RETHINKING GRAPH NEURAL NETWORKS FOR GRAPH COLORING

Anonymous authors
Paper under double-blind review

ABSTRACT

The development of graph neural networks (GNNs) stimulated the interest in GNNs for NP-hard problems, while most works apply GNNs for NP-hard problems by empirical intuition and experimental trials and improve the results with the help of some heuristic designs. We start from a simple contradiction that a graph coloring problem requires two connected and symmetric nodes to be assigned different colors while an expressively powerful GNN always maps the two nodes to the same node embeddings. To characterize the power of GNNs for the graph coloring problem, we first formalize the discrimination power of GNNs as the capability to assign nodes different colors. We then study a popular class of GNNs, called AC-GNN, and identify the node pairs that AC-GNNs fail to discriminate and provide corresponding solutions to discriminate them. We show that any AC-GNN is a local coloring method and any local coloring method is non-optimal. Moreover, we discuss the color equivalence of graphs and give a condition for the color equivalent AC-GNN theoretically. Following the approaches which prove to enhance the discriminative power, we develop a simple architecture for the graph coloring problem without heuristic pre-processing and post-processing procedures. We empirically validate our theoretical findings and demonstrate that our model is discriminatively powerful and even comparable with state-of-the-art heuristic algorithms.

1 INTRODUCTION

Graph neural networks (GNNs) have shown overwhelming success in various fields, such as molecules, social networks and web pages, by learning the representation of graph structured data (Hamilton et al., 2017b). The main idea behind GNNs is a neighborhood aggregation scheme (or called message passing), where each node aggregates feature vectors from its neighbors and combines them with its own feature vector producing a new one. A GNN following such scheme is also called aggregation-combination GNN (AC-GNN) (Barceló et al., 2019). After finite iterations of such aggregation and combination, the corresponding feature vector of each node is called node embedding to represent the node.

The development of GNNs stimulated the interest in GNNs for NP-hard problems (Li et al., 2018; Lemos et al., 2019; Huang et al., 2019), e.g. graph coloring problem. Although most works apply GNNs with the help of empirical intuition and some heuristic designs, some recent works (Loukas, 2019; Barceló et al., 2019; Xu et al., 2018) explore the theoretical properties of the general GNNs. These theoretical studies help to understand the GNNs and investigate their properties, but unfortunately they do not pay special attention to the application for the NP-hard problems and some objectives are even contrary thus result in poor results in some cases. One example is shown in Figure 1(a), where an expressively powerful GNN maps the symmetric nodes c and d to the same embedding while they are expected to be colored differently in the graph coloring problem.

In this paper, we study the capability of AC-GNNs especially for the graph coloring problem through a theoretical framework. Generally, we aim to answer questions of whether an AC-GNN can be an optimal coloring method for all graphs, when a GNN cannot be optimal and what properties a GNN should possess to avoid non-optimal cases. To answer these questions, we first formalize the discrimination power of an AC-GNN as the capability to assign nodes different colors. A coloring method can be then decided as non-optimal when it fails to discriminate some distinct node pairs in which the node colors are different in all optimal solutions. We then prove that AC-GNN is not

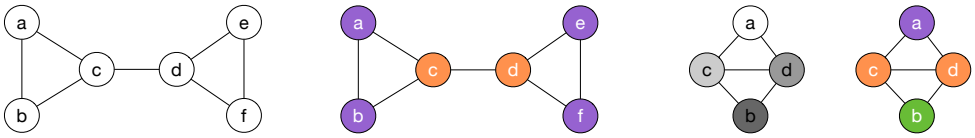

Figure 1: Examples of graph structures in which AC-GNNs fail to discriminate the node pair $\{c, d\}$. (a) left: the input graph with the same node attribute; right: the coloring results by the most powerful AC-GNN. (b) left: the input graph with different node attributes (represented by the gray scale); right: the coloring results by the most powerful integrated AC-GNN.

optimal if the attributes are the same among nodes or the aggregation and combination steps are integrated. Furthermore, we follow the study of locality in AC-GNN (Barceló et al., 2019; Xu et al., 2018) to show that AC-GNN is a local coloring method and derive a lower bound of the chromatic number of any local coloring method, which proves to be strictly larger than the optimal one and thus confirms the non-optimality of AC-GNN for the graph coloring problem. Besides the analysis of the optimality, we also investigate the color equivalence of graphs and give a condition for the color equivalent AC-GNN. Considering all propositions above, we propose a simple variation of AC-GNN, *Graph Discrimination Network (GDN)*. Also, we provide a simple version of GDN by assigning suitable values to the variables to deepen the network.

