Graph Coloring via Learning and Metric-Guided Independent Set Extraction

Anonymous authors
Paper under double-blind review

Abstract

Recent advances in Graph Neural Networks (GNNs) have enabled learning-based approaches for a wide range of combinatorial optimization problems. In this paper, we address one such classical and challenging problem—graph coloring. We introduce an algorithmic framework for graph coloring based on repeated extraction of large independent sets using GNNs. The choice of independent sets for coloring is further guided by metrics which favor use of minimum number of colors. To improve scalability, we further enhance this framework with a value-aware GNN that operates on reduced graphs, learning effective independent set extraction directly on compact representations before lifting solutions to the original graph. We also present a new formulation for quantum circuit depth optimization as a mixed graph coloring problem. The presented approach is evaluated on standard DIMACS benchmark graphs and citation network datasets. Experimental results show that the proposed algorithm achieves significant performance improvements over several traditional and state-of-the-art solvers and scales effectively to larger graphs. The results for benchmark quantum circuit instances are also encouraging.

1 Introduction

The graph coloring problem (GCP) is one of the fundamental problems in graph theory with wide-ranging applications in computer science, operations research, and artificial intelligence. Phrased as an optimization problem, the goal is to assign colors to the vertices of a graph $G = (V, E)$ such that no two adjacent vertices should share the same color while minimizing the total number of colors used. The minimum number of colors required to achieve such a coloring is known as the chromatic number of the graph, denoted by $\chi(G)$. Determining $\chi(G)$ is an NP-hard problem and even deciding whether a given graph is k-colorable is NP-complete for $k \geq 3$.

Graph coloring has been an active field of research as it plays a crucial role in many practical applications including register allocation in compilers, frequency assignment in wireless networks, task and resource scheduling, map coloring and pattern matching in computer vision. Over the decades, a wide range of algorithmic strategies have been proposed for solving GCP ranging from exact methods to heuristics, metaheuristics and learning-based approaches. Exact methods include backtracking algorithm, branch-and-bound algorithms, integer linear programming (ILP) formulation etc. Then we have heuristic methods which provide efficient, often greedy procedures to generate feasible colorings, though without guarantees of optimality. The simplest among them is the greedy coloring algorithm, which processes vertices in a specific order and assigns to each vertex the smallest available color not used by its neighbors. The quality of the resulting coloring heavily depends on the vertex ordering; thus, variants such as Largest Degree ordering (LD), Smallest Last ordering (SL) and Saturation degree ordering (DSATUR) (Brélaz, 1979) have been proposed to improve performance. Another heuristic graph coloring approach is the Recursive Largest First (RLF) algorithm (Leighton, 1979) that iteratively selects large independent sets to form color classes. The algorithm exhibits an average-case time complexity of $\mathcal{O}(n^2)$ and serves as a foundation for more advanced greedy coloring schemes and benchmarks. We also have an algorithm based on tabu search metaheuristic named Tabucol (Hertz & de Werra, 1987) that finds a valid k-coloring of a graph by starting from an initial (possibly invalid) coloring and iteratively improving it through local search.

Recent years have witnessed a surge in leveraging Graph Neural Networks (GNNs) to address NP-hard problems like GCP. Feeding graphs to neural networks is challenging due to their complex topological structures, arbitrary size, non-Euclidean structure, no fixed ordering (isomorphism) etc. GNNs effectively deal with all these challenges and depict excellent application in graph related problems. These models exploit the structure of the graph to learn heuristics that generalize across instances. Some of the existing works include work by Lemos et al. (2019) in which graph coloring problem is formulated as a binary classification task that is, given a graph $\mathcal{G}$ and a number k, is $\mathcal{G}$ k-colorable. The algorithm proposed learns vertex embeddings which are then fed into an MLP, which computes a logit probability corresponding to the model's prediction of the answer to the decision problem. Another popular approach is by Schuetz et al. (2022b) using physics-inspired GNNs. They design a specific loss function based on ising model in physics to obtain colorings with minimum number of conflicts. We obtain a conflict when there exist a pair of vertices which are adjacent and have received same color. Due to message-passing mechanism of GNNs, the embeddings obtained for adjacent nodes tends to be similar. However, for problems like graph coloring the embeddings learned by GNN for adjacent nodes should be dissimilar and significantly apart in embedding space then only they will be labeled with different colors. Thus, several works have focused on designing GNNs specifically for graph coloring problem. Wang et al. (2024) introduces GNN-NU, a GNN based on negative message passing. Li et al. (2022b) designs Graph Discrimination Network (GDN) which gives significant improvements in results compared to those obtained using standard GNNs like GCN, Sage and GIN in unsupervised setting. GNNs have also been used to design ordering heuristics for coloring (Langedal & Manne, 2025) and guide traditional algorithms for coloring (Zheng et al., 2025).

Key Contributions.

- We present an algorithmic framework named GBS for graph coloring based on repeated extraction of efficient independent sets. Here, the choice of independent sets are governed by metrics which ensure that coloring obtained uses minimum number of colors.
- For solving GCP for large graphs, we introduce value-aware GNN that incorporates reduction techniques and efficiently extracts independent sets for original graphs by working on reduced graphs.
- We perform thorough experimental validation on DIMACS benchmark graphs and citation network datasets and compare results with both traditional algorithms and ML solvers. The results demonstrate superior performance of our method compared to existing state-of-the-art approaches. Compared with heuristic algorithms, our approach requires additional computational time but consistently yields higher-quality solutions, highlighting a trade-off between computational cost and solution quality.
- We model quantum circuit depth optimization, which is a practically significant problem in quantum computing as a mixed graph coloring problem. We evaluate the proposed algorithm on this task where it exhibits promising performance.

2 Preliminaries

2.1 Graph coloring problem (GCP)

Given a simple, undirected graph $G = (V, E)$ with vertex set $V = \{1, 2, 3 \ldots, n\}$ and edge set $E = \{(i, j) : i, j \in V\}$, a k**-coloring** is a function $f : V \to \{1, 2, ..., k\}$ such that

$$\forall\ (i, j) \in E,\ f(i) \neq f(j)$$

Graph coloring problem can be formulated in two ways. In the decision version, given a graph $G = (V, E)$ and an integer k, we ask whether G has a k-coloring or not. Moreover, in the optimization version we find the minimum k such that G has a k-coloring. This k is the chromatic number of G denoted by $\chi(G)$. A graph G with $\chi(G) = k$ is called k-chromatic and if $\chi(G) \leq k$ then G is said to be k-colorable. A k-coloring partitions the vertex set V into k independent sets called color classes.

2.2 Graph neural networks (GNNs)

Graphs are a natural representation for data where entities interact through complex relationships like social networks, molecular structures, citation graphs, communication networks etc. Traditional machine learning models, which typically assume vectorized inputs often struggle to process such irregular, non-Euclidean structures. This motivated the development of Graph Neural Networks (GNNs), a class of neural architectures designed specifically to learn from graph-structured data. GNNs offer a scheme to generate node representations using message-passing mechanism over the nodes and edges of a graph. The node embeddings learned incorporate the topology of the graph and are permutation invariant (i.e. node ordering does not affect output). Given a graph $G = (V, E)$ on n vertices with node features $\mathbf{X} \in \mathbb{R}^{d \times n}$, a GNN updates each node's representation h_v by aggregating information from its neighbors $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$. In its general form, the node embedding at layer l is computed as

$$h_v^{(l)} = \text{UPDATE}^{(l)} \left(h_v^{(l-1)}, AGG^{(l)} \right)$$
$$AGG^{(l)} = \text{AGGREGATE}^{(l)} \left(\{ h_u^{(l-1)} : u \in \mathcal{N}(v) \} \right)$$

where $AGGREGATE(.)$ combines messages from neighbors (e.g., sum, mean, max) and $UPDATE(.)$ combines the aggregated message with the current node state. For $l = 0$, the initial representations $h_v^0 \in \mathbb{R}^d$ are usually derived from the node's labels or given input features of dimensionality d. After L layers of message passing, the final node embedding $h^{(L)}$ captures the local neighborhood structure upto L hops. GNNs have a lot of variants such as graph convolutional networks (GCNs), graph attention networks (GATs), GraphSAGE etc.(Kipf, 2016; Velickovic et al., 2017; Hamilton et al., 2017) which differ mainly in how they aggregate and combine neighbor information. For instance, GCNs perform a form of spectral convolution on graphs while GATs introduce attention coefficients to assign different importance to neighboring nodes. These learned node embeddings can then be used for prediction tasks such as node classification. To optimize predictive power these parametrized node embeddings $h^{(L)}(\theta)$ are fed into a problem-specific loss function to optimize the weight parameters of the network.

