Article

Not peer-reviewed version

Single Solution Based Metaheuristics With Graph Network Based Machine Learning

Mia Cryzan *

Posted Date: 3 October 2024

doi: 10.20944/preprints202410.0187.v1

Keywords: Tabu Search; Simulated Annealing; Variable Neighborhood Search; Graph Convolutional Networks; Graph Neural Networks

Article

Single Solution Based Metaheuristics with Graph Network Based Machine Learning

Mia Cryzan

miacryzan@outlook.com

Abstract: Simulated Annealing (SA), Tabu Search (TS), and Variable Neighborhood Search (VNS) are well-established single-solution-based metaheuristics known for their efficiency. In this paper, we explore the application of these techniques to enhance the performance of graph network models, specifically Graph Neural Networks (GNNs) and Graph Convolutional Networks (GCNs). By leveraging these metaheuristics, we aim to optimize key aspects of graph-based learning, such as hyperparameter tuning, architecture design, feature selection, and graph structure learning. Our approach demonstrates how these metaheuristics can improve the generalization, efficiency, and robustness of GNNs/GCNs, facilitating their use in large-scale graph-based tasks. The proposed framework illustrates the potential of combining advanced optimization techniques with state-of-the-art graph machine learning models.

Keywords: tabu search; simulated annealing; variable neighborhood search; graph convolutional networks; graph neural networks

---

1. Introduction

Single-solution based metaheuristics are optimization algorithms that rely on a single candidate solution, iteratively improving it until a stopping criterion is met. When combined with Graph Neural Networks (GNNs), these algorithms can effectively explore and optimize complex, high-dimensional solution spaces, particularly in problems involving graph-based data structures such as social networks, recommendation systems, and biological networks. We will explore the possibilities of Simulated Annealing (SA), Tabu Search (TS) and Variable Neighborhood Search (VNS) to work with graph network based machine learning solutions. We look forward to server as a bridge to connect traditional neighbourhood search with machine learning,

2. Realted Work

2.1. Simulated Annealing

Simulated Annealing (SA) is one of the most well-known single-solution based metaheuristics, inspired by the annealing process in metallurgy. In this process, metals are heated and then cooled in a controlled manner to decrease the energy states of the material, ultimately leading to a more stable structure. Analogously, SA explores the solution space by initially allowing worse solutions to be accepted with a higher probability, which helps in escaping local optima. As the algorithm proceeds, the acceptance probability decreases, focusing the search on areas around the current solution. In machine learning applications, such as GNNs, SA can be used to optimize parameters or network structures, helping the model avoid local minima during training. [1]

Simulated Annealing (SA) is highly adaptable to graph-based problems, especially in cases where the solution space involves a combinatorial structure. For instance, in network routing problems, such as the shortest path problem or network flow optimization, SA can explore different routing configurations to minimize the cost or time associated with network traversal. It is particularly useful in graph partitioning tasks, where the goal is to divide a graph into smaller, balanced subgraphs while minimizing the number of edges cut between them. SA efficiently explores the space of possible partitions by allowing suboptimal partitions initially, which helps avoid getting stuck in local minima.

SA can also be employed in community detection within social networks. This problem involves identifying subgroups of nodes that are more densely connected internally than to the rest of the network. SA iteratively improves the partitioning, making it well-suited for handling the non-convex nature of the solution space often seen in social network structures.

2.2. Tabu Search

Tabu Search (TS) is another powerful single-solution based metaheuristic, which is distinguished by its use of a memory structure known as a "tabu list." The tabu search was first introduced in [2]. The tabu list stores recently visited solutions, preventing the algorithm from revisiting these points in the search space. This encourages exploration of new areas and helps avoid cycling between solutions. In the context of GNNs, TS can be employed to optimize hyperparameters or the model architecture, enhancing the ability to escape local optima in high-dimensional search spaces where complex dependencies exist between nodes and edges in a graph. The memory feature makes TS particularly effective when applied to problems where the search space contains many local minima, as it promotes exploration beyond them.

Tabu Search (TS) is particularly effective for solving graph-related optimization problems where cycles or revisits to previously explored configurations can hinder the search process. In vehicle routing problems (VRP), TS can optimize delivery routes by maintaining a memory of previous solutions (tabu list) to avoid revisiting inefficient routing configurations. This is especially useful in dynamic routing scenarios where traffic conditions or demands change frequently, as TS can help explore alternative routes without revisiting the same paths.

