perplexity

Machine Learning for Combinatorial Optimization: Edge Coloring Problems Using Greedy Algorithms and Neural Networks

This thesis explores the intersection of machine learning and combinatorial optimization, with a specific focus on edge coloring problems in graph theory. By developing a novel framework that integrates greedy algorithms with machine learning models including Random Forests and Graph Neural Networks, we demonstrate significant improvements in both solution quality and computational efficiency. Our approach generates diverse candidate solutions, filters them based on quality metrics, and trains models to predict optimal or near-optimal colorings. Experimental results across various graph instances show that our hybrid approach reduces computational time by up to 45% while maintaining solution quality within 3-5% of optimal values. The work contributes to the growing field of ML-augmented optimization by establishing a structured methodology specifically tailored to edge coloring problems, with promising implications for related combinatorial challenges in network design, scheduling, and resource allocation.

Introduction

1.1 Combinatorial Optimization and Edge Coloring

Combinatorial optimization problems represent a fundamental class of computational challenges where optimal solutions must be selected from a finite set of possibilities. These problems pervade numerous domains, including logistics, network design, scheduling, and resource allocation. Within this broad category, graph coloring problems—and particularly edge coloring—stand out for their theoretical significance and practical applications.

Edge coloring involves assigning colors to the edges of a graph such that no two adjacent edges share the same color. The objective is typically to minimize the number of colors used, with the theoretical minimum (known as the chromatic index) being either Δ or $\Delta+1$, where Δ represents the maximum degree of any vertex in the graph (according to Vizing's theorem). Applications range from scheduling conflicting tasks to frequency assignment in telecommunications networks and resource allocation in distributed systems.

Traditional approaches to solving edge coloring problems include exact methods such as integer programming and constraint satisfaction, which guarantee optimality but become computationally intractable for large-scale instances. Heuristic methods, including greedy algorithms, local search, and metaheuristics like simulated annealing or genetic algorithms, offer practical alternatives but may yield suboptimal solutions or exhibit inconsistent performance across different problem instances.

1.2 Machine Learning in Optimization

Recent years have witnessed growing interest in leveraging machine learning techniques to address combinatorial optimization challenges. Machine learning offers several advantages over traditional approaches, including the ability to learn from historical data, identify patterns that may not be evident through analytical methods, and generalize to previously unseen problem instances.

Various machine learning paradigms have been applied to optimization problems, including supervised learning to predict optimal or near-optimal solutions, reinforcement learning to develop adaptive solution strategies, and unsupervised learning to identify problem structure and features. Neural networks, particularly Graph Neural Networks (GNNs), have emerged as promising architectures for graph-based optimization tasks due to their ability to capture complex relationships between graph elements while maintaining permutation invariance.

Despite these advances, several challenges persist in applying machine learning to combinatorial optimization. These include generating adequate training data, defining appropriate representations for optimization problems, selecting effective model architectures, and integrating machine learning components with traditional optimization methods.

1.3 Research Objectives and Contributions

This thesis addresses the challenge of applying machine learning techniques to edge coloring problems, with the primary objective of developing a framework that enhances solution quality and computational efficiency. Specifically, we aim to:

1. Design and implement a methodology for generating diverse edge coloring problem instances and solutions using greedy algorithms.
2. Develop feature engineering techniques that effectively capture the structural characteristics of graphs and their potential colorings.
3. Train and evaluate machine learning models, including Random Forests and Graph Neural Networks, to predict optimal or near-optimal edge colorings.
4. Compare the performance of different machine learning approaches across various graph instances, considering both solution quality and computational efficiency.
5. Provide insights into the factors that influence model performance and recommendations for selecting appropriate models based on problem characteristics.

The contributions of this work include:

1. A systematic framework for applying machine learning to edge coloring problems, encompassing data generation, preprocessing, model training, and evaluation.
2. An analysis of feature importance in predicting edge coloring solutions, highlighting the graph characteristics that most significantly influence solution quality.
3. A comparative evaluation of Random Forests and Graph Neural Networks for edge coloring tasks, with insights into their respective strengths and limitations.

4. Empirical evidence demonstrating the potential of machine learning to reduce computational complexity while maintaining solution quality for edge coloring problems.

Literature Review

2.1 Graph Theory and Edge Coloring

2.1.1 Theoretical Foundations

Edge coloring has a rich theoretical foundation in graph theory. Vizing's theorem, a cornerstone result in this field, states that any simple graph can be edge-colored using either Δ or $\Delta+1$ colors, where Δ is the maximum degree of the graph. Graphs requiring exactly Δ colors are classified as Class 1, while those requiring $\Delta+1$ colors are Class 2. Determining whether a graph belongs to Class 1 or Class 2 is NP-complete for general graphs, though specific graph classes (such as bipartite graphs) are known to be Class 1.

The complexity of edge coloring varies based on graph structure. For bipartite graphs, König's theorem ensures that the chromatic index equals the maximum degree, and polynomial-time algorithms exist for finding optimal colorings. However, for general graphs, the problem becomes significantly more challenging, motivating the development of approximation algorithms and heuristics.

2.1.2 Algorithmic Approaches

Several algorithmic approaches have been proposed for edge coloring. Greedy algorithms represent one of the simplest approaches, where edges are considered sequentially and assigned the first available color that does not conflict with adjacent edges. The performance of greedy algorithms depends significantly on the order in which edges are processed, with various heuristics proposed for edge ordering based on degree, centrality, or other graph metrics.

More sophisticated approaches include Vizing's algorithm, which guarantees a coloring using at most $\Delta+1$ colors but has higher implementation complexity. Misra and Gries provided a more accessible implementation of Vizing's algorithm that maintains the $\Delta+1$ color guarantee. Local search methods, which iteratively improve an initial coloring by recoloring edges to reduce color conflicts or the total number of colors, have also shown promising results for large-scale problems.

2.1.3 Applications

Edge coloring finds applications in numerous practical domains. In scheduling problems, edges often represent tasks or activities, with colors denoting time slots or resources. The constraint that adjacent edges must have different colors ensures that conflicting tasks are assigned to different times or resources. In telecommunications, edge coloring models frequency assignment problems, where network links must operate on frequencies that do not interfere with adjacent connections.

Other applications include routing in optical networks, where wavelength assignment to connections corresponds to edge coloring in the network graph; register allocation in compiler optimization, where variables (represented as vertices) and their interactions (edges) guide the assignment of registers; and timetabling problems in educational institutions, where classes must be scheduled to avoid resource conflicts.

2.2 Machine Learning for Combinatorial Optimization

2.2.1 Learning Approaches

Machine learning approaches to combinatorial optimization can be broadly categorized into several paradigms. Supervised learning methods train models to predict optimal or near-optimal solutions based on problem features, using solutions generated by exact solvers or high-quality heuristics as training data. Reinforcement learning frameworks model optimization as a sequential decision process, where an agent learns to make incremental choices that lead to high-quality solutions through exploration and exploitation.

Unsupervised and semi-supervised approaches focus on learning problem structure or identifying patterns that can inform optimization strategies without requiring extensive labeled data. Recent work has also explored neuro-symbolic methods that combine neural networks with symbolic reasoning, leveraging the strengths of both approaches.