2.3 GCN based MIS Extraction

We make use of the approach discussed by (Li et al., 2018) where they find maximum independent set (MIS) for a graph $G = (V, E)$ by estimating the likelihood for each vertex $v \in V$ of being present in MIS. Given a graph $G = (V, E)$, the goal was to produce a binary labeling for each vertex in G such that label 1 indicates that a vertex is in the independent set and label 0 indicates that it's not. They learn a network f that takes the graph G on N vertices as input and outputs $f(G)$ which is a probability map on vertices $p : V \to [0, 1]^N$ that indicates how likely each vertex is to belong to the MIS. Simply rounding the probability map $f(G)$ to a discrete assignment may violate the independence constraint. Thus to obtain a feasible solution, the authors follow a greedy labeling approach in which the learned probability map $f(G)$ is used as a heuristic function.

For a multi-solution problem such as finding MIS, one probability map need not be sufficient. For example, a 4-cycle have two equivalent optimal solutions that induce completely different labels. But the network due to lack of any supervision may assign equal probabilities of 0.5 to each vertex. To solve this issue, authors suggest to train the network f to generate multiple probability maps $\langle f^1, f^2, \ldots, f^M \rangle$. Later, these multiple probability maps are used within a tree search procedure that generates a large number of equivalent solutions simultaneously. More details on this approach are given in appendix A.1.

3 Guided Beam Search

We know that a k-coloring for a graph $G = (V, E)$ partitions its vertex set V into independent sets (IS) called color classes. To reduce the number of colors used, it is intuitive to look for independent sets that are maximal and have large sizes so that more number of vertices can be assigned a single color class. However, these choice of independent sets may not be suitable always and we may require an additional metric to choose the most suitable IS. Thus we introduce Guided Beam Search algorithm (GBS) which iteratively

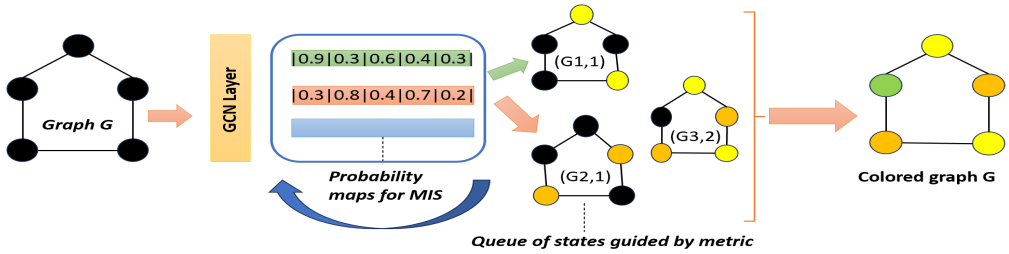

Figure 1: The figure explains the implementation of Guided Beam Search for Graph Coloring.

Algorithm 1 Guided Beam Search for Graph Coloring

Input: Graph $G = (V, E)$, queue capacity c
Output: Coloring with minimum number of colors
Initialize queue Q with capacity c
Initialize $min_colors \leftarrow \infty$
Enqueue $(G, 0)$ into Q
while Q is not empty **do**
  $(G_{current}, num_colors) \leftarrow$ Dequeue from Q
  if $G_{current}$ has no vertices **then**
    $min_colors \leftarrow \min(min_colors, num_colors)$
    continue
  end if
  $MIS_list \leftarrow$ find_multiple_MIS$(G_{current})$
  for each MIS in MIS_list **do**
    $G_{new} \leftarrow G_{current}$ with MIS removed
    Enqueue $(G_{new}, num_colors + 1)$ into Q
  end for
  update_queue(Q)
end while
return min_colors

extracts MIS from the graph and chooses the most suitable ones as color classes. The algorithm maintains a queue of fixed size consisting of graph states where each state corresponds to a partially colored graph along with the number of colors used so far. At each step, multiple MIS candidates are generated for the current graph. For every candidate MIS, a new graph state is created by removing the MIS vertices. These new states are evaluated using a predefined metric function that estimates the potential of each state toward an efficient coloring. Only the most promising states as determined by this metric and constrained by the beam width are retained for further exploration. This process continues until all vertices are assigned a color. Algorithm 1 outlines the guided beam search approach.

The algorithm predominantly uses two functions, $find_multiple_MIS(.)$ and $update_queue(.)$. The $find_multiple_MIS(.)$ function outputs multiple MIS sets for the graph using MIS extraction approach discussed above. On the other hand the $update_queue(.)$ function sorts queue(Q) based on a pre-defined metric and keep at most c graph states (c is queue size) with minimum value for the metric. Note that although the network f is trained to output maximum independent set for a graph G, the independent set I obtained may have size $|I| < \alpha(G)$ (where $\alpha(G)$ is independence number). However, I is maximal and this gap $(\alpha(G) - |I|)$ is small. In a sense, this observation is useful as extraction of independent sets of maximum size in beginning need not lead to an optimal coloring (coloring that uses minimum number of colors) always. The efficiency of the algorithm significantly depends upon the MIS extracted and choice of metric used to evaluate and rank intermediate graph states. These also influence the exploration of solution space for finding efficient colorings. Next, we discuss the different metrics that we designed for use in the

algorithm. For a given graph $G = (V, E)$, a graph state (G', c) denotes that $G' \subseteq G$ obtained after removing the vertex set $V(G) \setminus V(G')$ which has already been colored using c colors.

- **Edge density:** Each edge (u, v) corresponds to a constraint that vertices u and v cannot share the same color. Thus, graph states with minimal edge density would impose fewer coloring constraints. For a graph $G = (V, E)$, the metric is defined as

$$M_{ed}(G) = \frac{\text{no. of edges in } G}{\text{Total no. of possible edges in } G} = \frac{|E|}{\binom{|V|}{2}}$$

- **Edge density with colors used:** While edge density is a useful measure, relying on it alone can lead to suboptimal decisions as it ignores the number of colors used so far. For example, consider two graph states (G_1, c_1) and (G_2, c_2) such that $M_{ed}(G_1) < M_{ed}(G_2)$ and $c_1 > c_2$. Then G_1 appears more favorable under the edge density metric, but it could have reached its current form only after multiple MIS extractions, using many colors. In contrast, G_2 might still be in early stages of coloring and may offer better long-term potential. To account for this trade-off, we propose the combined metric M_{edc} for graph state (G, c) given by

$$M_{edc}((G, c)) = M_{ed}(G) \times \frac{c}{\Delta(G) + 1}$$

  where $\Delta(G)$ denotes maximum degree of G. This metric favors graph states which are sparse while simultaneously using minimum number of colors.

- **Average closeness centrality:** Closeness centrality is a graph-theoretic measure that quantifies how close a given node is to all other nodes in the graph, based on the lengths of the shortest paths. For a connected graph $G = (V, E)$, the closeness centrality of a node $v \in V$ is defined as

$$C(v) = \frac{n - 1}{\sum_{u \in V, u \neq v} d(v, u)}$$

  where $n = |V|$ is the number of vertices and $d(v, u)$ is the shortest path distance between nodes v and u. The average closeness centrality of the graph is then defined as the mean value of $C(v)$ over all vertices. Thus, for a graph state (G, c) where $G = (V, E)$,the metric is defined as

$$M_{acc} = \frac{1}{n} \sum_{v \in V} C(v) = \frac{1}{n} \sum_{v \in V} \frac{n - 1}{\sum_{u \in V, u \neq v} d(v, u)}$$

  In the context of graph coloring, this metric can serve as a quantifier for the complexity of the remaining graph structure. A subgraph with high average closeness centrality tends to have more interconnected and centrally positioned nodes, which may lead to more coloring conflicts due to tighter clustering. On the other hand, a graph with lower average closeness centrality may be more tree-like or loosely connected, potentially allowing easier decomposition into independent sets.

