Review of Meta-heuristics Methods for Machine Learning With Graph Neural Network

Andrea Maldini *

Posted Date: 3 October 2024

doi: 10.20944/preprints202410.0192.v1

Keywords: supervised learning; machine learning; heuristics; graph neural network; variable neighborhood search; genetic algorithm; particle swarm optimization

 doi:10.20944/preprints202410.0192.v1

Article

Review of Meta-Heuristics Methods for Machine Learning with Graph Neural Network

Andrea Maldini

andreakmaldini@outlook.com

Abstract: In this paper, we review metaheuristic optimization techniques for machine learning with Graph Neural Networks (GNNs), focusing on two major categories: single-solution-based and population-based metaheuristics. We provide an in-depth analysis of Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), highlighting their effectiveness in optimizing GNN hyperparameters. By exploring the strengths and weaknesses of these methods, we aim to provide insights into their application and performance in enhancing the efficiency and accuracy of GNN models. We will also review the similarities of supervised learning and metaheuristic solutions.

Keywords: supervised learning; machine learning; heuristics; graph neural network; variable neighborhood search; genetic algorithm; particle swarm optimization

1. Introduction

We are interested in optimal or near optimal solutions using metaheuristics. Among all machine learning methodologies, Graph Neural Networks (GNNs) have emerged as a prominent method for processing graph-structured data, which is prevalent in various domains such as social networks, recommendation systems, and molecular biology. Traditional neural network models, such as convolutional neural networks (CNNs), fail to adequately capture the irregular structure of graphs. GNNs, on the other hand, excel in handling non-Euclidean data structures, offering unique capabilities in node classification, link prediction, and graph classification tasks. However, the training and optimization of GNNs remain computationally intensive and prone to challenges like overfitting and local optima in large-scale and complex graphs. Metaheuristic optimization techniques, which offer robust solutions for high-dimensional, non-convex, and noisy search spaces, have gained traction as viable methods to improve the performance and efficiency of GNNs. It is more like a supervised learning methodology. This paper explores the integration of metaheuristic solution methods with GNNs, investigating how these techniques can be leveraged to enhance the training and optimization of GNNs.

2. Review of Literature

[1] provide a comprehensive review of how machine learning, particularly metaheuristics, can solve complex combinatorial optimization problems. They delve into techniques like evolutionary algorithms, swarm intelligence, and hybrid systems, emphasizing the role of reinforcement learning in guiding metaheuristic searches for better solutions in neural networks (e.g., GNNs). [2] explore the optimization of deep belief networks and GNNs using evolutionary hyper-heuristics, which combine different metaheuristic strategies to tune hyperparameters efficiently. Their work demonstrates how metaheuristics can significantly improve neural network performance in specific tasks, such as image classification. The work of [3] discusses implementing metaheuristics in extreme learning machines and GNNs, particularly focusing on optimizing neural architecture and weight connections. This work shows how metaheuristics such as genetic algorithms can effectively fine-tune architectures for tasks like traffic control and wireless communications.

A GNN is a type of neural network specifically designed to operate on graph structures. A graph $G = (V, E)$ consists of a set of vertices V and edges E, where each vertex $v \in V$ can represent a relationship between two entities. The fundamental idea behind GNNs is to iteratively aggregate information from neighboring nodes, thereby learning node embeddings that capture both the local and global structure of the graph.

 doi:10.20944/preprints202410.0192.v1

GNNs typically use message-passing mechanisms to update node embeddings. For a node v, its updated embedding is computed by aggregating the embeddings of its neighboring nodes $N(v)$. This can be formally represented as in Eq 1:

$$h_v^{k+1} = AGGREGATE(h_v^{(k)}, \{h_u^{(k)} | u \in N(v)\}) \quad (1)$$

where $h_v^{(k)} is the embedding of node v$ at iteration k, and $AGGREGATE$ is a function that combines information from the node's neighbors.

Despite their strengths, GNNs face several optimization challenges. The parameters of GNNs are typically learned using gradient-based optimization methods like stochastic gradient descent (SGD) or Adam. These methods, while effective, can sometimes get stuck in local optima, struggle with saddle points, or suffer from vanishing gradients, especially in deep architectures. Moreover, GNNs require tuning of hyperparameters such as learning rates, weight decay, and network depth, which are traditionally optimized through grid search or random search. These methods are computationally expensive and may not guarantee global optima. Metaheuristics, which are optimization techniques inspired by natural processes and phenomena, have shown potential in addressing these challenges.