Overall, in this work, (1) we identify the node pairs that cannot be discriminated by AC-GNN and provide solutions to discriminate them; (2) we show that any AC-GNN is a local coloring method and any local coloring method is non-optimal, and thus prove the non-optimality of AC-GNN; (3) we discuss the color equivalence of graphs and gives a condition for the color equivalent AC-GNN; (4) we propose a simple variation of AC-GNN, *graph discrimination network (GDN)* which avoids the non-optimal cases discussed.

2 Preliminaries

We leave the basic notations and graph terminology in Appendix A.

Graph coloring. The graph coloring problem is to assign colors to the vertices of a graph, such that the color assigned to each vertex is as different as possible from its neighbors. To be more specific, let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent the input graph and each vertex $v \in \mathcal{V}$ is associated with an attribute $\boldsymbol{x}_v$, a function $f_k : (v, \mathcal{G}, \boldsymbol{x}_v) \to \{1, ..., k\}$ for graph coloring returns a color of v indexed by $col_v \in \{1, ..., k\}$. Given a graph $\mathcal{G}$ colored by f, a *conflict function* $c : (u, v, f) \to \{0, 1\}$ is used to measure the performance of f on $\mathcal{G}$. Specifically, $c(u, v, f) = 1$ when u and v are connected and assigned the same color:

$$c(u, v, f) = \begin{cases} 1, & \text{if } f(v, \mathcal{G}, \boldsymbol{x}_v) = f(u, \mathcal{G}, \boldsymbol{x}_u) \text{ and } \{u, v\} \in \mathcal{E}; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The node pair $\{u, v\}$ is called a *conflict* if $c(u, v, f) = 1$. The objective of the graph coloring problem is widely formulated in two ways: 1) Given an available color number constraint k, minimize the number of conflicts as in Equation (2); 2) Given a conflict constraint c_{max}, minimize the number of used colors as in Equation (3).

$$\min \sum_{\{u,v\} \in \mathcal{E}} c(u, v, f_k). \tag{2}$$

$$\min k, \text{ s.t. } \sum_{\{u,v\} \in \mathcal{E}} c(u, v, f_k) \leq c_{max}. \tag{3}$$

When we set c_{max} as 0, i.e., no conflict is introduced by f_k, we call the obtained minimum color number as the *chromatic number* of $\mathcal{G}$. In this paper, we focus on the k-coloring problem, in which k is the chromatic number of $\mathcal{G}$. In the following pages, we say a coloring function is *optimal* if it colors the graph without conflict in the k-coloring problem.

Graph neural networks. GNNs are to learn the embedding of a node or the entire graph based on the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and node features $\{\boldsymbol{x}_v : v \in \mathcal{V}\}$. We follow the same notations in a

previous work Barceló et al. (2019) to formally define the basics for GNNs. Let $\{\text{AGG}^{(i)}\}_{i=1}^{L}$ and $\{\text{COM}^{(i)}\}_{i=1}^{L}$ be two sets of *aggregation* and *combination* functions. An *aggregation-combine GNN* (AC-GNN) computes the feature vectors $\boldsymbol{h}_v^{(i)}$ for every node $v \in \mathcal{V}$ by:

$$\boldsymbol{h}_v^{(i)} = \text{COM}^{(i)}(\boldsymbol{h}_v^{(i-1)}, \text{AGG}^{(i)}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}(v)\})), \tag{4}$$

where $\mathcal{N}(v)$ denotes the *neighborhood* of v, i.e., $\mathcal{N}(v) = \{u : \{u, v\} \in \mathcal{E}\}$ and $\boldsymbol{h}_v^{(0)}$ is the node attribute $\boldsymbol{x}_v$. Finally, each node v is classified by a node classification $CLS(\cdot)$ applied to the node embedding $\boldsymbol{h}_v^{(L)}$. When the AC-GNN is used as the coloring function for the graph coloring problem, $CLS(\cdot)$ returns a $col_v \in \{1, ..., k\}$. Then, an AC-GNN $\mathcal{A}$ with L layers is defined as $\mathcal{A} = (\{\text{AGG}^{(i)}\}_{i=1}^{L}, \{\text{COM}^{(i)}\}_{i=1}^{L}, CLS(\cdot))$. Here, we define $\mathcal{A}(v, \mathcal{G}, \boldsymbol{x}_v)$ as the color of v assigned by $\mathcal{A}$.

The properties of aggregation, combination and classification functions are widely studied in recent years (Barceló et al., 2019; Xu et al., 2018; Li et al., 2020; Morris et al., 2019; Kipf & Welling, 2016) and many variations of these functions are proposed. Among various function architectures, we say an AC-GNN is *simple* as in Barceló et al. (2019) if the aggregation and combination functions are defined as follows:

$$\text{AGG}^{(i)}(\mathbb{X}) = \sum_{\boldsymbol{x} \in \mathbb{X}} \boldsymbol{x}, \tag{5}$$

$$\text{COM}^{(i)}(\boldsymbol{x}, \boldsymbol{y}) = \sigma(\boldsymbol{x}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)}), \tag{6}$$

where $\boldsymbol{C}^{(i)}$, $\boldsymbol{A}^{(i)}$ and $\boldsymbol{b}^{(i)}$ are trainable parameters, σ is an activation function.