- **Color efficiency:** To minimize number of colors used one can also look for maximizing the number of vertices per colors class. This can be done using color efficiency metric which for a graph state (G', c) is defined as

$$M_{ce} = \frac{|V(G) \setminus V(G')|}{c}$$

  A higher color efficiency indicates that more vertices are being colored with fewer colors, which is a desirable property in graph coloring.

- **Progress rate:** This metric keeps a track of number of vertices that have already been colored. For coloring graph G with intermediate state (G', c), it is defined as

$$M_{pr} = \frac{|V(G) \setminus V(G')|}{|V(G)|}$$

4 Weighted Beam Search

In this section, we aim to enhance guided beam search by employing a weighted combination of multiple metrics with the weights predicted by a neural network trained to adapt to the structural properties of each input graph. The single metric score is replaced with a weighted metric M_w which is weighted sum of four metrics namely edge density (M_{ed}), average degree (M_{avg}), color efficiency (M_{ce}) and progress rate (M_{pr}).

$$M_w = w_1.M_{ed} + w_2.M_{acc} + w_3.M_{ce} + w_4.M_{pr}$$

Here, metric average degree (M_{avg}) computes average degree of graph G' in intermediate state ($G'c$) and normalizes by max degree of initial graph G. To determine effective weights for this weighted metric for each graph instance, we developed a two-stage learning pipeline which included training data generation and neural network training. For generating training data we chose a representative subset of training graphs and performed random search over possible weight combinations to identify those that yield the best coloring results (i.e., minimum number of colors used). Each graph in training set was labeled with the optimal combination obtained for it. Further, we trained a lightweight feed-forward neural network referred to as the $Weight_Predictor(.)$ that takes as input a set of structural features of a graph (including number of nodes, number of edges, density, average/maximum degree, clustering coefficient, and diameter). and outputs optimal weight vector $\mathbf{w} \in [0,1]^4$ for graph G that can be used while using weighted metric M_w in GBS. This adaptive weighting mechanism allows the GBS algorithm to dynamically modify its metric prioritization to the underlying structure of each graph, resulting in improved performance over using a fixed metric across all instances.

5 Value-aware GNN for graph reductions

Real-world graphs such as citation networks, social networks and communication systems etc. often consist of millions of nodes and edges. Applying graph coloring algorithms directly to such large-scale graphs can be computationally challenging. To address this, we explore graph reduction techniques and how they can be incorporated in the framework discussed above.

Definition 5.1. A *graph reduction* is a mapping $R : \mathcal{G} \rightarrow \mathcal{G}$ over the set of graphs $\mathcal{G}$ that maps a graph $G = (V, E)$ to a smaller graph $G' = (V', E')$ (known as the reduced graph) such that $|V'| < |V|$ and/or $|E'| < |E|$ while preserving certain structural or problem-specific properties of G.

For coloring we explore two reduction techniques namely vertex folding and pendant folding which are also mentioned in paper by Li et al. (2018).

- **Vertex folding:** This technique focus on vertices of degree 2 having non-adjacent neighbors. If $deg(v) = 2$ and $N_v(G) = \{u, w\}$ such that $uw \notin E(G)$ then u, v, w are deleted and representative vertex v' is added. All neighbors of u and w are joined to v'.
- **Pendant folding:** This technique targets the set of leaf nodes $L(G)$ (degree 1 vertices). If $v \in L(G)$ such that $N_v(G) = u$ then u and v are deleted and representative vertex v' is added. All the neighbors of u are joined to v'.

Such reductions can be used as pre-processing steps for a graph G and reduced graph G' can be given as input to obtain graph coloring for G. Note that in cases where we obtain $min_colors = 1$ using GBS for the reduced graph, we need to increase color count by 1 as the original graph will require 2 colors when the vertices will be unfolded.

For a given reduction function, size of maximum independent set need not be preserved. For example, for graph G in figure 2 the reduced graph G' has an MIS of size 1. If either of sets $\{D\}, \{F\}$ is chosen then MIS size will be 1 for original graph and if node $ABCE$ is chosen then unfolding vertices will generate $\{A, E\}$ or $\{B, C\}$ as MIS which are of size 2. However, original graph has an MIS of size 3. Also in some cases, the reduced graph obtained have chromatic number $\chi(G') > \chi(G)$ (see figure 2) This happens when the reduced graph becomes structurally complex and dense due to reduction.

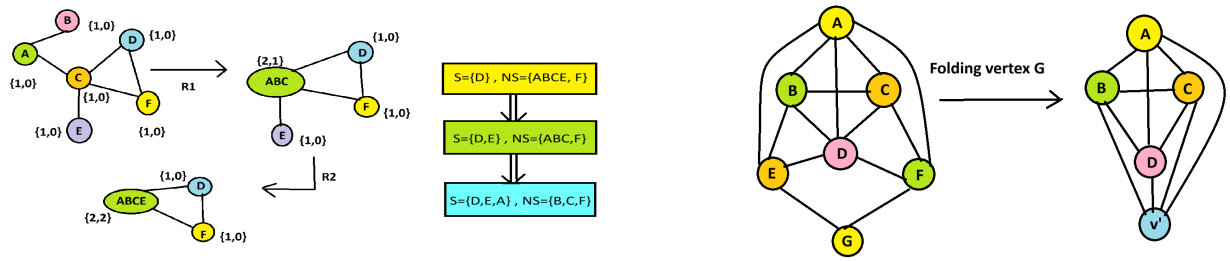

Figure 2: In first figure, graph G' is obtained after folding vertex A followed by folding vertex E. S and NS denotes sets of selected and non-selected nodes respectively. In second figure, Original graph uses 4 colors whereas the reduced graph uses 5 colors in optimal coloring.

Thus, it becomes necessary to incorporate reduction information in the reduced graph. This can be done by labeling each node with select and nonselect values which will measure its contribution to original MIS if it is being chosen or not. This value-tracking mechanism will help in preserving cardinality information throughout the reduction process. Select and nonselect values for vertex folding a vertex v with non-adjacent neighbors u and w are governed by equations (1) and (2) and for pendant folding vertex v (degree 1) with its neighbor u by equations (3) and (4). These values are initialized to 1 and 0 respectively for all original (unmerged) vertices and then updated systematically based on graph reduction technique used (see figure 2).

$$\text{select}(v') = \text{select}(u) + \text{select}(w) + \text{nonselect}(v) \tag{1}$$
$$\text{nonselect}(v') = \text{select}(v) + \text{nonselect}(u) + \text{nonselect}(w) \tag{2}$$
$$\text{select}(v') = \text{select}(u) + \text{nonselect}(v) \tag{3}$$
$$\text{nonselect}(v') = \text{nonselect}(u) + \text{select}(v) \tag{4}$$

Once a maximum independent set S is identified in the reduced graph, the exact cardinality of the corresponding maximal independent set in the original graph can be computed using the value-aware representation:

$$\text{cardinality} = \sum_{v \in S} \text{select}(v) + \sum_{v \notin S} \text{nonselect}(v)$$

This formulation provides an efficient mechanism for estimating the quality of independent sets without reconstructing the full original graph. To incorporate this value-tracking mechanism, we learn $value_aware_multiple_MIS(.)$ which similar to $find_multiple_MIS(.)$ finds multiple value-aware MIS for G by taking reduced graph G' as input. Value-aware MIS for reduced graph G' is an independent set I' of G' which will produce MIS for original graph after unfolding vertices. Unfolding vertices here means that for each representative vertex v' we add the corresponding vertices from G based on whether v' belongs to I' or not. Algorithm 2 explains the working of value-aware GBS approach. Compared to GBS, this algorithm works directly on the reduced value-aware graph in which each node v has a tuple $(select(v), nonselect(v))$ associated with it. While generating a new graph G_{new} from the current graph select and non-select values are updated according to the chosen MIS. If $v \in MIS$, then $select(v)$ is updated to 0 making sure that v will not contribute any select value further as it is already being selected by the MIS. Similarly, if $v \notin MIS$ then $nonselect(v)$ is updated to 0. Also, there may arise a case when $select(v) = 0$ but $nonselect(v) > 0$, such a node is made isolated so that next time it is selected in MIS and the vertices corresponding to nonselect value are being colored. Finally, if $select(v) = 0$ and $nonselect(v) = 0$ then such a node is removed from $G_{current}$.