In communication networks, where the goal is to maximize throughput or minimize latency, TS can optimize network configurations by preventing the exploration of suboptimal configurations that have been recently evaluated. Similarly, in graph coloring, where the aim is to color the nodes of a graph such that no two adjacent nodes share the same color with the least number of colors, TS ensures efficient exploration of coloring patterns without getting stuck in cycles of repeated configurations.

2.3. Variable Neighborhood Search

Variable Neighborhood Search (VNS) differs from both SA and TS by systematically changing the neighborhood structure during the search process. The core idea is that different neighborhood structures may reveal new, better solutions that are not accessible when using a single neighborhood definition. VNS alternates between exploring a local neighborhood of the current solution and jumping to a different neighborhood when stuck in a local optimum. This adaptive mechanism is well-suited for optimizing GNN-based models, as it can dynamically adjust the search process depending on the complexity of the graph's structure. For example, different neighborhoods can represent various graph topologies or node/edge features, allowing VNS to efficiently explore the solution space.

Variable Neighborhood Search (VNS) is particularly well-suited for graph networks where the problem's complexity requires exploring multiple neighborhood structures. In supply chain networks, for instance, VNS can be applied to optimize the location of facilities or distribution centers, switching between different neighborhood structures representing local and global supply chain configurations. By dynamically changing the neighborhood structure, VNS can escape local optima and find better configurations for minimizing transportation costs or improving service levels.

VNS is also highly applicable in network reliability problems. These problems often involve determining the most robust network configuration, given the possibility of node or edge failures. VNS systematically explores different subgraphs, trying to maximize network connectivity and robustness under varying failure scenarios. The flexibility of VNS in switching between neighborhoods allows it to adapt to the changing complexity of real-world network conditions.

In maximum clique problems, where the goal is to find the largest fully connected subgraph, VNS efficiently explores different subgraph configurations by switching between neighborhoods defined by clique sizes or edge densities. This makes it suitable for applications in social networks, biological networks, and other graph structures where cliques are a central feature.

2.4. Graph Based Machine Learning

Graph-based machine learning has gained significant attention due to its ability to model complex, relational data that is naturally represented as graphs. Unlike traditional data structures such as arrays or matrices, graphs can capture both the entities (nodes) and their relationships (edges) in a structured manner. This unique ability makes graph-based approaches well-suited for a wide range of applications, including social networks, biological networks, recommendation systems, and communication networks.

Traditional machine learning models were initially adapted to graph data through feature extraction techniques. Early methods relied on graph kernels, which aimed to measure the similarity between graphs by mapping them into a high-dimensional feature space. Notable works such as the Random Walk Kernel [3] and the Weisfeiler-Lehman Kernel [4] have been developed to compute graph similarities. These methods were instrumental in making traditional machine learning models compatible with graph data, but they suffered from high computational costs and limited scalability in large graph structures.

Another early technique that gained traction was spectral graph theory, which laid the foundation for learning on graphs by leveraging the graph Laplacian matrix. Techniques such as spectral clustering [1] and manifold learning methods like Laplacian Eigenmaps [1] and Diffusion Maps [1] introduced ways to perform dimensionality reduction or clustering in graph-structured data. These spectral methods, however, also faced scalability issues due to the computational cost of eigenvalue decompositions, particularly in very large graphs.

The development of Graph Neural Networks (GNNs) marked a significant shift in graph-based machine learning, as these models could learn directly from the graph structure without requiring manual feature extraction. Early GNNs, such as the one proposed by [5], extended the idea of recursive neural networks to graphs. However, it was with the introduction of Graph Convolutional Networks (GCNs) by [6] that GNNs gained widespread adoption. GCNs generalized the concept of convolution from grid-based data (such as images) to arbitrary graphs, allowing for node embeddings to be computed based on the local graph structure.

Subsequent works introduced more sophisticated models that improved the expressive power of GNNs. For example, Graph Attention Networks (GATs) [7] introduced an attention mechanism to focus on the most relevant neighboring nodes, improving performance in cases where node importance varies across the graph. GraphSAGE [8] was another pivotal work that allowed for inductive learning on large graphs by sampling and aggregating information from a node's neighbors. This enabled the model to generalize to unseen nodes and graphs, overcoming the limitations of transductive GCNs.