Metaheuristic algorithms are high-level procedures designed to generate or select heuristics that provide good enough solutions for optimization problems. Unlike traditional optimization methods, which often rely on gradients or second-order information, metaheuristics use mechanisms such as exploration and exploitation to search through the solution space. Metaheuristics are generally divided into two categories: single-solution-based and population-based algorithms.

Single-solution-based metaheuristics

improve a single candidate solution iteratively. Some of the popular single-solution metaheuristics include:

- Simulated Annealing (SA): Inspired by the annealing process in metallurgy, SA explores the solution space by accepting worse solutions with a probability that decreases over time [4]. This helps the algorithm avoid local optima and encourages exploration of the search space.
- Tabu Search (TS): TS utilizes a memory structure (tabu list) to avoid revisiting recently explored solutions [5]. This encourages the algorithm to explore new areas of the solution space, avoiding cycles [6].
- Variable Neighborhood Search (VNS): VNS systematically changes the neighborhood structure during the search process, allowing the algorithm to escape local optima by exploring progressively larger neighborhoods [7].

Population-based metaheuristics

maintain a population of candidate solutions and improve them over successive iterations. Some popular population-based metaheuristics include:

- Genetic Algorithms (GA): GA mimics the process of natural evolution, where solutions are represented as chromosomes and undergo crossover and mutation operations to produce better offspring. Selection mechanisms favor the survival of better solutions. [8] presented this algorithm that break the computation constraint at the time of annoucement.
- Particle Swarm Optimization (PSO): PSO models the behavior of swarms, such as birds or fish, to optimize a solution [9]. Particles (candidate solutions) move through the solution space by updating their velocity based on their own experience and that of their neighbors.
- Ant Colony Optimization (ACO): ACO is inspired by the behavior of ants in finding the shortest paths to food [10]. Artificial ants construct solutions incrementally based on pheromone trails, which guide future search efforts. ACO later evolves with more efficient solutions in [11].

With what demostrated in [12], Graph Neural Networks (GNNs) are a class of deep learning models designed to work with graph-structured data. Unlike traditional neural networks, which

 doi:10.20944/preprints202410.0192.v1

handle Euclidean data like images or text, GNNs can model relationships and interactions between data points represented as nodes and edges. This structure allows GNNs to capture complex dependencies and perform tasks such as node classification, link prediction, and graph classification. GNNs have shown significant success in domains where relational data is critical, such as social networks, molecule analysis, and recommendation systems. The key strength of GNNs lies in their ability to aggregate information from neighboring nodes through multiple layers, effectively learning rich node and graph-level representations. We see a potential synergy between GNN and metaheuristics.

3. Integrating Metaheuristics with GNNs

3.1. Hyperparameter Optimization

Hyperparameter optimization is crucial for GNN performance. Metaheuristics have been effectively applied to optimize hyperparameters such as learning rate, number of layers, dropout rate, and weight decay. Traditional grid search and random search approaches suffer from the curse of dimensionality and are computationally expensive. Metaheuristics, such as GA and PSO, offer a more efficient approach by intelligently exploring the hyperparameter space.

For instance, genetic algorithms can be applied to optimize the learning rate and the number of GNN layers simultaneously. Chromosomes can represent the hyperparameter settings, and the fitness function can be the performance of the GNN on a validation set. Crossover and mutation operations allow exploration of new hyperparameter combinations, while selection favors high-performing configurations.

Particle Swarm Optimization (PSO) is another method used for hyperparameter optimization. Each particle in the swarm represents a candidate set of hyperparameters, and particles adjust their positions based on their own past performance and that of neighboring particles. This enables PSO to effectively balance exploration and exploitation in the hyperparameter space.

3.2. Architecture Search

In addition to hyperparameters, the architecture of GNNs—such as the number of hidden units per layer, the choice of activation functions, and the aggregation functions—can significantly impact their performance. Metaheuristics have been used to automate the search for optimal architectures, a process known as neural architecture search (NAS). Metaheuristic-based NAS avoids the exhaustive search characteristic of traditional NAS approaches.

Simulated annealing, for example, has been employed for GNN architecture search. By exploring different GNN architectures and accepting suboptimal architectures at the beginning of the search, SA avoids premature convergence. As the temperature decreases, the search becomes more exploitative, homing in on a locally optimal architecture.