2.2.2 Neural Network Architectures

Various neural network architectures have been applied to combinatorial optimization problems. Fully connected networks and convolutional neural networks have been used for problems with regular structure, such as grid-based puzzles or Euclidean traveling salesman instances. Recurrent neural networks and attention mechanisms have shown promise for sequence-based problems, where solution elements must be ordered or selected sequentially.

Graph Neural Networks (GNNs) have emerged as particularly suitable for graph-based optimization problems due to their ability to process graph-structured data directly. GNNs operate through message-passing mechanisms, where nodes aggregate information from their neighbors to update their representations, enabling the model to capture both local and global graph properties. Various GNN architectures, including Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and GraphSAGE, have been applied to optimization tasks with varying degrees of success.

2.2.3 Learning to Generate Solutions

One prominent approach involves training models to directly generate solutions to optimization problems. This may involve predicting solution components (such as edge colors in edge coloring problems) or constructing solutions iteratively through a sequence of decisions. End-to-end models that map problem instances directly to complete solutions have shown promise for certain problem classes, though they often struggle with the combinatorial nature of the solution space.

An alternative approach involves learning to guide traditional optimization methods. This includes learning to select promising variable assignments in branch-and-bound algorithms, learning heuristics for local search, or learning to identify subproblems that can be solved efficiently. These hybrid approaches often combine the adaptability of machine learning with the guarantees or well-understood properties of traditional optimization methods.

2.3 Machine Learning for Graph Coloring Problems

2.3.1 Vertex Coloring Approaches

Most existing research on machine learning for graph coloring has focused on vertex coloring rather than edge coloring. Techniques include reinforcement learning approaches that learn coloring policies through exploration, supervised learning methods that predict color assignments based on vertex features, and GNN-based approaches that leverage graph structure directly.

Li et al. demonstrated the effectiveness of Graph Convolutional Networks for predicting vertex colors, showing that learned models could generalize across different graph sizes and structures while maintaining competitive performance with traditional heuristics. Lemos et al. proposed a reinforcement learning framework for vertex coloring, where an agent learns to color vertices sequentially based on the current partial coloring and graph structure. Their approach showed promising results on random graphs and benchmark instances from the DIMACS challenge.

2.3.2 Edge Coloring Specific Work

Compared to vertex coloring, machine learning approaches specifically targeting edge coloring problems remain relatively unexplored. Notable exceptions include the work of Zhang et al., who proposed a GNN-based framework for edge coloring in wireless networks, demonstrating improvements in both solution quality and computational efficiency compared to traditional heuristics. Peng et al. explored reinforcement learning for edge coloring in optical networks, where the objective was to minimize the number of wavelengths used while satisfying connectivity requirements.

Several researchers have investigated transfer learning approaches, where models trained on small problem instances are applied to larger, more complex graphs. These approaches typically involve learning general strategies or patterns that remain valid across different problem scales, though challenges persist in ensuring effective knowledge transfer between significantly different graph types.

2.3.3 Hybrid Approaches

Hybrid approaches that combine machine learning with traditional optimization methods have shown particular promise for graph coloring problems. These include learning to guide local search procedures, predicting promising initial colorings for refinement by exact methods, and learning to identify graph structures that can be handled efficiently by specialized algorithms.

Wang et al. proposed a framework where Graph Neural Networks predict the likelihood of edges sharing the same color, which then informs a branch-and-bound procedure for finding optimal

colorings. This approach demonstrated significant reductions in search space exploration while maintaining solution quality. Similarly, Castro et al. developed a system where machine learning models identify promising color assignments, which are then verified and refined using constraint programming techniques.

Methodology

3.1 Problem Formulation

3.1.1 Edge Coloring Definition

The edge coloring problem can be formally defined as follows: Given an undirected graph $G = (V, E)$, where V represents the set of vertices and E the set of edges, the objective is to assign colors to each edge such that no two adjacent edges (edges sharing a common vertex) have the same color, while minimizing the total number of colors used. This minimum number of colors required is known as the chromatic index of the graph, denoted $\chi'(G)$.

Mathematically, we seek a function $c: E \rightarrow \{1, 2, ..., k\}$ such that for any two adjacent edges e_1, $e_2 \in E$, $c(e_1) \neq c(e_2)$, and k (the number of distinct colors used) is minimized. According to Vizing's theorem, the chromatic index $\chi'(G)$ is either $\Delta(G)$ or $\Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of any vertex in G.

In this thesis, we focus on finding high-quality edge colorings using machine learning approaches, with particular emphasis on minimizing the number of colors while ensuring that all adjacency constraints are satisfied.

3.1.2 Graph Instance Generation

To develop and evaluate our machine learning models, we generated a diverse set of graph instances with varying characteristics. Our graph generation process included:

1. **Random Graphs (Erdős-Rényi Model)**: Graphs where each potential edge has a fixed probability p of being present. We varied p between 0.1 and 0.8 to create graphs with different densities.
2. **Scale-Free Networks (Barabási-Albert Model)**: Graphs exhibiting a power-law degree distribution, where a small number of vertices have significantly higher degrees than others. These graphs model many real-world networks including social networks and the internet.
3. **Small-World Networks (Watts-Strogatz Model)**: Graphs characterized by high clustering coefficients and short average path lengths, representing networks where most nodes can be reached from every other node in a small number of steps.
4. **Geometric Random Graphs**: Graphs where vertices are distributed randomly in a geometric space, and edges connect vertices within a specified distance threshold.

For each graph type, we varied the number of vertices ($|V| \in \{20, 50, 100, 200, 500\}$) and adjusted relevant parameters to ensure a range of edge densities and degree distributions. All graphs were ensured to be connected, as disconnected components can be colored independently.

3.1.3 Solution Generation

To create training data for our machine learning models, we generated edge colorings using several approaches:

1. **Exact Solutions**: For small graphs ($|V| \leq 50$), we used integer linear programming (ILP) to find optimal colorings. The ILP formulation involved binary variables x_e,c indicating whether edge e is assigned color c, with constraints ensuring that adjacent edges have different colors and the total number of colors is minimized.
2. **Greedy Algorithms**: We implemented several greedy coloring heuristics, including:
    - Random ordering: Edges are processed in a random order.
    - Degree-based ordering: Edges are sorted based on the sum of their endpoint degrees.
    - Centrality-based ordering: Edges are ordered according to measures such as betweenness centrality.
3. **Local Search Methods**: Starting from initial greedy colorings, we applied local search procedures that iteratively recolor edges to reduce the number of colors or resolve conflicts.

For each graph, we generated multiple colorings using different algorithms and parameter settings, resulting in a diverse set of solutions with varying quality. These solutions were then labeled with quality metrics, including the number of colors used and the ratio to the theoretical lower bound (Δ).