The aggregation and combination functions can also be integrated such as the networks studied in Kipf & Welling (2016); Loukas (2019). We say that such AC-GNN is *integrated* and the aggregation and combination functions are integrated as follows:

$$\boldsymbol{h}_u^{(i)} = \text{COM}^{(i)}(\text{AGG}^{(i)}(\{\boldsymbol{h}_w^{(i-1)} : w \in \mathcal{N}(u) \cup \{u\}\})). \tag{7}$$

3 Connection between GNNs and graph coloring

GNNs have been applied to solve a set of graph-related problems, and a few trials have been made for the graph coloring problem. However, GNNs are designed in a local aggregation scheme, which has a totally different objective from the graph coloring context, i.e., GNNs often map connected nodes into similar node embeddings while the graph coloring function assigns the connected nodes to different colors. In this section, we theoretically study the connection between GNNs and the graph coloring problem through the discussions and proofs of several distinct characteristics. We first formalize the power of GNNs for the graph coloring problem. We then exploit the locality of AC-GNN and give conditions on how the power of AC-GNN can be enhanced for the graph coloring problem. Furthermore, we study the color equivalence of the graph coloring problem and, in reverse, seek the ways to make the AC-GNN color equivalent.

3.1 Discrimination power

Recent works (Xu et al., 2018; Barceló et al., 2019; Morris et al., 2019) study the expressive power of a GNN by analyzing when a GNN maps two nodes to the same node embedding. In the study of the expressive power, a GNN is called maximally powerful if it maps two nodes to the same node embedding only if they are local equivalent. Nevertheless, the analysis of the expressiveness power is not suitable for the graph coloring problem. On the contrary, a GNN with the maximum expressiveness power may harm the results of graph coloring: Two local equivalent nodes are often assigned different colors in the optimal solution. To stress the distinction and study the power of a GNN for the graph coloring problem, we introduce the discrimination power of GNNs. Ideally, a GNN with a strong discrimination power maps two connected nodes to node embeddings as different as possible. Formally, we define that a coloring method f discriminates a node pair (u, v) as follows:

Definition 1 (discriminate)**.** *A coloring method f discriminates a node pair (u, v) if the color of each node in the pair assigned by f is different, i.e., $f(u, \mathcal{G}, \boldsymbol{x}_u) \neq f(v, \mathcal{G}, \boldsymbol{x}_v)$.*

Definition 2 (distinct node pair)**.** *A node pair (u, v) is a distinct node pair if the colors of u and v are different in any optimal solution.*

According to the definitions above, we can study the discrimination power by analyzing the distinct node pair, i.e., a GNN with stronger discrimination power discriminates more distinct node pairs. Clearly, a connected node pair is also distinct and an optimal coloring method always discriminates the distinct node pair. One may try to build an optimal AC-GNN which colors all graphs without conflict; however, the following Proposition 1 refutes the existence of such a "perfect" AC-GNN:

Proposition 1. *All AC-GNNs cannot discriminate any equivalent node pair.*

It is not difficult to see that an equivalent node pair can be connected, which makes the node pair distinct. The node pair $\{c, d\}$ in Figure 1(a) is one example of such node pairs. Hence, AC-GNN is not optimal for any graph that contains these node pairs, i.e., connected and also equivalent.

Many previous works use a constant value or degree value as the node attribute, which makes any topologically equivalent node pair also equivalent. To enhance the discrimination power of an AC-GNN, one may impose more difference for each node like a random feature (Sato et al., 2020) or one-hot vector (Barceló et al., 2019).

Actually, this distinct node attribute solution also fits the conclusion in Loukas (2019), which proves that with discriminative attributes GNNs become significantly more powerful. Indeed, the handling of node attribute strengthens the AC-GNN by eliminating the equivalent node pairs (the topological equivalence preserves). However, the superficial methods on the node attributes cannot influence and solve the underlying defects of specific AC-GNNs, for example, the integrated AC-GNN. Intuitively, integrated AC-GNN is reasonable in the clustering tasks, while it may not fit for the graph coloring problem due to its aggregation scheme. Considering that an integrated AC-GNN treats the features of each node and features of its neighbors equally, i.e., all of them are aggregated into the same multi-set, the integrated AC-GNN is more difficult to discriminate the node against its neighbors. Proposition 2 points out that an integrated AC-GNN cannot be optimal since there always exists a graph in which at least one distinct node pair is not discriminated by the integrated AC-GNN.

