search," in *AAAI Conference on Artificial Intelligence*, 2019.

[7] M. Verma, P. P. Sinha, K. Goyal, A. Verma, and S. Susan, "A novel framework for neural architecture search in the hill climbing domain," *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 1–8, 2019.

[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, pp. 550–570, 2020.

[9] X. Zhou, A. K. Qin, Y. Sun, and K. C. Tan, "A survey of advances in evolutionary neural architecture search," *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 950–957, 2021.

[10] C. Pan and X. Yao, "Neural architecture search based on evolutionary algorithms with fitness approximation," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[11] R. Shang, S. Zhu, J. Ren, H. Liu, and L. Jiao, "Evolutionary neural architecture search based on evaluation correction and functional units," *Knowledge-Based Systems*, vol. 251, p. 109206, 2022.

[12] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *International Joint Conference on Artificial Intelligence*, 2020.

[13] ——, "Graphnas: Graph neural architecture search with reinforcement learning," *ArXiv*, vol. abs/1904.09981, 2019.

[14] Y. Gao, P. Zhang, H. Yang, C. Zhou, Z. Tian, Y. Hu, Z. Li, and J. Zhou, "Graphnas++: Distributed architecture search for graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 6973–6987, 2023.

[15] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *Frontiers in Big Data*, vol. 5, 2019.

[16] H. Zhao, L. Wei, and Q. Yao, "Simplifying architecture search for graph neural network," *ArXiv*, vol. abs/2008.11652, 2020.

[17] W. Zhang, Y. Shen, Z. Lin, Y. Li, X. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, "Pasca: A graph neural architecture search system under the scalable paradigm," *Proceedings of the ACM Web Conference 2022*, 2022.

[18] P. Xu, L. Zhang, X. Liu, J. Sun, Y. Zhao, H. Yang, and B. Yu, "Do not train it: A linear neural architecture search of graph neural networks," *ArXiv*, vol. abs/2305.14065, 2023.

[19] X. Zheng, M. Zhang, C. cheng Jason Chen, Q. Zhang, C. Zhou, and S. Pan, "Auto-heg: Automated graph neural network on heterophilic graphs," *Proceedings of the ACM Web Conference 2023*, 2023.

[20] J. Chen, J. Gao, Y. Chen, B. M. Oloulade, T. Lyu, and Z. Li, "Auto-gnas: A parallel graph neural architecture search framework," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, pp. 1–1, 2022.

[21] X. Miao, W. Zhang, Y. Shao, B. Cui, L. Chen, C. Zhang, and J. Jiang, "Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture (extended abstract)," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 1561–1562.

[22] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *ArXiv*, vol. abs/1811.05868, 2018.

[23] J. Ma and D. Yarats, "Quasi-hyperbolic momentum and adam for deep learning," *arXiv preprint arXiv:1810.06801*, 2018.

[24] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft computing*, vol. 22, pp. 387–408, 2018.

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv*, 2018.

[26] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ArXiv*, vol. abs/1609.02907, 2017.

[27] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," 2020.

[28] Y. Yan, M. Hashemi, K. Swersky, Y. Yang, and D. Koutra, "Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks," *2022 IEEE International Conference on Data Mining (ICDM)*, pp. 1287–1292, 2021.

[29] W. Huang, X. Guan, and D. Liu, "Revisiting homophily ratio: A relation-aware graph neural network for homophily and heterophily," *Electronics*, vol. 12, no. 4, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/4/1017

[30] M. Krzywda, S. Łukasik, and A. H. Gandomi, "Cartesian genetic programming approach for designing convolutional neural networks," *arXiv preprint arXiv:2410.00129*, 2024.

[31] R. Shen, A. S. Bosman, A. Schreuder, M. Krzywda, and S. Łukasik, "Training graph neural networks with particle swarm optimisation," *Sacair 2023*, 2023.