State of the art solutions for the Graph coloring problem

Lungu Andrei, Isac Lucian, Fomin Bogdan

March 2025

1 GNN-GCP (2019)

Graph Neural Network architecture to solve the decision version of the graph coloring problem (GCP) (i.e. "is it possible to colour the given graph with C colours?"). [4]

1.1 Introduction to GNN and embeddings

GNNs excel in applications like social networks, molecular modeling, and recommendation systems, where understanding the structure and connections between entities is crucial. Description of a graph neural network architecture:

1. **Nodes** represent entities, and **edges** represent their relationships.
2. Each node has **features**, and the network learns by iteratively updating these features through a process called **message passing**.
3. In each step, nodes gather information from their neighbors, aggregate it, and update their own state.
4. After several iterations, the updated node features can be used for tasks like **node classification**, link prediction, or **graph classification**.

Input: GCP instance I = (G, C) made up of a graph G = (V,E) and a number of colors $C \in N \mid C > 2$, each color is assigned to a random initial embedding on an object. The same idea is applied to vertex embeddings. They are both assigned to a random initial embedding over a uniform distribution.

Example of color embedding: *Red=[0.12,0.85],Green=[0.45,0.33],Blue=[0.78,0.56]*
An update of this embedding might look like: NewRed=W*Red+U*(Green+Blue)
Vertex embeddings may have the following:

1. Its degree (number of neighbors)
2. Any initial color or label information (if provided)

3. Other domain-specific features (e.g., whether the node is part of a particular subset of vertices)

Example of vertex embeddings:
vertex1 = [degree1, color1]
vertex2 = [degree2, color2]

1.2 Algorithm and training methodology

Initialization steps:

- Assign a random embedding to each color from a uniform distribution.
- Assign a random embedding to each vertex from a uniform distribution.
- Compute binary adjacency matrix from vertex to vertex
- Compute binary adjacency matrix from vertices to colours

Message-passing procedure:

- Run t_{max} message passing iterations
- For each vertex: Aggregate embeddings from neighboring vertices. Randomly pick a color and get its embedding. Update the vertex embedding as a combination of: Its previous embedding (self-influence). The aggregated embeddings from neighbors (neighbor influence). The color embedding (color influence).
- Refine each colour embedding with messages received from all vertices

Output:

- Translate vertex embeddings into logit probabilities.
- Average logits and translate to probability

To train these message computing and updating modules, MLPs and RNNs respectively, the authors used the Stochastic Gradient Descent algorithm implemented via TensorFlow's Adam optimiser. They defined as loss the binary cross entropy between the model final prediction and the ground-truth for a given GCP instance – a boolean value indicating that the graph accepts (or not) the target colorability. Upon the initialisation, the model is defined with 64-dimensional embeddings for vertices and colours. The MLPs responsible for message com- puting are three-layered (64,64,64) with ReLU nonlinearities as the activations for all layers except for the linear activation on the output layer. The RNN cells are basic LSTM cells with layer normalisation and ReLu activation.

2 ReLCol (2023)

"*Our proposed approach, ReLCol, uses* ***deep Q-learning*** *together with a graph neural network for feature extraction, and employs a novel way of parameterising the graph that results in improved performance. Using standard benchmark graphs with varied topologies, we empirically evaluate the benefits and limitations of the* ***heuristic learned*** *by ReLCol relative to existing construction algorithms, and demonstrate that reinforcement learning is a promising direction for further research on the graph colouring problem.*"[1]

2.1 GCP as a Markov Decision Process (MDP)

- States: Each state represents a partition of the graph's vertices into subsets, where each subset corresponds to a specific colour, and one subset contains uncoloured vertices. A state is terminal when all vertices are coloured.
- Actions: An action involves selecting an uncoloured vertex to assign the next colour.
- Transition Function: Upon selecting a vertex, it is assigned the lowest permissible colour that doesn't conflict with its neighbours, leading to a new state. This function is deterministic.
- Reward Function: The reward is defined as the negative difference in the number of colours used between the new state and the previous state, encouraging the use of fewer colours.
- Discount Factor: A discount factor of 1 is used, as each episode (complete colouring of a graph) has a finite length equal to the number of vertices.

2.2 State Parameterization

The state is represented as a complete graph where:

- Vertices: Each vertex retains its original identity and is associated with a feature indicating its current colour (or lack thereof).
- Edges: Edges are assigned a binary feature indicating whether they existed in the original graph. This complete graph representation allows information to flow between all pairs of vertices, enhancing the GNN's ability to learn effective embeddings.