Proposition 2. *There is a graph in which at least one distinct node pair is not discriminated by any integrated AC-GNN.*

3.2 LOCALITY

With a random feature vector and separated aggregation and combination functions, one may ask whether it is possible to build an optimal AC-GNN. To answer the question, we first introduce the definition of locality in the graph coloring problem:

Definition 3 (local method)**.** *A coloring method f is r-local if it fails to discriminate any r-local equivalent node pair. A coloring method f is local if f is r-local for at least one positive integer r.*

Local methods are widely used in combinatorial optimization such as maximum independent set (MIS) and graph coloring. Along with the study for the local methods, the upper bound of the power of a local method for the MIS problem are investigated. For example, Gamarnik & Sudan (2014) gives an upper bound $1/2 + 1/(2\sqrt{2})$ of the size of a MIS produced by any local method in the random d-regular graph as $d \to \infty$ and Rahman et al. (2017) enhances the bound to $1/2$. A random d-regular graph is a graph with n nodes and the nodes in each node pair are connected with a probability d/n. Starting from the upper bound of a local method for the MIS problem, Corollary 1 states the non-optimality of a local method for the graph coloring problem:

Corollary 1. *A local coloring method is non-optimal in the random d-regular tree as $d \to \infty$.*

Due to the localized nature of the aggregation function in GNNs, an AC-GNN with a fixed number of layers, say L layers, cannot see the structure or information of nodes at a distance further than L, i.e., an AC-GNN maps each node u to the node embedding using the information of the subtree $\mathcal{T}(u, L)$ instead of the global graph structure. Following the non-optimality of a local coloring method stated in Corollary 1 and the localized nature of GNNs, we can reduce our analysis of whether there exists an optimal AC-GNN for graph coloring to the question whether an AC-GNN is a local coloring method. Corollary 2 answers the latter question as yes:

Corollary 2. *AC-GNN is a local coloring method.*

Corollary 1 and Corollary 2 direct give the following theorem:

Theorem 1. *AC-GNN is not optimal.*

To obtain a more discriminatively powerful GNN for the graph coloring problem, the locality of AC-GNN should be broken, i.e., the AC-GNN should be able to discriminate r-local equivalent node pairs. In fact, the locality constraint of AC-GNNs has been widely investigated in previous works and corresponding global GNNs (or at least the GNNs that break the local equality) are proposed (Barceló et al., 2019; Li et al., 2020; You et al., 2019; Pei et al., 2020; Rong et al., 2019). Among these explorations, global features can be introduced to make the whole graph structure visible to the GNN. Nevertheless, these variations of AC-GNN are still local *for the graph coloring problem* if the global features applied also fail to discriminate the local equivalent node pairs. For example, (Barceló et al., 2019) uses a readout operation that assigns each node a global feature calculated by the summation of all node features. The summation feature, however, keeps the same for all nodes and thus fails to help discriminate the local equivalent node pair. On the contrary, some methods succeed in distinguishing the local equivalent node pair without a global mechanism such as DropEdge (Rong et al., 2019) which drops edge randomly in each layer and thus breaks the local equality.

3.3 COLOR EQUIVARIANCE

Equivariance is an important property for a function if it is defined on sets that are *equivariant* to the permutation of the set elements. Many works studied the equivariance of a neural network (Zaheer et al., 2017; Qi et al., 2017). For the network on a graph, the *node equivariance* in a GNN for the node embeddings are studied (Maron et al., 2018; Xu et al., 2018), that is, when the input node set is permuted, the GNN should produce node embeddings with the same permutation.

When we adopt GNN for the graph coloring problem, we should consider not only the order equivariance, but also the color equivariance when the node attribute of each node is the probability distribution of colors as in Braunstein et al. (2006); Lemos et al. (2019). One may argue that colors are treated equally in the graph coloring problem, i.e., the solution of an optimal coloring function is already optimal with no need to follow the same permutation on the input coloring distribution. In fact, this claim holds for the pure graph coloring problem but may not fit for the practical application. For example, in the layout decomposition problem (Jiang & Chang, 2017), each color represents a mask and some features (nodes) are pre-assigned to some specific masks, which makes the color equivariance necessary for the coloring function. To investigate the type of functions to be color equivariant, we first formalize the definition of an equivariant function:

Definition 4 (Maron et al. (2018); Zaheer et al. (2017))**.** *A function $f : \mathbb{R}^k \to \mathbb{R}^k$ is equivariant if $f(\boldsymbol{h})\boldsymbol{P} = f(\boldsymbol{h}\boldsymbol{P})$ for any permutation matrix $\boldsymbol{P} \in \mathbb{R}^{k\times k}$ and feature vector $\boldsymbol{h} \in \mathbb{R}^k$.*