Ant Colony Optimization (ACO) can also be applied to architecture search, where artificial ants construct different GNN architectures incrementally, guided by a fitness function that evaluates the performance of the architecture [11]. The pheromone trails guide future ants towards more promising architectural choices, allowing the exploration of novel GNN configurations.

3.3. Training Optimization

Beyond hyperparameter and architecture optimization, metaheuristics can be directly integrated into the training process of GNNs. Training optimization involves finding optimal weights and biases for the GNN during the learning process. While gradient-based methods such as SGD are commonly used, metaheuristics offer a gradient-free alternative that can avoid some of the pitfalls of gradient-based methods, such as getting stuck in local minima or suffering from vanishing gradients.

Genetic algorithms can be applied to optimize GNN weights by treating each set of weights as a chromosome and using evolutionary operators to search for better solutions. The fitness function can be the GNN's performance on a validation set. One advantage of GA is that it can effectively explore the weight space without relying on gradients, making it robust to non-differentiable loss functions.

 doi:10.20944/preprints202410.0192.v1

Simulated Annealing (SA) has also been applied to the training of GNNs. In this case, the GNN's weights are adjusted by perturbing them in a manner similar to gradient-based updates, but with the acceptance of worse solutions at the beginning of the training process to avoid local optima.

3.4. Comparative Analysis

The integration of metaheuristics into GNNs has been explored through various empirical studies. In hyperparameter optimization, metaheuristics such as GA and PSO consistently outperform traditional methods like grid search in terms of both efficiency and final performance. The ability of metaheuristics to balance exploration and exploitation allows them to discover better hyperparameter settings in fewer iterations.

For architecture search, metaheuristics offer a compelling alternative to brute-force methods. Studies show that metaheuristic-based NAS methods can discover architectures that perform competitively with manually designed architectures, while requiring significantly less computational resources.

In training optimization, the use of genetic algorithms and simulated annealing has been shown to improve the convergence speed of GNNs, especially for large and complex graphs. Moreover, metaheuristic-based training methods demonstrate robustness in avoiding local minima, leading to better generalization performance.

However, metaheuristics are not without limitations. They can be computationally expensive, particularly when applied to large-scale problems with many parameters. Moreover, the success of a metaheuristic algorithm often depends on careful tuning of its own parameters, such as population size in GA or cooling schedule in SA, which can offset some of the efficiency gains.

4. Genetic Algorithms (GA) for Hyperparameter Optimization

Genetic Algorithms (GA) are inspired by the process of natural evolution, using concepts such as selection, crossover, and mutation to evolve a population of candidate solutions over several generations. In the context of hyperparameter optimization for GNNs, the candidate solutions are different sets of hyperparameters, and the goal is to evolve these hyperparameter sets to optimize the performance of the GNN. [13] and [14] presented frameworks on how genertic algorithm work together with GNN. In general, GA in Hyperparameter Optimization goes on the following steps:

Chromosome Representation

: Each solution (or individual) is represented as a chromosome, where each gene in the chromosome corresponds to a specific hyperparameter. For example, a chromosome could look like:

$$[learningrate, numberoflayers, dropoutrate, weightdecay] \tag{2}$$

For instance, a chromosome might look like:

$$[0.001, 3, 0.5, 0.01] \tag{3}$$

Here, the genes represent the learning rate (0.001), the number of layers (3), the dropout rate (0.5), and the weight decay (0.01).

Population Initialization

: A population of chromosomes (candidate hyperparameter sets) is initialized either randomly or using some prior knowledge. The size of the population is a tunable parameter, typically set based on the computational resources available.

 doi:10.20944/preprints202410.0192.v1

Fitness Function

: Each chromosome's fitness is evaluated by training the GNN with the corresponding set of hyperparameters and measuring its performance on a validation dataset. The fitness function could be the accuracy, F1 score, or another performance metric depending on the problem.