2.3 Q-Network Architecture

The Q-network comprises multiple stacked GNN blocks followed by fully connected layers with ReLU activations. The GNN blocks utilize Principal Neighbourhood Aggregation to effectively aggregate information from neighbouring

vertices and edges, producing embeddings that inform the action-value predictions for each vertex.

2.4 Action Selection Mechanism

While an ϵ-greedy policy is generally employed to balance exploration and exploitation, the following specific rules are applied to optimize performance:

- First Vertex Rule: The initial vertex to be coloured is selected randomly, ensuring that any vertex can start the colouring process without affecting the optimality of the solution.
- Isolated Vertices Rule: Vertices that have all their neighbours already coloured (isolated vertices) are immediately coloured, as their colour assignment doesn't impact the rest of the graph's colouring.

2.5 hyperparameters

The Q-network consists of 5 GNN blocks and 3 fully connected layers with weights initialised at random. The GNN blocksuse only edge and vertex features; we experimented with including global features but found no evidence of performance improvement. The vertex and edge embeddings, as well as the hidden layers in all fully connected neural networks, have 64 dimensions. The authors use the Adam optimiser with learning rate 0.001 and batch size 64, τ = 0.001, and an ϵ-greedy policy for exploration, where ϵ decays exponentially from 0.9 to 0.01 through 25000 episodes of training.

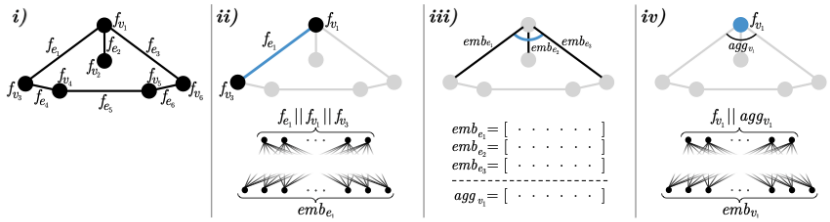

Fig. 1. Demonstration of how a GNN block generates edge and vertex embeddings. Blue indicates the element that is updated in that step. ***i)*** The input graph with edge and vertex features; ***ii)*** For each edge e, concatenate the features of e with the features of the vertices it connects, and pass the resulting vector through a small neural network to generate the edge embedding emb_e; ***iii)*** For each vertex v, aggregate the embeddings of the incident edges using an elementwise operation like *sum* or *max* to generate the edge aggregation agg_v; ***iv)*** For each vertex v, concatenate the features of v with the associated edge aggregation agg_v, and pass the resulting vector through a small neural network to generate the vertex embedding emb_v. Blocks can be stacked by repeating this process with the previous block's edge and vertex embeddings used as the features.

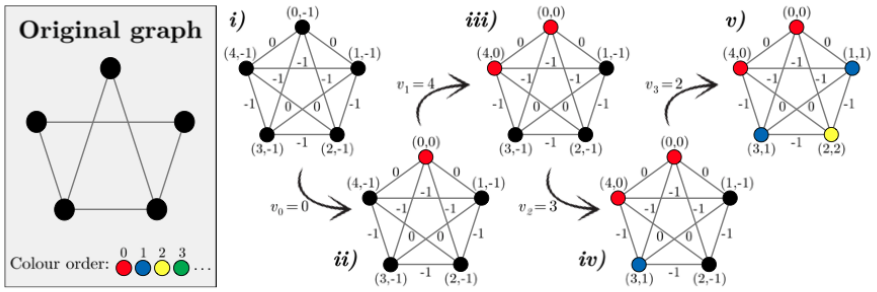

Fig. 2. Example of graph parameterisation and colouring rules, where $v_t = j$ indicates the vertex with name j is selected at step t. ***i)*** Initial parameterisation of the original graph; ***ii)*** First vertex $v_0 = 0$ is selected at random and assigned colour 0; ***iii)*** Vertex $v_1 = 4$ can also be assigned colour 0; ***iv)*** Vertex $v_2 = 3$ cannot be assigned colour 0 so takes colour 1; ***v)*** Vertex $v_3 = 2$ cannot be assigned colour 0 or 1 so takes colour 2, leaving vertex 1 isolated; vertex 1 cannot be assigned colour 0 so takes colour 1.