Similarly, in the graph coloring problem, we say a function is *color equivariant* if it is equivariant when the node attribute is the probability distribution of colors. A GNN $\mathcal{A}$ with L layers is color equivalent if and only if all functions in $\{\text{AGG}^{(i)}, \text{COM}^{(i)} : i \in 1, ..., L\}$ are color equivalent. Then, the following theorem states the sufficient and necessary conditions for $\mathcal{A}$ to be color equivalent:

Theorem 2. *Let $\mathcal{A}$ be a simple AC-GNN and both input and output be the probability distribution $\boldsymbol{h} \in \mathbb{R}^k$ of k colors, $\mathcal{A}$ is color equivariant if and only if the following conditions hold:*

- *For any layer i, all the off-diagonal elements of $\boldsymbol{C}^{(i)}$ are tied together and all the diagonal elements are equal as well. That is,*
$$\boldsymbol{C}^{(i)} = \lambda_C^{(i)}\boldsymbol{I} + \gamma_C^{(i)}(\boldsymbol{1}\boldsymbol{1}) \quad \lambda_C^{(i)}, \gamma_C^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^\top \in \mathbb{R}^k. \tag{8}$$
- *For any layer i, all the off-diagonal elements of $\boldsymbol{A}^{(i)}$ are also tied together and all the diagonal elements are equal as well. That is,*
$$\boldsymbol{A}^{(i)} = \lambda_A^{(i)}\boldsymbol{I} + \gamma_A^{(i)}(\boldsymbol{1}\boldsymbol{1}) \quad \lambda_A^{(i)}, \gamma_A^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^\top \in \mathbb{R}^k. \tag{9}$$
- *For any layer i, all elements in $\boldsymbol{b}^{(i)}$ are equal. That is,*
$$\boldsymbol{b}^{(i)} = \beta^{(i)}\boldsymbol{1} \quad \beta^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^\top \in \mathbb{R}^k. \tag{10}$$

The theorem above is actually an extension of (Zaheer et al. (2017), Lemma 3) from a standard neural network layer $f = \epsilon(\Theta\boldsymbol{x})$ to a simple AC-GNN. Similarly, when $\lambda_C^{(i)}, \gamma_C^{(i)}, \lambda_A^{(i)}, \gamma_A^{(i)}$ are the matrices, the result above can be also easily extended to a higher dimension, i.e., the node attribute is not a probability distribution $\boldsymbol{h} \in \mathbb{R}^k$ but a feature with higher arbitrary dimensional $\boldsymbol{h} \in \mathbb{R}^d : d > k$.

4 OUR METHOD

Having developed the conditions for a GNN which is less discriminative powerful and not color equivalent, we summarize a series of rules that make a GNN $\mathcal{A}$ color equivalent and improve its discrimination power as follows: (1) The input graph contains no equivalent node pair (Proposition 1); (2) $\mathcal{A}$ does not integrate the aggregation and combination function (Proposition 2); (3) $\mathcal{A}$ is able to discriminate the local equivalent node pairs (Theorem 1); (4) $\mathcal{A}$ is color equivalent. If $\mathcal{A}$ follows the form of a simple AC-GNN, it should satisfy the conditions in Theorem 2.

We then propose a simple architecture, *Graph Discrimination Network (GDN)*, which satisfies all the rules above and follows the form of the simple AC-GNN. We describe GDN for the graph coloring problem as follows:

Forward Computation For a k-coloring problem, given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, each node $v \in \mathcal{V}$ is assigned an discriminative attribute $\boldsymbol{x}_v \in \mathbb{R}^k$. $\boldsymbol{x}_v$ is normalized to represent the probability distribution of k colors and is initialized randomly to eliminate the equivalent node pairs as in Braunstein et al. (2006); Lemos et al. (2019).

In the aggregation function, let $\boldsymbol{m}_v^{(i)} \in \mathbb{R}^k$ be the result returned by $\text{AGG}^{(i)}$ for the node v in the i-th layer, the aggregation layer is organized as follows:

$$\boldsymbol{m}_v^{(i)} = \sum_{u \in \mathcal{N}'(v)} \boldsymbol{h}_u^{(i-1)}, \tag{11}$$

where $\mathcal{N}'(v)$ is defined as the subset of $\mathcal{N}(v)$, in which only the nodes with degree larger or equal to k are preserved with the probability p. A degree constraint follows the fact that a node with degree less than k is always colorable without conflict. We also use a random dropout mechanism proposed in Rong et al. (2019) so that GDN can discriminate the local equivalent node pair by removing edges randomly in each layer. Meanwhile, the global graph structure is still invisible to our model. It would be interesting to characterize how a global feature can be applied in the discrimination of the local equivalent node pair. We leave the question for future work.