For effectively learning this network we modify node embeddings and loss function. Node embeddings are initialized with the value pair $[\text{select}(v), \text{nonselect}(v)]$ for each vertex v, instead of uniform or identity features as in standard GCN settings. The GCN is trained to predict membership in independent sets that maximize

Algorithm 2 Value aware coloring with Reduction R

Input: Graph $G = (V, E)$, queue capacity c, Reduction function R
Output: Coloring with minimum number of colors
Initialize queue Q with capacity c
Initialize $min_colors \leftarrow \infty$
Generate value-aware graph, $G_{reduced} \leftarrow R(G_{current})$
Enqueue $(G_{reduced}, 0)$ into Q
while Q is not empty **do**
  $(G_{current}, num_colors) \leftarrow$ Dequeue from Q
  if $G_{current}$ has no vertices **then**
    $min_colors \leftarrow \min(min_colors, num_colors)$
    continue
  end if
  $MIS_list_red \leftarrow$ value_aware_multiple_MIS$(G_{reduced})$
  for each MIS in MIS_list_red **do**
    $G_{new} \leftarrow update_graph(G_{current}, MIS)$
    Enqueue $(G_{new}, num_colors + 1)$ into Q
  end for
  update_queue(Q)
end while
return min_colors

the *total select value* rather than merely the number of nodes. The guided tree search component is adapted to prioritize candidate nodes based not only on GCN output scores but also their select and nonselect values.

To effectively train the GCN under this modified objective, we propose a Value-Aware Binary Cross-Entropy (VA-BCE) loss function, which weights the loss contribution of each node according to its selectivity:

$$L_{\text{VA-BCE}} = -\frac{1}{n}\sum_{i=1}^{n} w_i \cdot \left[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)\right], \tag{5}$$

where $y_i \in \{0, 1\}$ is the ground-truth label for node i, p_i is the predicted probability of node i being in the MIS, and w_i is a value-aware weight defined as:

$$w_i = \begin{cases} \frac{s_i}{\bar{s}} & \text{if } y_i = 1, \\ \frac{ns_i}{\overline{ns}} & \text{if } y_i = 0, \end{cases} \tag{6}$$

where $s_i = \text{select}(v_i)$, $ns_i = \text{nonselect}(v_i)$, and $\bar{s}$, $\overline{ns}$ are the mean values over all nodes with the respective labels.

This value-aware loss encourages the model to prioritize including nodes with high select values and excluding nodes with high nonselect values, leading to more informed and effective MIS predictions. By explicitly encoding node-level contribution information, this approach enhances the GCN's ability to learn the structural and value-based dynamics that arise during graph reduction. We provide empirical evidence of its effectiveness on several representative graph instances in results section.

6 Quantum Circuit Depth Optimization

Quantum algorithms are typically implemented using the quantum circuit model in which computations are represented as sequences of quantum operations (known as quantum gates) acting on qubits. These circuit models are highly susceptible to errors and inefficiencies due to quantum noise and the limited capabilities of existing quantum hardware. Also, quantum computers are highly susceptible to decoherence which is qubit loss of quantum information over a period of time. Thus, optimizing quantum circuits becomes critical

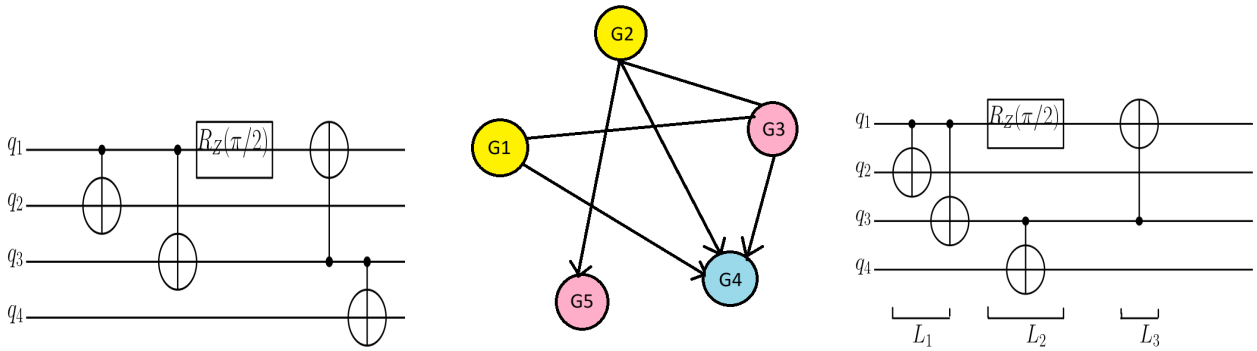

Figure 3: Sample circuit C ($depth = 5$), its corresponding dependency graph G and the optimized circuit C' ($depth = 3$).

for enhancing computational speed and mitigating errors caused by quantum noise. Over the past years, researchers have devised innovative techniques and algorithms to optimize quantum circuits focusing on characteristics of circuits such as circuit depth, gate count, qubit count, gate fidelity etc. (Karuppasamy et al., 2025; Saeedi & Markov, 2013; Nam et al., 2018; Huang & Sun, 2022)

In this paper, we focus on circuit-depth optimization which can be formulated as GCP. Circuit depth is the number of quantum gates executed sequentially in a quantum circuit. Circuits with smaller depth are generally preferable as shorter execution time reduces exposure to decoherence, limits accumulation of noise and thereby improves fidelity. Recent work by Lee et al. (2025) modeled this problem as a graph coloring task where quantum gates are vertices and edges capture commutation-based conflicts. However, their model makes a strong assumption that all gates commute and thus only undirected edges are required to model restrictions. We generalize the prior formulation for arbitrary circuits by modeling the quantum circuit as a mixed graph $G(V, E_d, E_u)$, which includes both, directed edges (E_d) to enforce execution order due to non-commuting gate dependencies and undirected edges (E_u) to prevent simultaneous execution of commuting gates that act on overlapping qubits.

To perform graph coloring, we transform a quantum circuit into a *dependency graph* $G(V, E_d, E_u)$ where each node represents a quantum gate and edges are added based on dependencies. A directed edge from gate g_1 to g_2 represents a hard dependency that g_1 must be executed before g_2. This occurs when both gates act on overlapping qubits and do not commute. An undirected edge between g_1 and g_2 implies that the two gates commute but share at least one qubit. Given a dependency graph $G(V, E_d, E_u)$, we require a suitable coloring that can be converted to an execution order that satisfies the constraints imposed in a given circuit.

Definition 6.1. A *suitable coloring* $f : V \to \mathbb{N}$ is one that assigns color labels such that for each directed edge $(u, v) \in E_d$, $f(u) < f(v)$ which enforces the correct execution order and for each undirected edge $(u, v) \in E_u$, $f(u) \neq f(v)$ which ensures that conflicting (commuting) gates do not overlap.

Corresponding to the obtained coloring, gates must be executed in the sequence of increasing color label with gates labeled with same color being executed in parallel. Then the circuit depth is given by $max_{v \in V} f(v)$ which can be minimized by minimizing the number of colors used. The coloring algorithms defined in previous sections can be used for coloring this dependency graph. Since, the dependency graph obtained is directed we require additional preprocessing and postprocessing steps. Given a dependency graph $G(V, E_d, E_u)$, we extract a subgraph $G_{exec}(V', E'_u)$ which only consist of vertices with indegree 0 and corresponding undirected edges between these vertices. There always exist a vertex with indegree 0 i.e. the vertex corresponding to 1st gate. Then the GBS algorithm finds a suitable independent set in G_{exec}. Once an independent set is identified, graph G is updated to G' by removing the colored nodes and updating the indegrees of all remaining nodes. The process is repeated again for updated graph G' until all nodes in G are colored. The steps added above ensures that the coloring obtained will be suitable. (Refer to Algorithm 4 in appendix).