Beyond GCNs, several extensions to graph neural networks have emerged, targeting specific challenges in graph-based machine learning. Relational Graph Convolutional Networks (R-GCNs) [5] extended GCNs to handle multi-relational graphs, which are graphs where edges can have multiple types, making them particularly useful for knowledge graphs and heterogeneous networks. Dynamic GNNs, such as the Temporal Graph Networks (TGNs) [9], address the challenge of learning from dynamic or evolving graphs, which is crucial in applications like social networks and financial markets.

The combination of GNNs with reinforcement learning has also opened new research directions. Deep Q-Networks (DQN) and policy gradient methods have been integrated with GNNs to solve complex decision-making problems in graph-structured environments, such as traffic management and robot navigation.

Despite the impressive progress, several challenges remain in graph-based machine learning. One key issue is the scalability of GNNs on large, real-world graphs, such as social networks with millions or billions of nodes and edges. Techniques like mini-batching and sampling (e.g., GraphSAGE) have been proposed, but further work is needed to make GNNs more scalable without compromising accuracy.

Another challenge is the expressiveness of GNNs. Recent works have highlighted that traditional GNNs, such as GCNs, may struggle to capture certain graph properties, such as cycles or cliques, leading to the development of more powerful models like higher-order GNNs and neural message passing networks. Understanding the theoretical limitations of GNNs and designing architectures that can overcome them is a key area of ongoing research.

Furthermore, the interpretability of GNNs remains a significant concern, especially in domains like healthcare and finance, where understanding model predictions is critical. Techniques for explaining GNN decisions, such as GraphGrad-CAM or graph explainability frameworks, are being actively explored.

3. SA for Graph Network

SA, a well-established single-solution metaheuristic optimization technique, can play a crucial role in enhancing the performance of Graph Neural Networks (GNNs) and Graph Convolutional Networks (GCNs) by optimizing various aspects of their architecture and learning process. Below, I elaborate on several ways SA can be integrated to improve GNNs or GCNs:

3.1. Hyperparameter Optimization

One of the most direct applications of SA in GNNs and GCNs is hyperparameter tuning. Like many deep learning models, GNNs depend heavily on hyperparameters such as the learning rate, number of layers, number of hidden units, dropout rate, and regularization terms. Manually selecting these hyperparameters or using grid/random search can be inefficient and time-consuming, especially as the model's complexity grows.Simulated Annealing offers an effective way to navigate this hyperparameter space:

- Exploration and Exploitation: SA starts with a high "temperature," allowing it to explore a wide range of hyperparameter combinations by accepting both better and worse configurations with certain probabilities. As the temperature decreases, the algorithm gradually shifts to more exploitation, focusing on refining the best combinations found.
- Avoiding Local Optima: Hyperparameter optimization landscapes can have many local optima, and GNNs or GCNs are susceptible to suboptimal configurations. SA's probabilistic acceptance of worse solutions early in the process helps in escaping local optima, leading to potentially better-performing hyperparameters. By efficiently optimizing hyperparameters, SA can improve the training performance of GNNs/GCNs, enabling the model to generalize better and avoid overfitting.

3.2. Architecture Search

SA can also be applied to optimize the architecture of GNNs. Designing the optimal GNN architecture involves decisions like:

- The number of convolutional layers,
- The type of aggregation functions (e.g., mean, sum, max),
- The activation functions,
- The type of skip connections or residual structures, and
- How to model multi-hop neighborhood information.

Rather than relying on manually designed architectures or using methods like random search, SA can help navigate the space of possible architectures by:

- Modifying Layer Compositions: SA can iteratively modify the number and types of layers in the GNN or GCN. For instance, early in the annealing process, SA might explore architectures with many layers, various activation functions, or complex aggregation schemes. Over time, as temperature decreases, it converges to more refined architectures that are more effective at capturing graph representations.
- Balancing Model Complexity and Performance: Because deeper or more complex GNN architectures are not always better, SA helps balance the trade-off between overfitting (when the model

is too complex) and underfitting (when the model is too simple). The probabilistic nature of SA ensures that the architecture search does not get stuck in suboptimal configurations, especially in the early stages of exploration.

This approach can lead to more powerful, task-specific GNN architectures that better capture the graph's structural information, potentially improving accuracy and efficiency.