In the combination function, we define the $\text{COM}^{(i)}$ following Theorem 2 to make GDN color equivalent:

$$\boldsymbol{h}_v^{(i)} = \sigma(\boldsymbol{h}_v^{(i-1)}\lambda_C^{(i)} + \boldsymbol{h}_v^{(i-1)}\gamma_C^{(i)}(\mathbf{11}) + \boldsymbol{m}_v^{(i)}\lambda_A^{(i)} + \boldsymbol{m}_v^{(i)}\gamma_A^{(i)}(\mathbf{11}) + \beta^{(i)}\mathbf{1}). \tag{12}$$

Generally, there exist many other GNNs which satisfy the rules above and GDN is only one possible example.

Loss Function Considering that an optimal color solution is still optimal after the permutation of colors, i.e., there exist many different optimal solutions for each graph, it is not an easy job to develop a supervised training scheme. Li et al. (2018) uses supervised learning followed by a heuristic tree search to alleviate the multi-solution issue in the MIS problem. However, the complexity of the tree search explodes when the number of colors k and the number of nodes n become large. Here, we use a self-supervised loss which mimics the cost of the graph coloring problem specified in Equation (2). The objective function is motivated by the intuition that the product of probability distributions of connected nodes should be minimized and formulated by:

$$\min \sum_{\{u,v\} \in \mathcal{E}} \boldsymbol{h}_u \cdot \boldsymbol{h}_v. \tag{13}$$

where $\boldsymbol{h}_u$ is the probability distribution obtained by $\mathcal{A}$ when $\boldsymbol{h}_u \in \mathbb{R}^k$. If the dimension is not k, $\boldsymbol{h}_u$ is normalized to measure the similarity.

Dealing with Depth Here GDN is still based on a neighbor aggregation-based scheme, that is, the structural information is constrained by the depth of GDN , i.e., the number of layers (Loukas, 2019; Xu et al., 2018). We set the depth of GDN following the Proposition proven by Loukas (2019); Abboud et al. (2016), which gives the lower bound of the depth for a GNN to be optimal in the graph coloring problem:

Proposition 3 (Loukas (2019); Abboud et al. (2016))**.** *There exists a graph $\mathcal{G}$ on which every AC-GNN of width $w = O(1)$ requires depth at lest $d = \Omega(n^2/\log^2 n)$ to solve the coloring problem optimally.*

Graph	Nodes	Edges	GNN-GCP		Tabucol		Ours	
			Cost	Time(s)	Cost	Time(s)	Cost	Time(s)
Cora	2708	5429	1291	3.90	31	15410	**3**	4.80
Citeseer	3327	4732	1733	2.74	6	44700	**3**	4.88
Pubmed	19717	44338	4393	4.50	NA	>24h	**35**	21.71
Power Law Tree	7856	7756	4519	10.47	**33**	4417	465	36.76
Small World	7856	29716	5563	7.56	**64**	2021	235	171.5
Holme and Kim	7856	15712	8443	7.99	**87**	5317	506	48.46
Layout	641202	787242	386009	3896	**2392**	82301	4626	8474

Table 1: Results of citation datasets, random dataset and the layout dataset by different coloring methods.

Here n is the number of nodes in $\mathcal{G}$ and w is the dimension of the features. According to the lower bound of the depth $d = \Omega(n^2/\log^2 n)$, the required number of layers explodes as the graph goes larger. However, deep AC-GNN suffers from the severe vanishing gradient problem (Li et al., 2019). To solve this problem, we propose a simpler version of GDN : we assign constant values to $\lambda_C^{(i)}, \gamma_C^{(i)}, \lambda_A^{(i)}, \gamma_A^{(i)}$ and $\beta^{(i)}$, which makes GDN free of training and thus free of the vanishing gradient problem. The values are assigned as follows:

$$\lambda_C^{(i)} = 1, \gamma_C^{(i)} = 0, \lambda_A^{(i)} = -\delta, \gamma_A^{(i)} = 0, \beta^{(i)} = \delta/k, \tag{14}$$

where δ is the manually-defined step rate and k is the number of available colors to ensure that the summation of $\boldsymbol{h}_v^{(i+1)}$ remains 1. We then prove that such value assignment, at least, always decreases the cost (although it may not be optimal):

Proposition 4. *The cost monotonically decreases under specific parameters* $\lambda_C^{(i)} = 1, \gamma_C^{(i)} = 0, \lambda_A^{(i)} = -\delta, \gamma_A^{(i)} = 0$.

There are no trainable parameters in the simpler version, which avoids the time-consuming training procedures but may not be the best assignment scheme for these parameters. An interesting direction for future work is to seek the solution for the vanishing gradient problem and keep the model trainable.