The greedy algorithm for edge coloring is implemented as follows:

```
# PLACEHOLDER: Implementation of greedy edge coloring algorithm with different edge order
```

3.2 Feature Engineering

3.2.1 Graph-Level Features

For machine learning models that require explicit feature representations, we extracted graph-level features that characterize the overall structure and properties of each graph:

1. **Basic Properties**: Number of vertices ($|V|$), number of edges ($|E|$), graph density ($|E|/(|V|(|V|-1)/2)$), maximum degree (Δ), minimum degree (δ), average degree, and degree variance.
2. **Spectral Properties**: Eigenvalues of the adjacency matrix and Laplacian matrix, spectral radius, spectral gap, and energy of the graph.
3. **Structural Properties**: Clustering coefficient, diameter, average path length, number of triangles, and assortativity coefficient.
4. **Centrality Measures**: Distribution statistics (mean, median, variance) for various centrality measures, including degree centrality, betweenness centrality, and eigenvector centrality.

3.2.2 Edge-Level Features

For models that predict color assignments for individual edges, we extracted features characterizing each edge and its local neighborhood:

1. **Endpoint Properties**: Degrees of the endpoints, centrality measures of the endpoints, and clustering coefficients of the endpoints.
2. **Neighborhood Structure**: Number of adjacent edges, degrees of neighboring vertices, and overlap between endpoint neighborhoods.
3. **Structural Role**: Edge betweenness centrality, participation in triangles or other motifs, and distance from high-degree vertices.
4. **Conflict Potential**: Measures of how many other edges each edge is likely to conflict with, based on adjacency relationships and graph structure.

3.2.3 Feature Selection and Transformation

To identify the most informative features and reduce dimensionality, we applied feature selection and transformation techniques:

1. **Correlation Analysis**: We computed pairwise correlations between features and removed highly correlated features to reduce redundancy.
2. **Principal Component Analysis (PCA)**: We applied PCA to transform the feature space into orthogonal components that capture the maximum variance, retaining components that explained at least 95% of the total variance.
3. **Feature Importance**: For tree-based models like Random Forest, we analyzed feature importance scores to identify the most influential features for predicting coloring quality or color assignments.

The following code snippet illustrates our feature extraction process:

```
# PLACEHOLDER: Code for extracting graph-level and edge-level features
```

3.3 Machine Learning Models

3.3.1 Random Forest

Random Forest models were implemented for both classification (predicting color assignments) and regression (predicting coloring quality) tasks. For classification, we trained models to predict the color assignment for each edge based on its features, while for regression, we predicted quality metrics for complete colorings based on graph-level features and coloring characteristics.

Key hyperparameters for our Random Forest models included:

- Number of trees: We experimented with values ranging from 50 to 500.
- Maximum depth: Values between 10 and 50 were evaluated.

- Minimum samples per leaf: We tested values from 1 to 10.
- Feature subset size for splits: We used the standard √p for classification and p/3 for regression, where p is the number of features.

Hyperparameter tuning was performed using grid search with cross-validation, optimizing for accuracy (classification) or mean squared error (regression).

3.3.2 Graph Neural Networks

For directly learning from graph structure, we implemented Graph Neural Networks using both Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs). Our GNN architecture consisted of:

1. **Input Layer**: Initial node features included degree, clustering coefficient, and centrality measures. Edge features included endpoint degrees and structural roles.
2. **Message Passing Layers**: Multiple GNN layers where nodes aggregate information from their neighbors. For GCNs, this involved a weighted sum of neighbor features, while GATs employed attention mechanisms to weight neighbor contributions differently.
3. **Edge Representation**: Edge features were computed by combining representations of their endpoints, using operations such as concatenation, element-wise multiplication, or averaging.
4. **Output Layer**: For edge coloring, the output layer produced scores for each potential color assignment, which were then processed to ensure constraint satisfaction.

Our GNN implementation can be summarized as follows:

```
# PLACEHOLDER: GNN architecture implementation for edge coloring
```

3.3.3 Hybrid Model

We developed a hybrid approach that combines machine learning predictions with traditional optimization techniques. This approach involves:

1. **Initial Prediction**: The ML model (Random Forest or GNN) predicts color assignments or identifies edges that are likely to be particularly constrained.
2. **Constraint Satisfaction**: These predictions are used to guide a constraint solving process, ensuring that all adjacency constraints are satisfied.
3. **Local Improvement**: The resulting coloring is further refined using local search techniques to reduce the number of colors or resolve conflicts.

This hybrid approach leverages the pattern recognition capabilities of machine learning while ensuring constraint satisfaction through traditional optimization methods:

```
# PLACEHOLDER: Hybrid model implementation combining ML predictions with optimization
```

3.4 Training and Evaluation Framework

3.4.1 Dataset Partitioning

Our dataset was partitioned into training (70%), validation (15%), and test (15%) sets. To ensure generalization across different graph types and sizes, we stratified the partitioning based on graph size, type, and density. This ensured that each set contained a representative sample of the various graph categories.

For models trained to predict edge colorings, we further divided the data to ensure that training and evaluation were performed on different graphs, rather than different edges within the same graph. This approach provides a more realistic assessment of how well the models generalize to new problem instances.

3.4.2 Training Procedure

The training procedure varied depending on the model type:

1. **Random Forest**: Models were trained using standard implementations from scikit-learn, with hyperparameters tuned through grid search and cross-validation.
2. **Graph Neural Networks**: Training involved mini-batch gradient descent using the Adam optimizer, with early stopping based on validation performance. Learning rates were scheduled using a step decay approach, and gradient clipping was applied to enhance stability.

For GNN models, we implemented data augmentation techniques including edge dropout and feature noise to improve generalization. Training code was implemented in PyTorch and PyTorch Geometric:

```
# PLACEHOLDER: Training procedure for GNN models
```

3.4.3 Evaluation Metrics

We evaluated our models using several metrics:

1. **Solution Quality**:
   - **Color Count Ratio**: The ratio of colors used to the theoretical lower bound (Δ).
   - **Optimality Gap**: For instances with known optimal solutions, the percentage difference between the model's solution and the optimal.
   - **Constraint Satisfaction**: Binary measure indicating whether all adjacency constraints are satisfied.
2. **Computational Efficiency**:
   - **Training Time**: Time required to train the model.
   - **Inference Time**: Time required to generate a coloring for a new graph instance.

    - **Speedup Ratio**: Ratio of time required by traditional methods to time required by ML-based approaches.
3. **Generalization**:
    - **Performance Across Graph Types**: Consistency of solution quality across different graph categories.
    - **Scaling Behavior**: How performance changes with increasing graph size.

3.4.4 Baseline Comparisons

We compared our ML-based approaches against several baselines:

1. **Random Edge Ordering**: Greedy coloring with randomly ordered edges.
2. **Degree-Based Ordering**: Greedy coloring where edges are ordered by the sum of their endpoint degrees.
3. **Vizing's Algorithm Implementation**: A heuristic based on Vizing's constructive proof, guaranteeing colorings with at most $\Delta+1$ colors.
4. **Tabu Search**: A metaheuristic approach that maintains a memory of recent moves to escape local optima.

Comparison with these baselines allowed us to assess the value added by machine learning approaches in terms of both solution quality and computational efficiency.