Theorem 6.2. *The optimized circuit C' obtained from above algorithm is executable i.e. the coloring obtained is suitable.*

Proof: The proof is deferred in Appendix A.5

7 Numerical Experiments

7.1 Experimental setup

We solve GCP on DIMACS coloring dataset (Johnson & Trick) as well as on citation network datasets which are often used for graph-based benchmark experiments. The presented approach is compared to several traditional heuristics based algorithms and ML-solvers developed for graph coloring. The heuristics based algorithms used for comparison are LF, SL, DSatur (Brélaz, 1979),RLF (Leighton, 1979),Tabucol (Hertz & de Werra, 1987). Further, the ML based solvers used for comparison are PI-GCN and PI-SAGE (Schuetz et al., 2022b),GDN (Li et al., 2022b), GIN (Zhang et al., 2024) and GNN-NU (Wang et al., 2024). Given a graph G and a positive integer k, all of these solvers try to obtain coloring using k colors minimizing conflicts of colors between adjacent vertices. For testing, the best known value of k for a graph G is given and the number of conflicts in the obtained solution are noted. For detailed information on experimental setup refer appendix A.2.

Table 1: Coloring results and comparison with traditional heuristic algorithms. Number of colors used by optimal coloring obtained from each solver is reported. Metric used for GBS is edge density with colors used.

Graph Name	$\|V\|$	$\|E\|$	χ	**GBS**	**W-GBS**	**RLF**	**DSatur**	**LF**	**SL**
queen13_13	169	6656	13	15	**15**	16	17	22	20
queen11_11	121	3960	11	13	**13**	14	15	19	18
queen8_8	64	728	9	10	**9**	11	12	13	14
le450_5a	450	5714	5	8	**7**	8	10	11	11
DSJC500.9	500	224874	?	142	**140**	152	170	169	173
DSJC1000.5	1000	249826	?	95	**94**	107	115	121	125
DSJC250.9	250	27897	?	79	**76**	83	92	93	96
school1_nsh	352	14612	?	21	**20**	22	27	32	23
queen15_15	225	10360	?	17	**16**	19	21	25	23
DSJC125.9	125	13922	?	49	**46**	49	51	53	52
DSJC1000.9	1000	898898	?	252	**246**	282	299	313	319
latin_square_10	900	307350	?	113	**110**	126	132	204	204

7.2 Results

DIMACS coloring dataset consist of 71 graphs out of which chromatic number is known for 51 graphs. For these graphs, the GBS algorithm obtained correct chromatic number for 57% of graphs while for other graphs extra colors used were at most 4. For graphs whose chromatic number were unknown, our algorithm obtained better results than existent algorithms. Using weighted-GBS further improved the results. Results obtained for some of the graph instances are reported in table 1 and 2 along with comparison with baselines. We report the number of colors used along with conflicts obtained in final coloring for tabucol and GNN based solvers as they use $\chi(G)$ colors and returns coloring with minimum number of conflicts. We also analyzed the different metrics discussed in Section 3 out of which edge density with color metric performed significantly better then others especially for large graphs. Detailed results for different metrics are given in appendix A.3.

For testing value-aware GBS we considered few instances from DIMACS dataset which showed significant reduction in size after applying vertex folding and pendant folding reductions. Also citation graphs and social network graphs like Cora,Citeseer,Pubmed and Ego-Facebook are considered to test model efficiency for large-scale graphs. Value-aware GBS method helped in significantly reducing runtime while maintaining efficiency. Compared to applying GBS on reduced graph, value-tracking technique served as an efficient

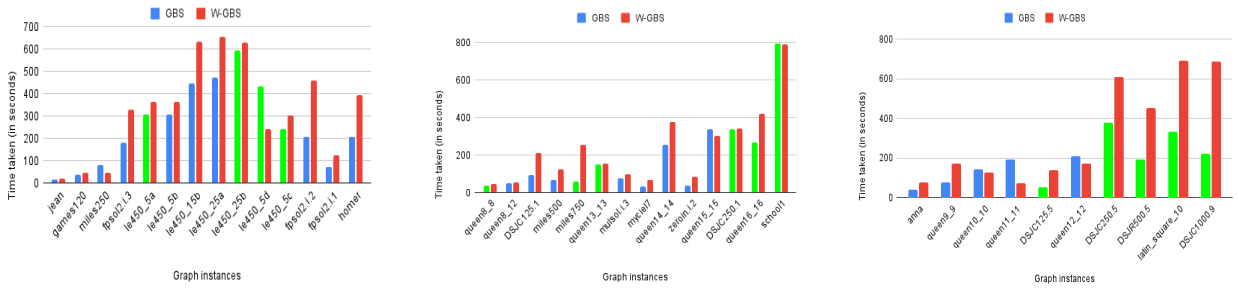

Figure 4: For better analysis the graph dataset is divided into three groups based on edge density, $G1$ with edge density $p < 1$ (left figure), $G2$ with edge density $1 \leq p < 5$ (middle figure), $G3$ with edge density $p \geq 5$ (right figure). In each figure, graphs are arranged in increasing order of number of vertices. Graphs for which W-GBS performed better than GBS are highlighted with green color.

Table 2: Coloring results and comparison with Tabucol and GNN-based models. For GBS we report colors used by coloring obtained whereas for rest of them, number of conflicts in the obtained $\chi(G)$-coloring are reported.

Graph Name	χ	**GBS**	**Tabu col**	**PI GCN**	**GDN**	**GNN NU**	**PI SAGE**	**GIN**
myciel5	6	6	0	0	0	0	0	0
myciel6	7	7	0	0	0	0	0	0
queen5_5	5	5	0	0	0	0	0	0
queen6_6	7	9	0	1	0	0	0	0
queen7_7	7	7	10	8	9	0	0	0
queen8_8	9	10	9	6	2	1	1	0
queen9_9	10	11	5	13	6	1	1	1
queen8_12	12	13	10	10	0	0	0	2
queen11_11	11	13	33	37	21	13	17	7
queen13_13	13	15	42	61	33	15	26	17

guiding tool in selection of useful intermediate states. Also, note that large reduction (above 50%) may increase color gap ($OPT(G) - ALG(G)$) as the graph may become much more complex like it happens in case of Pubmed. (refer table 3) For these large graphs, the presence of multiple MIS solutions could confuse the generation of MIS in starting states. While constructing MIS from probability maps (refer algorithm 3), the algorithm assigns labels 1/0 (for chosen/not-chosen) based on probability maps and breaks the loop once it hits an already labeled vertex. This semi-labeled graph is added to queue and process of labeling is further repeated for remaining graph. But for such large graphs the algorithm never hit a completely labeled graph state. Thus, we relaxed the labeling criterion by replacing break with continue and now picking only 3/4th of vertices with good probability scores in descending order. We refer this method as C-GBS. Hence, for large graphs in table 3 the results are obtained using C-GBS. We also test C-GBS for DIMACS dataset and for small to medium graphs the results obtained by GBS and C-GBS are almost same differing by at most 2-3 colors. However, as graph size becomes large (above ≈ 500 nodes) difference in colors used by the two methods increases.

Runtime analysis: Consider graph $G(V, E)$ with $|V| = n$ and $|E| = m$ and queue capacity c given as input to GBS algorithm. Let k be the maximum number of MIS extracted for a graph state. For each processed state the algorithm computes maximum k MIS (in time $T_{MIS}(n, m)$), build k candidate graphs (in time $k.O(n + m)$) and update queue (in time $O(k.log(c))$). The number of processed states can be at most $c.n$. The $find_multiple_MIS(.)$ function requires one GNN forward pass ($O(n+m)$) to generate all probability maps and further exploration of partial solutions which is controlled by giving time budget. In practice, the running time scales approximately linearly with graph size for the majority of tested graphs. Runtime comparison cannot be drawn with GNN-based solvers as all existent solvers use optimal/near-optimal colors

Table 3: Results for value aware guided beam search.