3.3. Node Feature Selection

GNNs rely on node features to generate embeddings and make predictions. However, not all node features are equally important, and selecting the optimal subset of features can significantly impact the model's performance. SA can be used for node feature selection in GNNs by:

- Searching for Optimal Feature Subsets: SA can iteratively explore subsets of node features, allowing for the possibility of accepting less-optimal sets early on. Over time, it converges to a more refined subset of features that yield better embeddings and improved model performance.
- Regularizing Over-Complex Feature Sets: SA can prevent over-reliance on noisy or redundant features by probabilistically eliminating them during the annealing process. This can result in more interpretable and robust GNN models, especially in domains like biology or chemistry where interpretability of node features (e.g., genes or molecular properties) is crucial.

By selecting the most informative node features, SA can reduce the complexity of GNNs, making them more efficient and interpretable without compromising performance.

3.4. Graph Sampling and Data Augmentation

In GNNs, particularly in large-scale graphs, full-batch training can be computationally expensive. Many methods, like GraphSAGE and Cluster-GCN, employ graph sampling techniques to reduce computational burden. However, the choice of sampling strategy can significantly affect the performance of the GNN. Simulated Annealing can improve GNNs by optimizing graph sampling strategies:

- Dynamic Sampling Optimization: SA can dynamically optimize the node or subgraph sampling strategies during training by exploring different sampling schemes in early iterations. For instance, in the initial stages, SA may allow less optimal sampling configurations, such as smaller subgraphs or nodes with low centrality. As training progresses, it refines these sampling schemes to focus on more informative subgraphs or higher-degree nodes.
- Data Augmentation: SA can explore different graph augmentation strategies to improve generalization. By allowing worse augmentations early on and gradually focusing on better ones, SA can help the GNN train on more varied and informative samples, leading to improved robustness.

3.5. Graph Structure Learning

In certain cases, the graph structure itself may not be fully known, or it might be noisy, as in social networks or biological networks where connections between nodes may be uncertain or missing. SA can assist in graph structure learning, where the goal is to improve the structure of the graph itself by:

- Edge Pruning and Addition: SA can iteratively prune or add edges to the graph during training, allowing it to probabilistically accept worse graph structures in early iterations. As the annealing process progresses, the focus shifts towards refining the graph to improve model performance. For instance, SA can help remove noisy or irrelevant edges that might otherwise negatively impact the learning process.
- Learning Better Topologies: This is especially useful in problems where the graph structure is only partially observed (e.g., link prediction or semi-supervised learning), where SA can optimize the graph's topology to improve the predictive performance of the GNN.

3.6. Optimization of Loss Functions

SA can also play a role in optimizing the loss function for GNNs, particularly when the objective function is non-convex or involves multiple competing objectives (e.g., accuracy vs. sparsity). By incorporating SA, the GNN training process can escape poor local minima in the loss landscape, leading to better solutions. For instance, in multi-task learning with GNNs, where the model must balance multiple loss functions, SA can help find a good trade-off between competing objectives.

4. Tabu Search for GCN and GNN

Tabu Search (TS) can enhance Graph Convolutional Networks (GCNs) and Graph Neural Networks (GNNs) by systematically exploring the solution space while avoiding cycles and revisiting previously explored suboptimal solutions.

- Hyperparameter Optimization: TS efficiently navigates the hyperparameter space (e.g., learning rate, layer depth) by utilizing a memory structure (tabu list) to avoid revisiting less promising configurations, leading to faster convergence and improved model performance [10].
- Architecture Search: TS can optimize GNN/GCN architectures by avoiding redundant exploration of previously evaluated layer structures or aggregation functions, promoting diverse and potentially more effective configurations.
- Feature Selection: TS can help select optimal node features by keeping track of previously discarded feature sets, thereby focusing on new and potentially better combinations of node attributes.
- Graph Sampling: In large-scale graphs, TS can optimize node or edge sampling strategies by avoiding inefficient sampling configurations, leading to better model efficiency without sacrificing performance.
- Graph Structure Learning: TS can optimize the graph's topology by modifying edges (e.g., adding/removing connections) and preventing repetitive suboptimal adjustments, improving tasks like link prediction and semi-supervised learning.