- Selection: Individuals (chromosomes) with better fitness are more likely to be selected for reproduction. Various selection strategies can be used, such as tournament selection, roulette wheel selection, or rank-based selection, ensuring that high-performing chromosomes have a greater chance of producing offspring.
- Crossover (Recombination): Pairs of selected chromosomes are combined to create offspring through a process called crossover. Crossover involves swapping genes between parent chromosomes, allowing the offspring to inherit characteristics from both parents. For example, if the learning rate gene is swapped between two parents, the offspring might inherit a good learning rate from one parent and an optimal number of layers from the other parent.
- Mutation: After crossover, random mutations are introduced to some genes in the offspring. Mutation helps maintain genetic diversity in the population and avoids premature convergence to suboptimal solutions. For example, the learning rate in a chromosome might be slightly perturbed to explore new areas of the hyperparameter space.
- Replacement: The new offspring replace some or all of the existing population, and the process continues over several generations. After a predefined number of generations or if a stopping criterion is met (such as no improvement in fitness), the best chromosome (hyperparameter set) is returned as the optimal solution.

4.1. Advantages of GA in Hyperparameter Optimization

Exploration-Exploitation Balance

: GA strikes a balance between exploration (diversifying the search space through crossover and mutation) and exploitation (selecting high-performing solutions through fitness-based selection).

Parallelization

: GA can be easily parallelized since the evaluation of each chromosome (hyperparameter set) is independent, allowing for faster optimization when computational resources are available.

Scalabilit

: GA is effective in exploring large, complex hyperparameter spaces, which are common in deep learning applications.

5. Particle Swarm Optimization (PSO) for Hyperparameter Optimization

Particle Swarm Optimization (PSO) is a population-based optimization technique inspired by the social behavior of birds flocking or fish schooling. In PSO, a population of candidate solutions (called particles) moves through the search space to find the optimal solution. Each particle adjusts its trajectory based on its own experience and the experience of its neighbors. We borrow the idea from [15] and [16]. The basic steps for PSO in Hyperparameter Optimization is:

Particle Representation

: In PSO, each particle represents a set of hyperparameters for the GNN, similar to how chromosomes represent solutions in GA. For instance, a particle might be:

$$[0.001, 3, 0.5, 0.01] \tag{4}$$

where each dimension represents a specific hyperparameter.

 doi:10.20944/preprints202410.0192.v1

Initialization

: A swarm of particles is initialized randomly in the hyperparameter space. Each particle has a position (its current set of hyperparameters) and a velocity (which controls how its position changes).

Fitness Evaluation

: The fitness of each particle is evaluated by training the GNN with the corresponding hyperparameter set and measuring its performance on a validation dataset. The fitness function in PSO is the same as in GA, typically based on the GNN's validation accuracy or another metric.

Velocity and Position Update

: Each particle updates its velocity based on three components:

- Inertia: The particle's previous velocity, which encourages it to maintain its current trajectory.
- Cognitive Component: The difference between the particle's current position and the best position it has encountered so far (personal best).
- Social Component: The difference between the particle's current position and the best position found by any particle in the swarm (global best). The update rule for velocity can be written as:

$$v_i^{t+1} = wv_i^t + c_1 r_1 (p_i^t - x_i^t) + c_2 r_2 (g^t - x_i^t) \tag{5}$$

where:

1. v_i^t is the velocity of particle i at iteration t,
2. x_i^t is the current position of particle i,
3. p_i^t is the personal best position of particle i,
4. g^t is the global best position,
5. w is the inertia weight,
6. c_1 and c_2 are acceleration coefficients,
7. r_1 and r_2 are random numbers between 0 and 1.

The position of each particle is then updated as:

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{6}$$

This process allows the particles to explore the search space, with velocity updates helping them converge toward promising areas.

- Personal and Global Best Updates: Each particle keeps track of its personal best solution (the best hyperparameter set it has encountered), and the swarm as a whole keeps track of the global best solution (the best hyperparameter set encountered by any particle).
- Stopping Criterion: The PSO algorithm continues iterating, updating the particles' velocities and positions until a stopping criterion is met, such as a predefined number of iterations or no improvement in the global best solution.

5.1. Advantages of PSO in Hyperparameter Optimization

Convergence

: PSO is known for its fast convergence, especially in continuous search spaces, making it well-suited for hyperparameter optimization in GNNs.

Simple Implementation

: PSO is relatively easy to implement compared to GA, as it does not require crossover or mutation operators.

 doi:10.20944/preprints202410.0192.v1

Efficient Exploration

: By balancing exploration and exploitation, PSO efficiently navigates the hyperparameter space, adapting based on both individual and collective experience of the particles.

Fewer Hyperparameters

: PSO typically requires fewer parameters to tune (e.g., inertia weight, acceleration coefficients) compared to GA, which requires tuning of population size, crossover rates, mutation rates, etc.