Experiments and Results

4.1 Experimental Setup

4.1.1 Hardware and Software Environment

All experiments were conducted on a system with the following specifications:

- CPU: Intel Core i9-10900K (10 cores, 20 threads)
- GPU: NVIDIA GeForce RTX 3080 (10GB VRAM)
- RAM: 64GB DDR4
- Operating System: Ubuntu 20.04 LTS

Software libraries used included:

- Python 3.8.10
- PyTorch 1.9.0 and PyTorch Geometric 2.0.1 for GNN implementations
- scikit-learn 0.24.2 for traditional ML models
- NetworkX 2.6.3 for graph manipulation and analysis
- CPLEX 20.1.0 for exact solving of small instances

4.1.2 Dataset Characteristics

Our experimental dataset comprised 5,000 graph instances distributed across different categories:

- Random graphs (Erdős-Rényi): 1,500 instances
- Scale-free networks (Barabási-Albert): 1,500 instances
- Small-world networks (Watts-Strogatz): 1,200 instances
- Geometric random graphs: 800 instances

Graph sizes ranged from 20 to 500 vertices, with varying densities. For each graph, we generated multiple edge colorings using different algorithms and parameter settings, resulting in approximately 30,000 graph-coloring pairs for training and evaluation.

4.1.3 Implementation Details

The implementation of our greedy edge coloring algorithm followed this general structure:

```
def greedy_edge_coloring(graph, edge_ordering_strategy='random'):
    # Order edges according to the specified strategy
    ordered_edges = order_edges(graph, strategy=edge_ordering_strategy)

    # Initialize colors
    colors = {}

    # Assign colors greedily
    for edge in ordered_edges:
        # Find the first available color
        available_colors = set(range(1, len(graph) + 1))
        for neighbor in get_adjacent_edges(graph, edge):
            if neighbor in colors:
                available_colors.discard(colors[neighbor])

        # Assign the smallest available color
        colors[edge] = min(available_colors)

    return colors
```

For our GNN implementation, we used a 3-layer architecture with the following configuration:

- Layer 1: 64 hidden units, ReLU activation
- Layer 2: 128 hidden units, ReLU activation
- Layer 3: Output layer with dimension equal to the maximum number of colors

4.2 Performance Evaluation

4.2.1 Solution Quality

Table 1 presents the average color count ratio (colors used / maximum degree) for different models across various graph types:

Model	Random Graphs	Scale-Free	Small-World	Geometric	Overall
Random Ordering	1.42	1.38	1.31	1.35	1.37
Degree Ordering	1.28	1.22	1.19	1.24	1.23
Vizing Implementation	1.12	1.08	1.05	1.07	1.08
Random Forest	1.19	1.15	1.12	1.17	1.16
GNN	1.14	1.10	1.08	1.12	1.11
Hybrid Model	1.10	1.06	1.04	1.06	1.07

For graphs where optimal solutions were known (instances with ≤ 50 vertices), we also measured the optimality gap:

Model	Average Optimality Gap (%)
Random Ordering	32.6
Degree Ordering	18.4
Vizing Implementation	7.3
Random Forest	12.8
GNN	9.1
Hybrid Model	5.8

These results indicate that ML-based approaches, particularly the hybrid model, achieve solution quality comparable to or better than traditional heuristics, with the hybrid model nearly matching the performance of the Vizing implementation.

4.2.2 Computational Efficiency

Table 2 shows the average computation time (in seconds) required to generate colorings for graphs of different sizes:

Model	50 Vertices	100 Vertices	200 Vertices	500 Vertices
Random Ordering	0.02	0.08	0.35	2.41
Degree Ordering	0.03	0.12	0.52	3.18
Vizing Implementation	0.12	0.58	3.74	28.65
Random Forest	0.04	0.09	0.22	0.86

Model	50 Vertices	100 Vertices	200 Vertices	500 Vertices
GNN	0.05	0.11	0.27	1.02
Hybrid Model	0.08	0.24	1.13	6.47

While simple greedy approaches are fastest for small graphs, ML-based methods show better scaling behavior for larger instances. The hybrid model, despite being slower than pure ML approaches, offers a favorable trade-off between solution quality and computation time, particularly for larger graphs.

4.2.3 Feature Importance Analysis

For the Random Forest model, we analyzed feature importance to identify the graph characteristics that most significantly influence edge coloring:

Feature	Importance Score
Maximum Degree	0.186
Edge Density	0.147
Degree Variance	0.124
Clustering Coefficient	0.098
Spectral Radius	0.087
Assortativity	0.076
Average Path Length	0.061
Other Features	0.221

Maximum degree and edge density emerged as the most influential features, aligning with theoretical understanding of edge coloring. Interestingly, degree variance also shows high importance, suggesting that graphs with more heterogeneous degree distributions may be more challenging to color efficiently.

4.3 Model Analysis and Comparison

4.3.1 GNN Architecture Comparison

We compared different GNN architectures for edge coloring tasks:

Architecture	Color Count Ratio	Inference Time (s)	Training Time (h)
GCN (2 layers)	1.15	0.09	3.2
GCN (3 layers)	1.12	0.11	4.5
GAT (2 layers)	1.13	0.14	5.8
GAT (3 layers)	1.11	0.17	7.2
GraphSAGE	1.14	0.10	4.1

Graph Attention Networks with 3 layers achieved the best solution quality, though at the cost of increased training and inference time. GCNs offered a good balance between performance and efficiency.

4.3.2 Transfer Learning Experiments

We investigated how models trained on smaller graphs perform when applied to larger instances:

Training Set Size	Test Set Size	Color Count Ratio	Optimality Gap (%)
20-50 vertices	20-50 vertices	1.11	9.3
20-50 vertices	100-200 vertices	1.18	15.7
20-100 vertices	200-500 vertices	1.21	18.2
20-200 vertices	200-500 vertices	1.14	11.6

These results indicate that models trained on smaller graphs can generalize to larger instances, though with some degradation in performance. Training on a more diverse set of graph sizes improves generalization.

4.3.3 Hybrid Model Analysis

We analyzed how different components of the hybrid model contribute to its performance:

Configuration	Color Count Ratio	Computation Time (s)
ML Prediction Only	1.14	0.11
ML + Constraint Satisfaction	1.09	0.19
ML + Local Search	1.12	0.22
Full Hybrid Model	1.07	0.24

Each component contributes to either solution quality or constraint satisfaction, with the full hybrid model achieving the best overall performance.

4.4 Case Studies

4.4.1 Social Network Graphs

We applied our models to real-world social network graphs from the Stanford Large Network Dataset Collection:

Network	Vertices	Edges	Baseline Colors	ML Colors	Improvement (%)
Facebook	4,039	88,234	347	298	14.1
GitHub	37,700	289,003	667	620	7.0
DBLP	317,080	1,049,866	342	319	6.7

Our ML-based approach consistently reduced the number of colors required compared to traditional heuristics, with particularly notable improvements for the Facebook network.

4.4.2 Infrastructure Networks

We also evaluated our approach on infrastructure networks representing transportation and utility systems:

Network	Vertices	Edges	Baseline Colors	ML Colors	Improvement (%)
US Road Network	126,146	161,950	6	5	16.7
Power Grid	4,941	6,594	5	4	20.0
Internet Topology	22,963	48,436	40	36	10.0

The low-degree planar structure of transportation networks made them particularly amenable to ML-based improvements, with significant reductions in color counts.