5. VNS for GCN and GNN

Applying VNS to GNNs and GCNs can improve various aspects of their design, training, and optimization. Below are several ways in which VNS can be beneficial for GNNs/GCNs:

5.1. Optimizing Hyperparameters through Neighborhood Exploration

Similar to other machine learning models, GNNs/GCNs have several hyperparameters that need to be tuned to ensure optimal performance. These include learning rates, layer depths, regularization parameters, and neighborhood aggregation functions [11]. VNS can aid in the hyperparameter optimization process by systematically exploring various hyperparameter combinations using different "neighborhoods," where each neighborhood represents a set of related hyperparameter configurations [12].

agraphNeighborhood Definition In the context of hyperparameters, different neighborhoods can be defined as distinct hyperparameter ranges or types. For instance, one neighborhood might consist of smaller learning rates, another could explore larger layer depths, and another might explore variations in activation functions (e.g., ReLU vs. Leaky ReLU). VNS cycles through these neighborhoods, evaluating the performance of each configuration and adapting based on the results [13]. agraphEscaping Local Optima Traditional gradient-based optimization techniques or random search may get stuck in suboptimal hyperparameter settings. By shifting to new neighborhoods, VNS increases the chance of escaping local minima and finding more effective configurations. This results in a more robust training process for GNNs/GCNs.

By employing VNS to optimize hyperparameters, GNNs and GCNs can achieve better accuracy, faster convergence, and greater stability across different datasets.

5.2. Improving Graph Sampling Techniques

Graph sampling is critical for training GNNs/GCNs efficiently, particularly when working with large-scale graphs. Techniques like GraphSAGE and FastGCN aim to sample subsets of the graph to reduce computational complexity. VNS can be employed to improve these graph sampling strategies by dynamically shifting between different neighborhoods of node or subgraph sampling techniques during training.

agraphNeighborhood Definition in Graph Sampling In the context of graph sampling, different neighborhoods could be defined as different sampling strategies. For example, one neighborhood might involve sampling higher-degree nodes, another might involve uniform random sampling, while a third might focus on subgraphs generated using clustering methods. agraphAdaptive Sampling During training, VNS can dynamically switch between these sampling neighborhoods to optimize the model's learning process. Early in training, VNS might explore diverse sampling methods, while later on, it can converge to more refined strategies that better capture the graph structure relevant to the task at hand.

This approach ensures that GNNs/GCNs are exposed to more diverse and informative subgraphs, improving both model generalization and computational efficiency.

5.3. Neighborhood Search for Network Architectures

Designing optimal architectures for GNNs/GCNs is a challenging task, as there are many potential choices, such as the number of layers, types of aggregation functions, and how to handle multi-hop neighbors. VNS can be applied to neural architecture search (NAS) by defining different neighborhoods as different sets of architectural configurations, and then systematically exploring these configurations to find the most effective model.

agraphLayer Exploration VNS can shift between neighborhoods with different numbers of graph convolutional layers. In some neighborhoods, fewer layers might be explored, while in others, deeper architectures might be evaluated. agraphAggregation Functions GNNs aggregate information from neighboring nodes in each layer. VNS can explore different aggregation functions (mean, max, sum, etc.) by switching between neighborhoods where each uses a distinct aggregation method. agraphExploring Different Graph Connectivity Another key component of GNN architecture is how multi-hop neighbors are aggregated. VNS can be applied to search for optimal combinations of neighborhood hops (e.g., one-hop vs. two-hop connections) and residual connections that balance the need for both local and global graph structure information.

By employing VNS to navigate the architectural search space, GNNs/GCNs can benefit from architectures that are more effective at capturing graph-based representations, ultimately leading to improvements in model performance.

5.4. Feature Selection Using VNS

The quality of the input features used in GNNs/GCNs plays a critical role in their ability to learn meaningful node embeddings and make accurate predictions. VNS can assist in feature selection by iteratively exploring different neighborhoods of feature subsets. Each neighborhood in this case would consist of a specific subset of node features (such as centrality measures, node degrees, or domain-specific features), and VNS can shift between these subsets to identify the most informative ones.

agraphNeighborhood Definition for Features Each neighborhood can correspond to a different subset or combination of node features. For example, one neighborhood may focus on centrality-based features (e.g., betweenness or closeness centrality), while another may explore node-level features such as node degrees or embedding dimensions from prior layers. agraphEscaping Local Optima in Feature Space Feature selection in graph-based models is a highly combinatorial problem, where relying on just a single feature or feature subset can result in suboptimal performance. VNS allows the model to systematically escape poor feature sets by shifting neighborhoods, thus improving generalization and reducing overfitting.