5.2. Comparative Analysis of GA and PSO for Hyperparameter Optimization

Exploration vs. Exploitation

: GA tends to focus more on exploration due to the crossover and mutation operators, making it well-suited for exploring highly complex, non-convex spaces. PSO, on the other hand, tends to converge faster and is better at fine-tuning solutions through exploitation, especially in continuous hyperparameter spaces.

Computational Efficiency

: PSO is generally more computationally efficient since it does not require population-wide recombination operations like crossover. However, both methods can be parallelized to speed up the fitness evaluation step.

Convergence Behavior

: PSO often converges faster than GA due to the direct influence of global best knowledge. However, GA might have a better chance of escaping local optima due to the diversity introduced by mutation.

6. Conclusions

In this paper, we mainly review the single solution based metaheuristics and population based metaheuristics for GNN. Among those methods, we particularly look at genetic algorithm and Particle Swarm Optimization for hyperparameter optimization. The application of metaheuristic solution methods to GNNs represents a promising direction for improving the training and optimization of these models. Metaheuristics offer an alternative to traditional gradient-based methods, providing robust optimization techniques that can handle non-convex, high-dimensional search spaces. The integration of metaheuristics into hyperparameter optimization, architecture search, and training optimization has demonstrated significant improvements in the performance and efficiency of GNNs across various domains. Supervised learning and metaheuristics have a lot of synergies that research can tap on.

References

1. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; et al.. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: a state-of-the-art. *European Journal of Operational Research* **2022**, *296*, 393–422.
2. Sabar, N.; Turky, A.; Song, A.; et al.. An evolutionary hyper-heuristic to optimise deep belief networks. 2017 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2017, pp. 2738–2745.
3. Escobar, H.; Cuevas, E. *Implementation of Metaheuristics with Extreme Learning Machines*; Springer, 2020; pp. 125–147.
4. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. doi:10.1126/science.220.4598.671.
5. Glover, F. Tabu Search—Part I. *ORSA Journal on Computing* **1989**, *1*, 190–206. doi:10.1287/ijoc.1.3.190.
6. Glover, F. Tabu Search—Part II. *ORSA Journal on Computing* **1990**, *2*, 4–32. doi:10.1287/ijoc.2.1.4.

7. Mladenovic, N.; Hansen, P. Variable Neighborhood Search. *Computers & Operations Research* **1997**, *24*, 1097–1100. doi:10.1016/S0305-0548(97)00031-2.
8. Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley, 1989.
9. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. Proceedings of ICNN'95 - International Conference on Neural Networks. IEEE, 1995, Vol. 4, pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
10. Dorigo, M.; Gambardella, L.M. Ant Colonies for the Traveling Salesman Problem. *Biosystems* **1997**, *43*, 73–81. doi:10.1016/S0303-2647(97)01708-5.
11. Dorigo, M.; Birattari, M.; Stützle, T. Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. *IEEE Computational Intelligence Magazine* **2006**, *1*, 28–39. doi:10.1109/MCI.2006.329691.
12. Wang, Z.; Zhu, Y.; Li, Z.; Wang, Z.; Qin, H.; Liu, X. Graph neural network recommendation system for football formation. *Applied Science and Biotechnology Journal for Advanced Research* **2024**, *3*, 33–39.
13. Shi, M.; Tang, Y.; Zhu, X.; Huang, Y.; Wilson, D.; Zhuang, Y.; Liu, J. Genetic-gnn: Evolutionary architecture search for graph neural networks. *Knowledge-based systems* **2022**, *247*, 108752.
14. Yuan, Y.; Wang, W.; Coghill, G.M.; Pang, W. A novel genetic algorithm with hierarchical evaluation strategy for hyperparameter optimisation of graph neural networks. *arXiv preprint arXiv:2101.09300* **2021**.
15. Shen, R.; Bosman, A.S.; Schreuder, A.; Krzywda, M.; Łukasik, S. Training Graph Neural Networks with Particle Swarm Optimisation. *Sacair 2023* **2023**.
16. Vodilovska, V.; Gievska, S.; Ivanoska, I. Hyperparameter Optimization of Graph Neural Networks for mRNA Degradation Prediction. 2023 46th MIPRO ICT and Electronics Convention (MIPRO). IEEE, 2023, pp. 423–428.