2.6 Output

The output of the model is a Q-value for each uncoloured vertex. This Q-value represents the "value" of choosing each vertex in the current state. The vertex with the highest Q-value is selected for colouring in each step, according to an ϵ-greedy policy (selecting the best action most of the time, but occasionally exploring other options).

After selecting a vertex:

- The vertex is coloured with the lowest available colour that doesn't conflict with its neighbours.
- The state is updated, and the process repeats until all vertices are coloured.

3 Deep-Learning & Memetic approaches

Using Deep Learning methods & metaheuristics for graph coloring to solve the minimization version of the graph coloring problem (GCP) (i.e. "which is the minimum number of colors such that any two adjacent vertices will have different colors") [2]

3.1 Deep-Learning: Overview

Deep neural networks are models designed to learn intricate patterns from extensive datasets. They excel in tasks such as image and speech recognition, natural language processing, and decision-making by extracting features from raw data that may not reveal direct correlations initially.

3.2 "Classical" metaheuristics for graph coloring

3.2.1 Local search

Local search involves iteratively improving a solution by making small changes and evaluating their impact. It starts from an initial solution and explores its neighborhood to find better solutions.
Techniques: Hill-climbing, and simulated annealing

3.2.2 Evolutionary Algorithms

Memetic algorithms combine the global search capabilities of genetic algorithms with local search methods. They use a population of solutions and apply multiple operators to refine individual solutions.
Techniques: Genetic / Memetic algorithms, using operators like selection & cross-over

3.2.3 STAGE Algorithm

The STAGE (Simultaneous Trajectory Analysis for Guiding Evolution) algorithm is a metaheuristic approach designed to enhance the performance of local search algorithms by learning from their search trajectories. This is more of an extension to local search techniques, as it uses a regression model to approximate the outcome of the local search.

3.3 Algorithm Structure

Step 1: Population Initialization

- Generate initial solutions using a randomized greedy algorithm.
- Use a large population ($|P| > 10^4$) for diversity.
- Initialize the neural network (f_θ) with random weights.

Step 2: Parallel Local Search

- Each individual undergoes iterated Tabu Search.
- Explores both feasible and infeasible solutions.
- Stores the best solution found.

Step 3: Deep Learning-Enhanced Crossover Selection (STAGE Algorithm Integration)

- STAGE Algorithm predicts good starting points for local search.
- Adapted with deep learning instead of classical regression.
- Neural network (f_θ) predicts the best crossover pairs.
- Guides genetic recombination toward better offspring.

Step 4: Distance-Based Population Update

- Ensures diversity by maintaining minimum spacing between individuals.
- Uses partition distance to measure solution similarity.
- Prunes similar solutions.

Step 5: Neural Network Training

- Collects (initial state, final local search outcome) pairs.
- Trains f_θ to predict promising solutions.
- Updates the model online in each iteration.

3.4 Pros and Cons

Pros:

- Uses deep learning to guide crossovers, greatly improving local search.
- Demonstrates resilience against getting trapped in local optima, with the cost of a slow convergence rate.
- It takes advantage of GPU computations to perform massively parallel local optimization

Cons:

- Computationally expensive due to GPU-based parallel search.
- Requires substantial memory for large populations.
- Neural network training adds overhead in each iteration.
- Performance may degrade on sparse graphs where learning patterns are harder to grasp.

4 New Integer Linear Programming Models for the Vertex Coloring Problem

1. Introduction

The vertex coloring problem is a well-known NP-hard problem in graph theory, which requires assigning the minimum number of colors to the vertices of a graph such that no two adjacent vertices share the same color. Traditional integer linear programming (ILP) formulations, such as the assignment and representatives models, have been widely studied.

2. Methods Proposed in the Paper

The paper introduces three main ILP formulations, which we describe along with their mathematical representations.

2.1 Classical Assignment Model

The assignment model defines binary variables $x_{i,c}$ indicating whether vertex i is assigned color c. The constraints ensure that each vertex receives exactly one color and that adjacent vertices do not share the same color.

$$\min z_{max} \tag{1}$$

s.t.

$$\sum_{c=1}^{H} x_{i,c} = 1 \qquad \forall i \in V \tag{2}$$

$$x_{i,c} + x_{j,c} \leq 1 \qquad \forall (i,j) \in E; 1 \leq c, e \leq H \tag{3}$$

$$z_{max} \geq c x_{i,c} \qquad \forall i \in V; 1 \leq c \leq H \tag{4}$$

$$x_{i,c} \in \{0,1\} \qquad \forall i \in V; 1 \leq c \leq H \tag{5}$$

$$z_{max} \in \mathbb{R} \tag{6}$$

where:

- z_{max} represents the maximum possible color.
- H is an upper bound on the number of colors.
- Constraint (2) ensures that each vertex is assigned exactly one color.
- Constraint (3) ensures adjacent vertices do not share the same color.