5 Experimental Results

Datasets We test our models and baselines on four datasets: (1) The Layout dataset from the layout decomposition problem (Jiang & Chang, 2017); (2) The citation datasets (Cora, Citeseer, Pubmed) (Sen et al., 2008) that are widely used in node classification tasks, and we here regard them as the coloring scenario for large but sparse graphs, hence dismissing their original node attributes; (3) Three random graph distributions namely: random power-law tree, Watts-Strogatz small-world and Holme and Kim model (Watts & Strogatz, 1998; Holme & Kim, 2002); (4) COLOR dataset[1] containing small and medium sized graphs.

Baselines We compare our models with two previous works which focus on the graph coloring problem and four state-of-the-art variants of AC-GNNs: (1) GNN-GCP (Lemos et al., 2019), combing GNN, RNN, and MLP to obtain the node embedding and using a k-means method to color the node; (2) Tabucol (Hertz & de Werra, 1987), a well-known heuristic algorithm using Tabu search. We also compare different variants of AC-GNN in previous works: GCN (Kipf & Welling, 2016), GAN (Veličković et al., 2017), GIN (Xu et al., 2018) and GraphSAGE (Hamilton et al., 2017a).

More details on the datasets and model configurations can be found in Appendix H.

5.1 Comparison with other AC-GNN variations

The comparison with other trainable AC-GNN variations is conducted on the layout dataset. Specifically, we only evaluate the AC-GNN based models on the layout dataset since other datasets are composed of either (1) a single graph, or (2) graphs with varying chromatic numbers, which are not

[1] https://mat.tepper.cmu.edu/COLOR02/

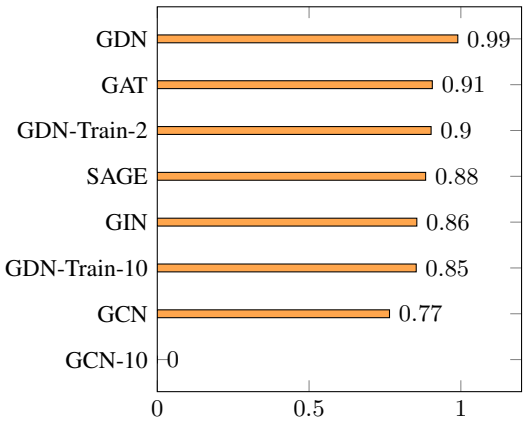

Figure 2: Solved ratio by different AC-GNN variations.

Graph	k	GNN-GCP		Tabucol		Ours	
		Cost	Time	Cost	Time	Cost	Time
jean	10	76	0.06	**0**	0.95	**0**	3.77
anna	11	87	0.08	**0**	3.23	1	5.61
huck	11	117	0.05	**0**	0.15	**0**	4.17
david	11	NA	NA	**0**	4.83	1	5.27
homer	13	1628	1.09	**0**	274	1	12.60
myciel5	6	35	0.04	**0**	0.20	**0**	1.57
myciel6	7	94	4.33	**0**	0.79	**0**	3.12
games120	9	301	0.07	**0**	0.93	**0**	7.28
Mug88_1	4	146	0.33	**0**	0.12	**0**	0.39
1-Insertions_4	5	42	0.05	**0**	0.16	**0**	0.82
2-Insertions_4	4	360	0.09	**1**	255	**1**	0.75
Queen5_5	5	37	0.03	**0**	0.13	**0**	0.68
Queen6_6	7	290	0.38	**0**	4.93	4	0.87
Queen7_7	7	126	0.04	10	36.9	**15**	1.03
Queen8_8	9	188	0.05	**8**	61.3	7	1.31
Queen9_9	10	296	0.07	5	97.8	**13**	1.99
Queen8_12	12	260	0.10	10	139	**7**	3.02
Queen11_11	11	396	0.10	**33**	213	**33**	2.54
Queen13_13	13	728	0.20	42	401	**40**	3.70

Table 2: Results of COLOR dataset by different coloring methods.

suitable for training. The results are shown in Figure 2, where the value of each bar represents the solved ratio, defined as the ratio between the number of edges without introducing conflicts and the number of total edges. For example, given a graph with 100 edges, if a coloring function colors the graph with 10 conflicts, then the solved rate is calculated by $(100-10)/100 = 0.9$. "GDN-Train-2" represents the trainable GDN with a depth of 2. According to the results, we observe the following: (1) The trainable AC-GNNs, such as GCN and GDN, face a severe performance degradation when the model becomes deep. The more complex the model is (the larger number of variables the model has), the more severe the degradation happens; (2) The variant of the simple AC-GNN, GCN, has the poorest coloring performance, which verifies our theoretical analysis; (3) Our model demonstrates the best discriminative power with the fewest number of parameters, as expected.