Graph Name	χ	**Nodes**	**Edges**	**Nodes (Red.)**	**Edges (Red.)**	**GBS**	**Time (sec)**	**Value-GBS**	**Time (sec)**
homer	13	556	1628	315	1331	14	250.34	14	100.54
huck	11	74	301	68	295	11	20.37	11	23.48
david	11	87	406	77	396	11	23.64	11	24.11
anna	11	138	493	112	467	12	42.06	12	45.12
miles250	8	125	387	118	380	9	45.65	8	35.85
jean	10	77	254	55	229	10	35.20	10	13.47
Cora	5	2708	5278	1632	3966	7	7.34	7	3.37
Citeseer	6	3327	4552	1436	2384	8	17.65	8	3.32
Pubmed(75%)	8	19717	44324	5646	23815	10	542.46	15	20.82
Pubmed(60%)	8	19717	44324	7885	30139	10	542.46	16	21.91
Pubmed(50%)	8	19717	44324	9857	34084	10	542.46	16	20.83
Pubmed(40%)	8	19717	44324	11830	36437	10	542.46	16	21.82
Pubmed(25%)	8	19717	44324	14787	39394	10	542.46	14	15.99
Ego Facebook	?	4039	88234	3962	88157	92	175.38	88	73.89

Table 4: Comparison of circuit depth between qiskit DAG method and GBS

Circuit	**Nodes**	**Edges**	**Qiskit DAG**	**GBS**
grover_n2.	18	93	12	12
grover_n2_trans	17	86	12	10
multiplier_n15	73	947	49	49
multiplier_n15_trans	563	29767	257	213
adder_n64	268	1556	78	71
adder_n64_trans	1220	24802	462	391

We test our algorithm on benchmark quantum circuits (Li et al., 2022a) against Qiskit DAG method. This method uses qiskit library to convert a given quantum circuit to a directed acyclic graph (DAG) and then find circuit depth as length of longest directed path. For the tested circuit instances, GBS significantly reduces circuit depth especially for circuits with higher gate count (i.e. higher number of nodes).

for coloring and find number of conflicts. Obtaining feasible coloring from such a coloring obtained is not specified which can take additional time. Moreover, in worst case we may have to introduce as many colors as number of conflicts which will significantly increase the number of colors used. However, we have drawn comparison between runtime of GBS and W-GBS for groups of graphs with different edge densities (refer figure 4). We have also reported runtime for Value-GBS in table 3.

8 Conclusion

In this work, we address the classical graph coloring problem (GCP) through a learning-based approach that integrates GNNs with guided beam search (GBS). A key idea throughout this work is the repeated extraction of MIS to build up color classes. The structured exploration capability of beam search guided by well-curated metrics enables the algorithm to generate high-quality color assignments for complex graph instances. We also introduce value-aware GNN that helps to integrate reduction techniques with GBS algorithm and enables scalable inference on larger graphs. However, there is room for further exploration for the proposed method. Alternative metrics can be explored that could offer better heuristics or problem-specific insights. In addition, domain-specific metrics could be incorporated when applying graph coloring to practical problems such as scheduling, register allocation, or quantum circuit mapping—where additional constraints or priorities exist beyond minimizing the number of colors. For scaling purposes, one could design reduction strategies that can operate effectively on dense graphs, especially those with high clustering or global connectivity.

References

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Chase Cummins and Richard Veras. Reinforcement learning for graph coloring: Understanding the power and limits of non-label invariant representations. *arXiv preprint arXiv:2401.12470*, 2024.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*, 2025. URL `https://docs.gurobi.com/projects/optimizer/en/current/index.html`. Version 12.0.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4): 345–351, 1987.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Zhenyu Huang and Siwei Sun. Synthesizing quantum circuits of aes with lower t-depth and less qubits. In *International conference on the theory and application of cryptology and information security*, pp. 614–644. Springer, 2022.

David S. Johnson and Michael Trick. Graph coloring instances. `https://mat.tepper.cmu.edu/COLOR/instances.html`.

Krishnageetha Karuppasamy, Varun Puram, Stevens Johnson, and Johnson P Thomas. A comprehensive review of quantum circuit optimization: Current trends and future directions. *Quantum Reports*, 7(1):2, 2025.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kenneth Langedal and Fredrik Manne. Graph neural networks as ordering heuristics for parallel graph coloring. In *2025 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pp. 56–67. SIAM, 2025.

Hochang Lee, Kyung Chul Jeong, and Panjin Kim. Quantum circuit optimization by graph coloring. *arXiv preprint arXiv:2501.14447*, 2025.

Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489, 1979.

Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 879–885. IEEE, 2019.

Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 2022a.

Wei Li, Ruxuan Li, Yuzhe Ma, Siu On Chan, David Pan, and Bei Yu. Rethinking graph neural networks for the graph coloring problem. *arXiv preprint arXiv:2208.06975*, 2022b.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):23, 2018.

Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

Yun Peng, Byron Choi, and Jianliang Xu. Graph learning for combinatorial optimization: a survey of state-of-the-art. *Data Science and Engineering*, 6(2):119–141, 2021.

Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)*, 45(2):1–34, 2013.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022a.

Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. Graph coloring with physics-inspired graph neural networks. *Physical Review Research*, 4(4):043131, 2022b.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

Xiangyu Wang, Xueming Yan, and Yaochu Jin. A graph neural network with negative message passing and uniformity maximization for graph coloring. *Complex & Intelligent Systems*, 10(3):4445–4455, 2024.

George Watkins, Giovanni Montana, and Juergen Branke. Generating a graph colouring heuristic with deep q-learning and graph neural networks. In *International Conference on Learning and Intelligent Optimization*, pp. 491–505. Springer, 2023.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Yicheng Zhang, Kai Zhang, and Ni Wu. Using graph isomorphism network to solve graph coloring problem. In *2024 4th International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pp. 209–215. IEEE, 2024.

Jiongzhi Zheng, Mingming Jin, Kun He, Jinghui Xue, Li Zhao, Lei Song, and Jiang Bian. Deep node embeddings for the graph coloring problem. *The European Journal on Artificial Intelligence*, pp. 30504554251325151, 2025.

Algorithm 3 MIS with diversity and tree search

Input: Graph $\mathcal{G}$
Output: The best solution found
Initialize queue $\mathcal{Q}$ with $\mathcal{G}$
while execution time is under budget **do**
  $\mathcal{G}' \leftarrow \mathcal{Q}$.random_pop()
  for $m := 1$ to M **do**
    $\mathbf{v} \leftarrow$ argsort$(f^m(\mathcal{G}_t; \theta))$ in descending order
    for $i := 1$ to $\|\mathbf{v}\|$ **do**
      if vertex $\mathbf{v}(i)$ is already labeled in $\mathcal{G}'$ **then**
        break
      end if
      Label $\mathbf{v}(i)$ as 1 in $\mathcal{G}'$
      Label the neighbors of $\mathbf{v}(i)$ as 0 in $\mathcal{G}'$
    end for
    if $\mathcal{G}'$ is completely labeled **then**
      Update the current list of best solutions
    else
      Remove labeled vertices from $\mathcal{G}'$
      Add $\mathcal{G}'$ to $\mathcal{Q}$
    end if
  end for
end while

A Appendix

A.1 Finding MIS using probability map

The approach used to learn $find_multiple_MIS(.)$ function is from the paper given by Li et al. (2018) where they find maximum independent set (MIS) for a graph $G = (V, E)$ by estimating the likelihood for each vertex $v \in V$ of being present in MIS. The network learned uses GCN architecture and takes the graph G on N vertices as input and generates multiple probability maps $\langle f^1, f^2, \ldots, f^M \rangle$.where each probability map on vertices $p : V \to [0, 1]^N$ indicates how likely each vertex is to belong to the MIS.