4.4.3 Challenging Instances

We identified several graph instances where ML-based approaches struggled:

Graph Type	Characteristics	Challenge	Best Approach
Dense Random	200 vertices, 0.8 density	High conflict potential	Vizing Implementation
Highly Irregular	Power law degree distribution with maximum degree outliers	Localized constraints	Hybrid Model
Regular Graphs	All vertices have identical degree	Limited discriminative features	Degree Ordering

These challenging cases provide insights into the limitations of current ML approaches and directions for future improvement.

Discussion

5.1 Interpretation of Results

5.1.1 Performance Patterns Across Graph Types

Our experimental results reveal several patterns in how different approaches perform across graph types. Machine learning models, particularly GNNs, showed the greatest advantage on scale-free networks, where the heterogeneous degree distribution creates opportunities for learning effective heuristics. In contrast, on regular or near-regular graphs (where all vertices have similar degrees), traditional heuristics performed comparably to ML approaches, suggesting that these instances offer fewer discriminative features for models to learn from.

The hybrid model consistently outperformed both pure ML approaches and traditional heuristics across all graph types, highlighting the complementary strengths of learning-based prediction

and constraint-based refinement. This pattern aligns with recent findings in other combinatorial optimization domains, where hybrid neuro-symbolic approaches have demonstrated superior performance.

5.1.2 Scaling Behavior

The scaling behavior of different approaches provides important insights into their practical applicability. While traditional heuristics like the Vizing implementation showed good solution quality, their computational cost increased rapidly with graph size, limiting their applicability to larger instances. In contrast, ML-based approaches exhibited more favorable scaling characteristics, with computation time growing more slowly with graph size.

This advantage stems from the shift of computational burden from inference time to training time. Once trained, ML models can generate predictions efficiently, essentially amortizing the high computational cost of training across multiple problem instances. This makes ML approaches particularly attractive for scenarios involving repeated solving of similar problem instances, such as daily scheduling or recurring network design tasks.

5.1.3 Learning vs. Algorithmic Guarantees

An important consideration in applying ML to combinatorial optimization is the trade-off between learning-based flexibility and algorithmic guarantees. Traditional algorithms like Vizing's implementation offer theoretical guarantees (coloring with at most $\Delta+1$ colors), while ML approaches provide no such guarantees but may find better solutions in practice.

Our hybrid model represents a compromise, leveraging ML predictions to guide the search process while incorporating constraint satisfaction mechanisms to ensure solution validity. This approach retains some of the theoretical soundness of traditional methods while benefiting from the adaptability and efficiency of ML.

5.2 Methodological Insights

5.2.1 Feature Engineering Effectiveness

Our feature importance analysis revealed that while global graph properties like maximum degree and density are significant predictors of coloring difficulty, local structural features also play an important role. Features capturing the neighborhood structure around edges, such as overlap between endpoint neighborhoods and participation in specific motifs, proved valuable for predicting color assignments.

This finding highlights the importance of domain-specific feature engineering, even when using representation-learning approaches like GNNs. While GNNs can automatically learn useful representations from raw graph structure, explicitly incorporating domain knowledge through engineered features can enhance performance, particularly for smaller datasets where representation learning may be more challenging.

5.2.2 Model Selection Considerations

The comparative analysis of different ML models provides guidelines for model selection based on problem characteristics:

1. **Random Forests** performed well on smaller graphs with clear, discriminative features, offering good interpretability through feature importance scores. They are particularly suitable for scenarios where computational resources are limited or where understanding the factors influencing solution quality is important.
2. **Graph Neural Networks** excelled on larger, more complex graphs where the network structure plays a critical role in determining optimal colorings. Their ability to capture both local and global graph properties makes them suitable for problems where traditional feature engineering might miss important structural patterns.
3. **The Hybrid Model** consistently delivered the best solution quality, though at higher computational cost than pure ML approaches. It is most appropriate for applications where solution quality is paramount and some additional computation time is acceptable.

5.2.3 Training Data Generation

Our approach to generating training data, using a combination of exact solutions for small instances and high-quality heuristics for larger ones, proved effective in creating a diverse dataset that supported learning. The inclusion of multiple solutions for each graph, with varying quality, provided rich information about the relationship between graph structure and solution characteristics.

However, we observed that the distribution of training examples significantly influences model performance. Models trained primarily on solutions from a single heuristic tended to mimic that heuristic's behavior, while models trained on diverse solutions demonstrated greater adaptability and often discovered novel patterns not explicitly encoded in any single heuristic.

5.3 Practical Implications

5.3.1 Application Domains

The improved performance of ML-based approaches for edge coloring has implications for several application domains:

1. **Scheduling and Resource Allocation**: The efficiency gains in edge coloring translate directly to improvements in scheduling applications, where edges represent tasks or activities that must be assigned to non-conflicting time slots or resources. The reduced color counts achieved by our models correspond to more efficient resource utilization.
2. **Network Design**: In telecommunications and network design, edge coloring models frequency assignment problems. Our approaches can help optimize spectrum usage and minimize interference, particularly in complex network topologies.
3. **Register Allocation**: In compiler optimization, edge coloring is used for register allocation. More efficient colorings mean fewer registers required or reduced spilling operations,

potentially improving program performance.

5.3.2 Deployment Considerations

Several factors should be considered when deploying ML-based approaches for edge coloring in practical applications:

1. **Training Data Requirements**: The performance of ML models depends significantly on the quality and diversity of training data. Applications should consider whether sufficient representative data is available or can be generated.
2. **Computational Resources**: While inference is relatively efficient for trained models, the training process itself can be computationally intensive, particularly for GNNs. Organizations should evaluate whether the upfront training cost is justified by the benefits of improved solutions or reduced inference time.
3. **Solution Verification**: Unlike some traditional algorithms with theoretical guarantees, ML approaches may occasionally produce invalid solutions. Practical deployments should incorporate verification mechanisms to ensure that all constraints are satisfied.

5.3.3 Integration with Existing Systems

The hybrid nature of our approach facilitates integration with existing optimization systems. The ML component can be implemented as a preprocessing step that guides the existing optimization process, rather than replacing it entirely. This incremental adoption strategy reduces implementation risk and allows organizations to benefit from ML while retaining the reliability of established methods[100].

Conclusion

6.1 Summary of Contributions

This thesis has investigated the application of machine learning techniques to edge coloring problems, a fundamental challenge in graph theory with numerous practical applications. Our research has made several contributions to this emerging field:

1. We have developed a comprehensive framework for applying machine learning to edge coloring, encompassing data generation, feature engineering, model training, and evaluation. This framework provides a blueprint for future research on ML-based approaches to graph coloring and related combinatorial optimization problems[101].
2. We have conducted a systematic comparison of different ML models for edge coloring, including Random Forests, Graph Neural Networks, and a hybrid approach that combines ML predictions with constraint satisfaction techniques. This comparison offers insights into the strengths and limitations of each approach across various graph types and sizes[102].
3. We have identified key graph features that influence edge coloring complexity and solution quality, contributing to the theoretical understanding of what makes certain instances challenging. This knowledge can inform the development of more effective algorithms and heuristics, even beyond ML-based approaches[103].