2.2 Hybrid Partial-Ordering Based ILP Model

The Hybrid Partial Order Method is a novel Integer Linear Programming (ILP) formulation introduced to solve the vertex coloring problem by combining aspects of the classical assignment model and partial ordering approaches. The goal is to minimize the number of colors required to properly color the graph while maintaining a partial ordering structure. [3]

$$\min 1 + \sum_{i=1}^{H} y_{i,q} \tag{1}$$

s.t.

$$z_{v,1} = 0 \qquad \forall v \in V \tag{2}$$

$$y_{H,v} = 0 \qquad \forall v \in V \tag{3}$$

$$y_{i,v} - y_{i+1,v} \geq 0 \qquad \forall v \in V,\ i = 1, \ldots, H-1 \tag{4}$$

$$y_{i,v} + z_{v,i+1} = 1 \qquad \forall v \in V,\ i = 1, \ldots, H-1 \tag{5}$$

$$x_{v,i} = 1 - (y_{i,v} + z_{v,i}) \qquad \forall v \in V, i = 1, \ldots, H-1 \tag{6}$$

$$x_{u,i} + x_{v,i} \leq 1 \qquad \forall (u,v) \in E,\ i = 1, \ldots, H \tag{7}$$

$$y_{i,q} - y_{i,v} \geq 0 \qquad \forall v \in V,\ i = 1, \ldots, H-1 \tag{8}$$

$$y_{i,v}, z_{v,i} \in \{0,1\} \qquad \forall v \in V,\ i = 1, \ldots, H \tag{9}$$

Explanation of Constraints

- Because q is singled out to hold the maximum color, the summation of yi,q effectively measures how many colors up to H are actively "dominating" higher indices on q.
- Fix boundary condition for z.
- Fix boundary condition for y.
- Solve monotonicity of y.
- Solve linkage relation between y and z (this and previous constraint also imply the monotonicity of y).
- Tie these partial-order variables to the assignment variables (this is why this model is called hybrid). This is done to reduce the number of non-zero coefficients in the next constraint.
- Impose the color difference constraint
- Ensure node q has the highest color

The Hybrid Partial Order Model effectively combines ideas from standard integer programming and partial ordering techniques to achieve better computational performance for the vertex coloring problem.

5 Mehrotra's Column Generation Model

The Mehrotra method is based on a column generation approach, formulating the problem as a set partitioning model. It introduces decision variables that

indicate whether a specific independent set (a feasible color class) is used in the coloring. [5]

$$\min \sum_{S \in \mathcal{S}} y_S \tag{1}$$

s.t.

$$\sum_{S \in \mathcal{S}: v \in S} y_S = 1, \quad \forall v \in V \tag{2}$$

$$y_S \in \{0, 1\}, \quad \forall S \in \mathcal{S} \tag{3}$$

where:

- $\mathcal{S}$ is the collection of all feasible independent sets in the graph.
- y_S is a binary variable indicating whether independent set S is selected in the coloring.
- Constraint (2) ensures that each vertex appears in exactly one selected independent set.
- Constraint (3) enforces that only valid independent sets are selected.

This formulation is solved using a column generation approach, where the number of independent sets is exponentially large, and only a subset is considered at a time. A pricing problem (typically solved as a maximum-weight independent set problem) is used to generate new columns iteratively.

References

[1] Olivier Goudet, Cyril Grelier, and Jin-Kao Hao. A deep learning guided memetic framework for graph coloring problems. *arXiv e-prints*, page arXiv:2109.05948, September 2021.

[2] Olivier Goudet, Cyril Grelier, and Jin-Kao Hao. A deep learning guided memetic framework for graph coloring problems. *Knowledge-Based Systems*, 258:109986, 10 2022.

[3] Anar Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. *European Journal of Operational Research*, 257(1):47–58, 2017.

[4] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luís Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. *arXiv e-prints*, pages 879–885, 11 2019.

[5] Sanjay Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.