Using these maps multiple MIS are generated using Algorithm 3 which uses breadth-first approach. For each probability map $f^i(G)$ the vertices are sorted in descending order of likelihood. The vertices are then labeled 1 with its neighbors 0. This process stops when the next vertex in the sorted list is already labeled as 0. Then a residual graph G' is obtained by removing all the labeled vertices and added in queue for further exploration. The exploration of all the intermediate states is further controlled by a time budget. For executing C-GBS as discussed in results section, break in line 10 of algorithm 3 is replaced by continue. Also, in line 7 instead of considering all vertices only top t vertices (sorted in descending order of probability map) are considered with t in range $50\% - 75\%$.

A.2 Experimental setup

Datasets: For training the MIS extracting GCN we use 1000 random graphs generated using the Erdos–Renyi(ER) model $G(n, p)$ where n (number of nodes) is randomly chosen between 10 and 300 and p (probability of edge formation) is randomly chosen between 0.01 and $min\left(0.1, \frac{5}{n}\right)$. p is chosen to ensure that the graphs are sparse for large n keeping it manageable.

Training: We generate multiple labeling solutions for the graphs in training dataset using integer linear programming(ILP) solver called Gurobi(Gurobi Optimization, Inc., 2025). Corresponding to each graph we have a list of binary label vectors, each vector representing one possible MIS solution for that graph. All of these label vectors have same size. We use Adam optimizer with single-graph mini-batches and learning

Table 5: Numerical results for Guided beam search with different metrics.

Graph Name	χ	$\mathbf{M_{ed}}$	$\mathbf{M_{edc}}$	$\mathbf{M_{ce}}$	$\mathbf{M_{acc}}$	$\mathbf{M_{w}}$
fpsol2.i.3	30	30	30	35	37	30
DSJR500.5	?	142	139	161	184	136
DSJC500.9	?	142	141	150	159	141
queen15_15	?	17	17	18	20	17
miles500	20	21	20	23	25	20
DSJC250.9	?	80	81	87	90	78
school1_nsh	?	21	18	20	22	20
DSJR500.1c	?	102	97	106	109	100

rate 10^{-4}. The loss function used is hindsight loss which captures loss from all the M probability maps generated. Hindsight loss for training data $\mathcal{D} = \{(\mathcal{G}_i, \mathbf{1}_i)\}$ and probability maps $< f^1, f^2, ..., f^M >$ is given by

$$\mathcal{L}(\mathcal{D}, \theta) = \sum_i \underset{\text{m}}{min}(l(\mathbf{1}_i, f^m(\mathcal{G}_i; \theta))$$

Network settings: The network settings for training MIS extraction GCN remains same as in paper by Li et al. (2018). The network has $L = 20$ GCN layers and the input feature vector for every vertex is set to $\mathbf{1}$ (vector having all entries equal to 1) and is of size $C_0 = 32$. The widths of the hidden layers are set identical to $C_l = 32$. The width of the output layer $C_L = M$,where M is the number of output probability maps. We use $M = 32$. For the Beam Search algorithm, we experimented with different values of the queue capacity c and found that $c = 4$ provides a good balance between computational cost and performance. Increasing c beyond this value results in negligible improvements. Also, time budget set for predicting MIS is $T = 60s$.

Baselines: We compare the presented approach to several traditional heuristics based algorithms and ML-solvers developed for graph coloring. The heuristics based algorithms used for comparison are LF, SL, DSatur (Brélaz, 1979),RLF (Leighton, 1979),Tabucol (Hertz & de Werra, 1987)

- **LF:** Vertices are ordered in decreasing order of degree and then colored sequentially using the smallest available color not used by their colored neighbors.
- **SL:** The vertices with the smallest degree are removed repeatedly and then coloring is done in reverse order of removal.
- **DSatur:** The algorithm (Brélaz, 1979) begins by coloring the vertex with the highest degree and then at each step an uncolored vertex with the highest saturation degree is selected for coloring. Here, saturation degree of a vertex is the number of different colors used by its already-colored neighbors.
- **RLF:** The algorithm (Leighton, 1979) tries to form large independent sets iteratively producing fewer colors. To form independent set, a vertex of maximum degree is chosen followed by iteratively adding vertices that are not adjacent to any vertex in the current color class but are most adjacent to vertices outside it.
- **Tabucol:** is a tabu search–based algorithm (Hertz & de Werra, 1987) designed to minimize the number of color conflicts in a graph.It starts from an initial (possibly infeasible) coloring and iteratively improves it by recoloring vertices while avoiding cycles and local minima through the use of a tabu list.

Further, the ML based solvers used for comparison are PI-GCN and PI-SAGE (Schuetz et al., 2022b),GDN (Li et al., 2022b), GIN (Zhang et al., 2024) and GNN-NU (Wang et al., 2024). All of these solvers differ in the type of GNN used. Given a graph G and a positive integer k, all of them try to obtain coloring using k colors minimizing conflicts of colors between adjacent vertices. For testing, the best known value of k for

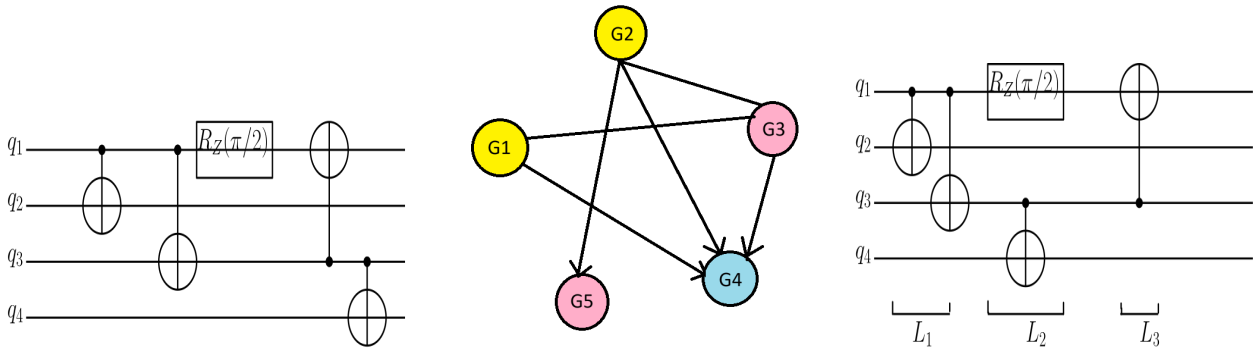

Figure 5: Sample circuit C ($depth = 5$), its corresponding dependency graph G and the optimized circuit C' ($depth = 3$).

a graph G is given and the number of conflicts in the obtained solution are reported. All the methods are trained with 2×Nvidia A100 card GPUs and evaluated on a single NVIDIA A100 GPU with 2× Intel Xeon Platinum 8358 (32 cores 2.6 GHz) CPU.

A.3 Metric analysis

We investigated the performance of Beam search using four different metrics for $update_queue(.)$ function. These include edge density M_{ed}, edge density with colors used M_{edc}, color efficiency M_{ce} and average closeness centrality M_{acc}. For 60% of the graphs, the colors obtained by all the four metrics were almost identical differing by at most 2 colors. However, significant variations were observed in results obtained for graphs with higher vertex count and edge density. For such graphs the edge density with colors used metric performed the best. Some of the results obtained are listed in table 5.

A.4 Dependency graph construction for Quantum Circuit

In this section we explain in detail how a dependency graph is generated from a quantum circuit and further how the circuit is optimized using optimal coloring. Consider the sample circuit C given in figure5, it consist of 4 qubits and 5 gates. To generate the corresponding dependency graph, the gates are labeled as follows:-

- $G1$: $CNOT(q_1, q_2)$
- $G2$: $CNOT(q_1, q_3)$
- $G3$: $R_Z(\pi/2)(q_1)$
- $G4$: $CNOT(q_3, q_1)$
- $G5$: $CNOT(q_3, q_4)$

Now, to obtain coloring for this mixed graph, we first obtain a corresponding undirected graph G_u selecting nodes having indegree equal to 0. Thus, G_u consist of nodes $G1, G2$ and $G3$. The algorithm chooses MIS $\{G1, G2\}$. Next this MIS is removed and now indegree of $G5$ becomes 0. Thus, in next iteration the subgraph with nodes $G3$ and $G5$ is selected and algorithm obtains MIS $\{G3, G5\}$. Finally node $G4$ is left which is colored with a new color. Thus, coloring obtained uses three colors as shown in figure 5. Nodes with same color labels can be executed in parallel and we obtain an optimized circuit with depth 3.