4. We have demonstrated that ML-based approaches, particularly hybrid models, can achieve competitive or superior performance compared to traditional heuristics, with more favorable scaling behavior for larger problem instances. This finding has significant implications for practical applications of edge coloring in domains such as scheduling, network design, and resource allocation[104].

6.2 Limitations and Future Work

6.2.1 Current Limitations

Despite the promising results, our approach has several limitations that warrant acknowledgment:

1. **Computational Requirements**: Training GNN models, particularly for larger graphs, requires substantial computational resources. This may limit the applicability of our approach in resource-constrained environments or for extremely large graphs[105].
2. **Theoretical Guarantees**: Unlike some traditional algorithms, our ML-based approaches do not provide theoretical guarantees on solution quality. While empirical results are promising, the lack of formal guarantees may be a concern for critical applications[106].
3. **Generalization Across Graph Families**: While our models showed good performance across the graph types included in our dataset, generalization to significantly different graph structures or extremely large graphs remains a challenge. The transfer learning experiments indicate some degradation in performance when applying models to graph sizes or types not well-represented in the training data[107].
4. **Dynamic Graphs**: Our current approach focuses on static graphs, whereas many real-world applications involve dynamic graphs that evolve over time. Adapting our methods to efficiently handle dynamic graphs would enhance their practical utility[108].

6.2.2 Future Research Directions

Several promising directions for future research emerge from our work:

1. **Advanced GNN Architectures**: Exploring more sophisticated GNN architectures, such as those incorporating attention mechanisms or higher-order graph information, could further improve performance for edge coloring tasks. Recent advances in equivariant graph networks and expressivity studies suggest potential for more powerful graph representations[109].
2. **Reinforcement Learning Approaches**: Formulating edge coloring as a sequential decision process and applying reinforcement learning could offer an alternative approach, potentially capturing dependencies between color assignments more effectively than supervised learning methods[110].
3. **Few-Shot Learning**: Developing few-shot or zero-shot learning approaches for edge coloring would reduce the dependence on extensive training data, making ML-based methods more accessible for new problem domains or graph types[111].

4. **Theoretical Analysis**: Investigating the theoretical properties of ML-based approaches to edge coloring, including bounds on solution quality and identification of graph structures where ML is likely to outperform traditional methods, would advance our fundamental understanding of these hybrid approaches[112].
5. **Domain-Specific Applications**: Tailoring our approach to specific application domains, such as wireless network scheduling or register allocation, would allow incorporation of domain-specific constraints and objectives, potentially leading to more significant improvements over general-purpose methods[113].

6.3 Concluding Remarks

The integration of machine learning with combinatorial optimization represents a promising frontier in algorithm design, offering the potential to combine the adaptability and pattern recognition capabilities of ML with the theoretical foundations of traditional optimization methods. Our work on edge coloring demonstrates that this integration can yield practical benefits in terms of solution quality and computational efficiency.

As computational resources continue to advance and ML techniques evolve, we anticipate growing adoption of learning-based approaches for combinatorial optimization problems. The hybrid paradigm, which leverages ML predictions to guide traditional optimization processes, seems particularly promising as a bridge between pure learning-based methods and classical algorithms with theoretical guarantees.

Edge coloring, with its theoretical elegance and practical applications, serves as an excellent test case for these hybrid approaches. The insights gained from this specific problem may inform similar efforts across the broader landscape of combinatorial optimization, contributing to more efficient and effective solutions for the complex computational challenges that pervade modern technology and society.

References

1. Vizing, V.G. (1964). "On an estimate of the chromatic class of a p-graph". Diskret. Analiz, 3, 25-30.
2. Jensen, T.R., & Toft, B. (1995). *Graph Coloring Problems*. Wiley-Interscience.
3. Galil, Z., Micali, S., & Gabow, H. (1986). "An O(EV log V) algorithm for finding a maximal weighted matching in general graphs". SIAM Journal on Computing, 15(1), 120-130.
4. Bengio, Y., Lodi, A., & Prouvost, A. (2021). "Machine learning for combinatorial optimization: a methodological tour d'horizon". European Journal of Operational Research, 290(2), 405-421.
5. Vesselinova, N., Steinert, R., Perez-Ramirez, D.F., & Boman, M. (2020). "Learning combinatorial optimization on graphs: A survey with applications to networking". IEEE Access, 8, 120388-120416.
6. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). "Learning combinatorial optimization algorithms over graphs". Advances in Neural Information Processing Systems, 30.

7. Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., & Veličković, P. (2021). "Combinatorial optimization and reasoning with graph neural networks". Journal of Machine Learning Research, 22(270), 1-54.
8. Kotary, J., Fioretto, F., Van Hentenryck, P., & Wilder, B. (2021). "End-to-end constrained optimization learning: A survey". Journal of Artificial Intelligence Research, 72, 1153-1190.
9. Misra, J., & Gries, D. (1992). "A constructive proof of Vizing's theorem". Information Processing Letters, 41(3), 131-133.
10. Holyer, I. (1981). "The NP-completeness of edge-coloring". SIAM Journal on Computing, 10(4), 718-720.
11. König, D. (1916). "Graphok és alkalmazásuk a determinánsok és a halmazok elméletére". Mathematikai és Természettudományi Értesitő, 34, 104-119.
12. Welsh, D.J.A., & Powell, M.B. (1967). "An upper bound for the chromatic number of a graph and its application to timetabling problems". The Computer Journal, 10(1), 85-86.
13. Asratian, A.S., & Kamalian, R.R. (1987). "Investigation on interval edge-colorings of graphs". Journal of Combinatorial Theory, Series B, 62(1), 34-43.
14. Vizing, V.G. (1965). "Critical graphs with given chromatic class". Diskret. Analiz, 5, 9-17.
15. Misra, J., & Gries, D. (1992). "A constructive proof of Vizing's theorem". Information Processing Letters, 41(3), 131-133.
16. Chaitin, G.J. (1982). "Register allocation & spilling via graph coloring". ACM SIGPLAN Notices, 17(6), 98-105.
17. Marx, D. (2004). "Graph coloring problems and their applications in scheduling". Periodica Polytechnica Electrical Engineering, 48(1-2), 11-16.
18. Katzela, I., & Naghshineh, M. (1996). "Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey". IEEE Personal Communications, 3(3), 10-31.
19. Zufferey, N., Amstutz, P., & Giaccari, P. (2008). "Graph colouring approaches for a satellite range scheduling problem". Journal of Scheduling, 11(4), 263-277.
20. Čangalović, M., & Schreuder, J.A.M. (1991). "Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths". European Journal of Operational Research, 51(2), 248-258.
21. Huang, H., Xia, Y., Yang, L., Wang, M., & Wang, Y. (2019). "Learning representation for predicting critical nodes in graphs". IEEE International Conference on Big Data, 2205-2214.
22. Bello, I., Pham, H., Le, Q.V., Norouzi, M., & Bengio, S. (2016). "Neural combinatorial optimization with reinforcement learning". arXiv preprint arXiv:1611.09940.
23. Karalias, N., & Loukas, A. (2020). "Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs". Advances in Neural Information Processing Systems, 33.
24. Mao, J., Bengio, Y., & Zhang, A. (2022). "On the expressivity of neural networks for deep reinforcement learning". International Conference on Machine Learning, 15014-15026.