Using VNS for feature selection enables GNNs to focus on the most relevant features, leading to a more compact and interpretable model while maintaining high accuracy.

5.5. Dynamic Neighborhood Selection for Training GNNs

GNNs and GCNs often aggregate information from neighbors within a graph, where the neighborhood size (e.g., number of hops or the number of sampled neighbors) can significantly affect performance. The ideal neighborhood size may vary depending on the graph structure, the task at hand, or even the stage of training. VNS can be used to dynamically adjust the neighborhood size during training.

agraphNeighborhood Size Exploration Different neighborhoods in VNS can represent varying numbers of neighbors or hops considered for aggregation. One neighborhood might explore only direct neighbors (one-hop), while another might include second-hop neighbors (two-hop), and another might aggregate from larger multi-hop neighborhoods.

agraphAdjusting Neighborhood Dynamically Early in training, VNS might explore larger neighborhoods to capture more global information, while later on, it may refine the focus on smaller, more local neighborhoods as it converges to an optimal solution.

By adapting the neighborhood size during training, VNS can help the GNN/GCN capture both local and global graph structure, leading to better performance in tasks like node classification, link prediction, or graph-level tasks.

5.6. Escaping Local Optima in Graph Structure Learning

In cases where the graph structure is not fully known or is noisy (such as in social networks or molecular graphs), graph structure learning is an important task. The goal is to infer or modify the edges between nodes to improve model performance. VNS can help optimize this process by searching different neighborhoods of graph structures, where each neighborhood represents a slightly modified graph.

agraphGraph Edge Modification Different neighborhoods can correspond to variations in the graph's edge structure. VNS can explore neighborhoods where some edges are added, removed, or re-weighted. This is particularly useful in problems like semi-supervised learning or link prediction, where the goal is to predict missing edges or modify incorrect ones. agraphGraph Topology Search VNS can also help optimize the topology of the graph by switching between neighborhoods that represent different topological structures. This can be useful in tasks like community detection or clustering, where the goal is to learn the underlying graph structure that best explains the data.

By dynamically modifying the graph structure, VNS helps GNNs/GCNs learn better representations, particularly when dealing with noisy, incomplete, or evolving graphs.

6. Conclusions

In conclusion, single-solution based metaheuristics like SA, TS, and VNS offer robust and flexible approaches to optimize GNNs. These methods provide a balance between exploration and exploitation of the solution space, making them suitable for complex, non-convex optimization problems often encountered in graph-based machine learning tasks.

References

1. 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.
2. Glover, F. Tabu Search—Part I. *ORSA Journal on Computing* **1989**, *1*, 190–206. doi:10.1287/ijoc.1.3.190.
3. Vishwanathan, S.; Schraudolph, N.N.; Kondor, R.; Borgwardt, K.M. Graph kernels. *Journal of Machine Learning Research* **2010**, *11*, 1201–1242.

4. Shervashidze, N.; Schweitzer, P.; Leeuwen, E.J.v.; Mehlhorn, K.; Borgwardt, K.M. Weisfeiler-Lehman graph kernels. Advances in Neural Information Processing Systems, 2011, pp. 1–9.
5. Schlichtkrull, M.; Kipf, T.N.; Bloem, P.; van den Berg, R.; Titov, I.; Welling, M. Modeling relational data with graph convolutional networks. *European Semantic Web Conference* **2018**, pp. 593–607.
6. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations (ICLR), 2017.
7. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. International Conference on Learning Representations (ICLR), 2018.
8. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive representation learning on large graphs. Advances in Neural Information Processing Systems, 2017, pp. 1024–1034.
9. Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; Bronstein, M. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* **2020**.
10. Guo, X.; Quan, Y.; Zhao, H.; Yao, Q.; Li, Y.; Tu, W. Tabgnn: Multiplex graph neural network for tabular data prediction. *arXiv preprint arXiv:2108.09127* **2021**.
11. 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.
12. Mladenovic, N.; Hansen, P. Variable Neighborhood Search. *Computers & Operations Research* **1997**, *24*, 1097–1100. doi:10.1016/S0305-0548(97)00031-2.
13. Johnn, S.N.; Darvariu, V.A.; Handl, J.; Kalcsics, J. A Graph Reinforcement Learning Framework for Neural Adaptive Large Neighbourhood Search. *Computers & Operations Research* **2024**, *172*, 106791.