For comparison we obtain circuit depth for test instances using Qiskit. Qiskit is an open-source quantum computing software development kit used to design, simulate, compile, and run quantum programs on both simulators and real quantum hardware. In Qiskit, the DAG depth method is a way to compute the circuit

depth by first converting a quantum circuit into a Directed Acyclic Graph (DAG) and then finding the length of the longest dependency path in that graph. For the sample circuit C we obtain DAG given in figure 6.

The circuit consist of two types of gates $CNOT(.,.)$ and single-qubit rotation gate $R_Z(\pi/2)(.)$. For a $CNOT(q_1, q_2)$ gate, q_1 is control qubit and q_2 is target qubit. $CNOT$ gates can commute only if they have disjoint qubits or same control and different targets. Thus, we obtain the corresponding dependency graph G with five vertices and following edges:-

- e_1: undirected edge between $G1$ and $G3$ because the two gates commute and share qubit $q1$.
- e_2: directed edge $(G1, G4)$ because the two gates cannot commute as $q1$ is control qubit for $G1$ and target qubit for $G4$.
- e_3: undirected edge between $G2$ and $G3$ because the two gates commute and share qubit $q1$.
- e_4: directed edge $(G2, G4)$ because the two gates cannot commute as $q1$ is control qubit for $G2$ and target qubit for $G4$.
- e_5: directed edge $(G2, G5)$ because the two gates cannot commute as $q3$ is control qubit for $G5$ and target qubit for $G2$.
- e_6: directed edge $(G3, G4)$ as the two gates do not commute.

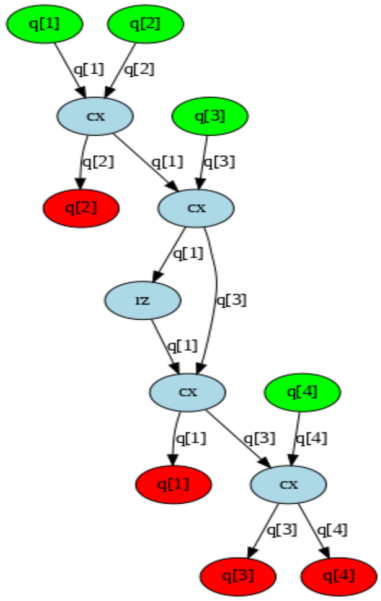

Figure 6: Directed acyclic graph (DAG) obtained for Sample circuit C in figure 5 using qiskit.

A.5 Quantum circuit depth optimization algorithm

Theorem A.1. *The optimized circuit C' obtained from above algorithm is executable that is the coloring obtained is suitable.*

Proof: Let ϕ be the coloring obtained from the algorithm. For each directed edge $(u, v) \in E_d$, v will have indegree 0 only if u is chosen in some independent set that is u is colored first. Thus, $\phi(u) < \phi(v)$ holds for coloring ϕ. Also for each undirected edge $\{u, v\} \in E_u$ either u and v belongs to G_{exec} or not. If not then they will always receive distinct colors. If both are present simultaneously in G_{exec} then also they will not be selected together in an independent set. Thus, $\phi(u) \neq \phi(v)$ holds and we can conclude that coloring ϕ is suitable and thus the circuit C' obtained from ϕ will be executable.

Algorithm 4 Guided Beam Search for Quantum circuit depth optimization

Input: Quantum circuit C, queue capacity c
Output: Quantum Circuit C' with minimized circuit depth
Generate dependency graph $G(V, E_d, E_u)$ from circuit C
Initialize queue Q with capacity c
Initialize $min_colors \leftarrow \infty$
Enqueue $(G, 0)$ into Q
while Q is not empty **do**
  $(G_{current}, num_colors) \leftarrow$ Dequeue from Q
  Find $G_{exec}(V', E'_u)$ from $G_{current}$ where V' are vertices with indegree 0
  and E'_u are undirected edges between them
  if G_{exec} has no vertices **then**
    $min_colors \leftarrow \min(min_colors, num_colors)$
    continue
  end if
  $MIS_list \leftarrow$ find_multiple_MIS(G_{exec})
  for each MIS in MIS_list **do**
    $G_{new} \leftarrow G_{current}$ with MIS removed
    Enqueue $(G_{new}, num_colors + 1)$ into Q
  end for
  update_queue(Q)
end while
Construct circuit C' by arranging gates in sequence of increasing color label.
Gates with same color label are arranged in parallel.
return Circuit C'

A.6 Results for DIMACS dataset

Table 6: Coloring results and comparison with traditional heuristic algorithms. Number of colors used by optimal coloring obtained from each solver is reported. Metric used for GBS and C-GBS is edge density with colors used. Here, time taken by C-GBS is comparable with traditional heuristics.

Graph Name	$\|V\|$	$\|E\|$	χ	**GBS**	**W-GBS**	**C-GBS**	**RLF**	**DSatur**	**LF**	**SL**
jean	80	254	10	10	10	11	10	10	10	10
games120	120	638	9	9	9	9	9	9	9	9
miles250	128	387	8	9	8	10	8	8	8	8
fpsol2.i.3.col	425	8688	30	30	31	31	30	30	30	30
le450_5a	450	5714	5	8	7	12	8	10	11	11
le450_5b	450	5734	5	8	8	13	8	9	12	13
le450_15b	450	8169	15	19	19	20	16	16	18	18
le450_25a	450	8260	25	27	27	27	25	25	25	25
le450_25b	450	8263	25	27	27	28	25	25	25	25
le450_5d	450	9757	5	5	5	10	5	12	14	11
le450_5c	450	9803	5	5	5	10	5	10	12	12
fpsol2.i.2.col	451	8691	30	30	31	31	30	30	30	30
fpsol2.i.1.col	496	11654	65	65	65	65	65	65	65	65
homer	561	1629	13	13	13	14	13	13	13	13
queen8_8	64	728	9	10	9	11	11	12	13	14
queen8_12	96	1368	12	13	13	14	13	14	16	16
DSJC125.1	125	1472	?	6	5	7	6	6	7	7
miles500	128	1170	20	21	20	21	20	20	20	20
miles750	128	2113	31	32	32	35	31	31	32	31
queen13_13	169	6656	13	15	15	18	16	17	22	20
mulsol.i.3.col	184	3916	31	31	31	31	31	31	31	31
myciel7	191	2360	8	8	8	8	8	8	8	8
queen14_14	196	8372	?	16	16	19	17	19	23	22
zeroin.i.2.col	211	3541	30	30	30	31	30	30	30	30
queen15_15	225	10360	?	17	17	20	19	21	25	23
DSJC250.1	250	6436	?	10	10	11	10	10	11	12
queen16_16	256	12640	?	19	18	22	19	23	26	25
school1	385	19095	?	19	18	38	27	17	32	15
anna	38	493	11	12	11	12	11	11	11	11
queen9_9	81	2112	10	10	10	13	12	13	15	15
queen10_10	100	2940	?	11	11	14	13	14	16	16
queen11_11	11	3960	11	13	13	16	14	15	19	18
DSJC125.5	125	7782	?	20	20	23	20	22	23	25
queen12_12	144	5192	?	14	14	16	15	16	20	19
DSJC250.5	250	31366	?	33	31	41	34	37	41	39
DSJR500.5	500	62624	?	142	136	156	133	130	134	127
latin_square_10	900	307350	?	112	110	131	126	132	204	204
DSJC1000.9	1000	898898	?	252	248	279	282	299	313	319