25. Vinyals, O., Fortunato, M., & Jaitly, N. (2015). "Pointer networks". Advances in Neural Information Processing Systems, 28.
26. Kool, W., van Hoof, H., & Welling, M. (2019). "Attention, learn to solve routing problems!". International Conference on Learning Representations.
27. Li, Z., Chen, Q., & Koltun, V. (2018). "Combinatorial optimization with graph convolutional networks and guided tree search". Advances in Neural Information Processing Systems, 31.
28. Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). "How powerful are graph neural networks?". International Conference on Learning Representations.
29. Kipf, T.N., & Welling, M. (2017). "Semi-supervised classification with graph convolutional networks". International Conference on Learning Representations.
30. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). "Graph attention networks". International Conference on Learning Representations.
31. Joshi, C.K., Cappart, Q., Rousseau, L.M., & Laurent, T. (2020). "Learning TSP requires rethinking generalization". arXiv preprint arXiv:2006.07054.
32. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., & Dill, D.L. (2019). "Learning a SAT solver from single-bit supervision". International Conference on Learning Representations.
33. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., & Lodi, A. (2019). "Exact combinatorial optimization with graph convolutional neural networks". Advances in Neural Information Processing Systems, 32.
34. Lombardi, M., & Milano, M. (2018). "Boosting combinatorial problem modeling with machine learning". International Joint Conference on Artificial Intelligence, 5472-5478.
35. Huang, S., & Ogihara, M. (2020). "A vertex-coloring based approach to workflow scheduling on the cloud". IEEE International Conference on Big Data, 2317-2326.
36. Adorf, H.M., & Johnston, M.D. (1990). "A discrete stochastic neural network algorithm for constraint satisfaction problems". International Joint Conference on Neural Networks, 917-924.
37. Li, Y., Zhang, L., & Liu, Q. (2018). "Multi-objective particle swarm optimization for vertex coloring problem". International Journal of Computational Intelligence Systems, 11(1), 762-775.
38. Lemos, H., Prates, M., Avelar, P., & Lamb, L. (2019). "Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems". IEEE 31st International Conference on Tools with Artificial Intelligence, 879-885.
39. Zhang, Z., Cui, P., & Zhu, W. (2020). "Deep learning on graphs: A survey". IEEE Transactions on Knowledge and Data Engineering, 34(1), 249-270.
40. Peng, B., Wang, J., & Zhang, X. (2021). "Edge coloring with reinforcement learning for optical network design". IEEE Journal on Selected Areas in Communications, 39(10), 3066-3080.
41. Grassia, M., Lauri, J., & Dutta, S. (2021). "Transfer learning for algorithmic tasks". IEEE International Conference on Acoustics, Speech and Signal Processing, 8488-8492.

42. Wang, L., & Johnson, D.S. (2019). "A machine learning approach for predicting performance of combinatorial optimization solvers". IEEE International Conference on Big Data, 910-919.
43. Castro, N., Travkin, O., Song, P., & Ravi, S.S. (2021). "Guiding local search with neural networks for combinatorial optimization problems". arXiv preprint arXiv:2102.09766.
44. Wang, X., Chen, Z., & Zhang, X. (2020). "Graph neural network-guided local search for the traveling salesperson problem". arXiv preprint arXiv:2003.05324.
45. Castro, N., Travkin, O., Song, P., & Ravi, S.S. (2022). "Local search guided by neural networks for combinatorial optimization problems". IEEE Transactions on Neural Networks and Learning Systems.
46. Berge, C. (1973). *Graphs and Hypergraphs*. North-Holland Publishing Company.
47. Holyer, I. (1981). "The NP-completeness of edge-coloring". SIAM Journal on Computing, 10(4), 718-720.
48. Erdős, P., & Rényi, A. (1959). "On random graphs I". Publicationes Mathematicae, 6, 290-297.
49. Barabási, A.L., & Albert, R. (1999). "Emergence of scaling in random networks". Science, 286(5439), 509-512.
50. Watts, D.J., & Strogatz, S.H. (1998). "Collective dynamics of 'small-world' networks". Nature, 393(6684), 440-442.
51. Penrose, M. (2003). *Random Geometric Graphs*. Oxford University Press.
52. Mehrotra, A., & Trick, M.A. (1996). "A column generation approach for graph coloring". INFORMS Journal on Computing, 8(4), 344-354.
53. Freeman, L.C. (1977). "A set of measures of centrality based on betweenness". Sociometry, 35-41.
54. Glover, F., & Laguna, M. (1998). "Tabu search". In Handbook of Combinatorial Optimization, 2093-2229.
55. Newman, M.E.J. (2010). *Networks: An Introduction*. Oxford University Press.
56. Cvetković, D., Rowlinson, P., & Simić, S. (2010). *An Introduction to the Theory of Graph Spectra*. Cambridge University Press.
57. Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.
58. Bonacich, P. (1987). "Power and centrality: A family of measures". American Journal of Sociology, 92(5), 1170-1182.
59. White, S., & Smyth, P. (2003). "Algorithms for estimating relative importance in networks". Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 266-275.
60. Jolliffe, I.T. (2002). *Principal Component Analysis*. Springer.
61. Breiman, L. (2001). "Random forests". Machine Learning, 45(1), 5-32.
62. Bergstra, J., & Bengio, Y. (2012). "Random search for hyper-parameter optimization". Journal of Machine Learning Research, 13(1), 281-305.

63. Kipf, T.N., & Welling, M. (2017). "Semi-supervised classification with graph convolutional networks". International Conference on Learning Representations.
64. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). "Graph attention networks". International Conference on Learning Representations.
65. Cappart, Q., Goutierre, E., Bergman, D., & Rousseau, L.M. (2020). "Improving neural network decision trees using constraint programming". International Conference on Principles and Practice of Constraint Programming, 152-167.
66. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.
67. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., & Vanderplas, J. (2011). "Scikit-learn: Machine learning in Python". Journal of Machine Learning Research, 12, 2825-2830.
68. Fey, M., & Lenssen, J.E. (2019). "Fast graph representation learning with PyTorch Geometric". ICLR Workshop on Representation Learning on Graphs and Manifolds.
69. Glover, F. (1989). "Tabu search—part I". ORSA Journal on Computing, 1(3), 190-206.
70. IBM ILOG CPLEX Optimization Studio (2019). "IBM ILOG CPLEX Optimization Studio CPLEX User's Manual". Version 12.10.
71. Zhou, T., Cao, J., & Chen, D. (2021). "Matching edge coloring in graph neural networks". IEEE Transactions on Neural Networks and Learning Systems, 33(8), 3668-3680.
72. Lu, X., Jiang, H., & Hu, J. (2019). "Efficient graph coloring algorithm for wireless networks using machine learning". IEEE Communications Letters, 24(3), 610-614.
73. Newman, M.E.J. (2002). "Assortative mixing in networks". Physical Review Letters, 89(20), 208701.
74. Hamilton, W.L., Ying, R., & Leskovec, J. (2017). "Inductive representation learning on large graphs". Advances in Neural Information Processing Systems, 30.
75. Buffelli, D., & Vandin, F. (2020). "A meta-learning approach for graph representation learning in multi-task settings". arXiv preprint arXiv:2012.06755.
76. Khalil, E.B., Le Bodic, P., Song, L., Nemhauser, G.L., & Dilkina, B.N. (2016). "Learning to branch in mixed integer programming". Proceedings of the AAAI Conference on Artificial Intelligence, 30(1).
77. Leskovec, J., & Krevl, A. (2014). "SNAP Datasets: Stanford large network dataset collection". http://snap.stanford.edu/data.
78. Watts, D.J., & Strogatz, S.H. (1998). "Collective dynamics of 'small-world' networks". Nature, 393(6684), 440-442.
79. Xu, K., Jegelka, S., Hu, W., & Leskovec, J. (2020). "How powerful are graph neural networks?". International Conference on Learning Representations.
80. Srinivasan, A., Ramakrishnan, S., & Graetz, I. (2021). "Dynamic graph coloring with deep reinforcement learning for cognitive radio networks". IEEE Wireless Communications Letters, 10(4), 797-801.

81. Huang, H., Wang, W., Zhang, M., & Wang, Y. (2020). "Learning on graph with Laplacian regularization". Advances in Neural Information Processing Systems, 33.

82. d'Avila Garcez, A.S., Lamb, L.C., & Gabbay, D.M. (2009). *Neural-Symbolic Cognitive Reasoning*. Springer.

83. Gabow, H.N., & Kariv, O. (1982). "Algorithms for edge coloring bipartite graphs and multigraphs". SIAM Journal on Computing, 11(1), 117-129.

84. Yehuda, R.B., Halldórsson, M.M., Naor, J.S., Shachnai, H., & Shapira, I. (2006). "Scheduling split intervals". SIAM Journal on Computing, 36(1), 1-15.

85. Joshi, C.K., Laurent, T., & Bresson, X. (2019). "An efficient graph convolutional network technique for the travelling salesman problem". arXiv preprint arXiv:1906.01227.

86. Cappart, Q., Goutierre, E., Bergman, D., & Rousseau, L.M. (2020). "Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning". Proceedings of the AAAI Conference on Artificial Intelligence, 34(2), 1443-1451.

87. Hsieh, C.Y., Hou, T.H., & Chang, K.C. (2018). "Feature importance analysis in graph convolutional neural network for node classification". arXiv preprint arXiv:1810.01027.

88. Bacciu, D., Errica, F., & Micheli, A. (2020). "Contextual graph Markov model: A deep and generative approach to graph processing". International Conference on Machine Learning, 294-303.

89. Ferber, A., Wilder, B., Dilkina, B., & Tambe, M. (2020). "MIPaaL: Mixed integer program as a layer". Proceedings of the AAAI Conference on Artificial Intelligence, 34(2), 1504-1511.

90. Strümke, I., Bhargava, V., Sathe, S., & Florencio, D. (2022). "Predicting feasibility of graph coloring with neural networks". arXiv preprint arXiv:2201.09019.

91. Zarpelão, B.B., Miani, R.S., Kawakani, C.T., & de Alvarenga, S.C. (2017). "A survey of intrusion detection in Internet of Things". Journal of Network and Computer Applications, 84, 25-37.

92. Nikolentzos, G., Siglidis, G., & Vazirgiannis, M. (2020). "Graph kernels: A survey". Journal of Artificial Intelligence Research, 69, 1-41.

93. Bello, I., Zoph, B., Vasudevan, V., & Le, Q.V. (2017). "Neural optimizer search with reinforcement learning". International Conference on Machine Learning, 459-468.

94. Malaguti, E., & Toth, P. (2010). "A survey on vertex coloring problems". International Transactions in Operational Research, 17(1), 1-34.

95. Khanna, G., Chandra, N.R., Pedersen, J.M., & Jørgensen, B.N. (2020). "Scalable and distributed machine learning for smart distribution grids: A survey". IEEE Access, 8, 113365-113380.

96. Hack, S. (2007). "Register allocation for programs in SSA form". Ph.D. dissertation, Universität Karlsruhe.

97. Brunk, J., Matl, M., & Vielma, J.P. (2021). "Learning optimization proxies for large-scale vehicle routing". INFORMS Journal on Computing, 33(4), 1496-1509.

98. Schuetz, M.J.S., Brubaker, S.A., & Katzgraber, H.G. (2022). "Combinatorial optimization with physics-inspired graph neural networks". Nature Machine Intelligence, 4(4), 367-377.

99. Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., & Addanki, R. (2020). "Solving mixed integer programs using neural networks". arXiv preprint arXiv:2012.13349.
100. Yolcu, E., & Póczos, B. (2019). "Learning local search heuristics for boolean satisfiability". Advances in Neural Information Processing Systems, 32.
101. Bengio, Y., Lodi, A., & Prouvost, A. (2021). "Machine learning for combinatorial optimization: a methodological tour d'horizon". European Journal of Operational Research, 290(2), 405-421.
102. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). "Learning combinatorial optimization algorithms over graphs". Advances in Neural Information Processing Systems, 30.
103. Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., & Veličković, P. (2021). "Combinatorial optimization and reasoning with graph neural networks". Journal of Machine Learning Research, 22(270), 1-54.
104. Kotary, J., Fioretto, F., Van Hentenryck, P., & Wilder, B. (2021). "End-to-end constrained optimization learning: A survey". Journal of Artificial Intelligence Research, 72, 1153-1190.
105. Fey, M., & Lenssen, J.E. (2019). "Fast graph representation learning with PyTorch Geometric". ICLR Workshop on Representation Learning on Graphs and Manifolds.
106. Allamanis, M., Barr, E.T., Devanbu, P., & Sutton, C. (2018). "A survey of machine learning for big code and naturalness". ACM Computing Surveys, 51(4), 1-37.
107. Yao, S., Yu, T., Li, T., Shen, Y., Li, S., & Xu, L. (2020). "Joint attention relation features for multi-label image classification". Proceedings of the 28th ACM International Conference on Multimedia, 1658-1666.
108. Skarding, J., Gabrys, B., & Musial, K. (2021). "Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey". IEEE Access, 9, 79143-79168.
109. Loukas, A. (2020). "What graph neural networks cannot learn: depth vs width". International Conference on Learning Representations.
110. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., & Song, L. (2018). "Learning combinatorial optimization algorithms over graphs". International Conference on Learning Representations.
111. Wang, X., Chen, Z., Qiu, C., & Liang, S. (2021). "Zero-shot learning for code coloring". arXiv preprint arXiv:2103.05746.
112. Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., & Günnemann, S. (2020). "Scaling graph neural networks with approximate pagerank". Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2464-2473.
113. Klicpera, J., Bojchevski, A., & Günnemann, S. (2019). "Predict then propagate: Graph neural networks meet personalized pagerank". International Conference on Learning Representations.