5.2 Comparison with other graph coloring methods

The comparison with other graph coloring methods is conducted on all collected datasets. The results are shown in Table 1 and Table 2, where k is the number of available colors and cost is the number of conflicts in the coloring result. GNN-GCP gives NA if it fails to find a chromatic number prediction, while Tabucol gives NA if it fails to color the graph within 24 hours. According to the results, we observe the following: (1) Our model is much more discriminatively powerful than GNN-GCP, with an acceptable sacrifice of efficiency. (2) Our model, which is designed as a pure end-to-end network without any heuristic preprocess or post-process methods, shows a competitive coloring results even compared to the state-of-the-art heuristic algorithm, especially on relatively large or complex graphs.

6 Conclusion

In this paper, we developed theoretical foundations for reasoning about the discriminative power of GNNs for the graph coloring problem. We further identified the node pairs that a popular class of GNNs, AC-GNNs fail to discriminate and gave conditions on how an AC-GNN can be more discriminative powerful. Moreover, we analyzed the color equivalence in the graph coloring problem and proposed a scheme to make AC-GNN color equivalent. Considering the conditions and schemes that are demonstrated to help enhance the discriminative power theoretically, we designed a simple variant of AC-GNN for the graph coloring problem. To complete the picture, it would be interesting to analyze the global terms for the discriminative power of GNNs.

REFERENCES

Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *International Symposium on Distributed Computing*, pp. 29–42. Springer, 2016.

Dimitris Achlioptas and Assaf Naor. The two possible values of the chromatic number of a random graph. In *ACM Symposium on Theory of computing (STOC)*, pp. 587–593, 2004.

Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Alfredo Braunstein, Marc Mézard, Martin Weigt, and Riccardo Zecchina. Constraint satisfaction by survey propagation., 2006.

David Gamarnik and Madhu Sudan. Limits of local algorithms over sparse random graphs. In *Proceedings on Innovations in theoretical computer science*, pp. 369–376, 2014.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 1024–1034, 2017a.

William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

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

Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.

Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with alphagozero. *arXiv preprint arXiv:1902.10162*, 2019.

Iris Hui-Ru Jiang and Hua-Yu Chang. Multiple patterning layout decomposition considering complex coloring rules and density balancing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 36(12):2080–2092, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *IEEE International Conference on Computer Vision (ICCV)*, pp. 9267–9276, 2019.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding–design provably more powerful GNNs for structural representation learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 539–548, 2018.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

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 *AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

Mustazee Rahman, Balint Virag, et al. Local algorithms for independent sets are half-optimal. *The Annals of Probability*, 45(3):1543–1577, 2017.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations (ICLR)*, 2019.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393 (6684):440–442, 1998.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 3391–3401, 2017.

A GRAPH TERMINOLOGY

Here we list the following graph theoretic terms encountered in our work:

Basic graph teminology. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be graphs on vertex set $\mathcal{V}$ and $\mathcal{V}'$, we define

- *isomorphism*: we say that a bijection $\pi : \mathcal{V} \to \mathcal{V}'$ is an *isomorphism* if any two vertices $u, v \in \mathcal{V}$ are adjacent in $\mathcal{G}$ if and only if $\pi(u), \pi(v) \in \mathcal{V}'$ are adjacent in $\mathcal{G}'$, i.e., $\{u, v\} \in \mathcal{E}$ iff $\{\pi(u), \pi(v)\} \in \mathcal{E}'$.
- *isomorphic nodes*: If there exists the isomorphism between $\mathcal{G}$ and $\mathcal{G}'$, we say that $\mathcal{G}$ and $\mathcal{G}'$ are *isomorphic*.
- *automorphism*: When π is an isomorphism of a vertex set onto itself, i.e., $\mathcal{V} = \mathcal{V}'$, π is called an *automorphism* of $\mathcal{G}$.
- *topologically equivalent*: We say that the node pair $\{u, v\}$ is *topologically equivalent* if there is an automorphism mapping one to the other, i.e., $v = \pi(u)$.
- *equivalent*: $\{u, v\}$ is *equivalent* if it is topologically equivalent by π and $\boldsymbol{x}_w = \boldsymbol{x}_{\pi(w)}$ holds for every $w \in \mathcal{V}$, where $\boldsymbol{x}_w$ is the node attribute of node w.

Local graph teminology. For every positive integer r and every node $u \in \mathcal{V}$, we define $\mathcal{T}_{\mathcal{G}}(u, r)$ as the depth-r neighborhood of u in $\mathcal{G}$, which is induced by a subtree of height r rooted at u. Consider two nodes $u, v \in \mathcal{V}$, we define

- r-*local automorphism*: A bijection $\pi_r : \mathcal{V} \to \mathcal{V}$ is an r-*local automorphism* mapping u to v.
- r-*local topologically equivalent*: the node pair $\{u, v\}$ is r-*local topologically equivalent* if π_r is an isomorphism from $\mathcal{T}_{\mathcal{G}}(u, r)$ to $\mathcal{T}_{\mathcal{G}}(v, r)$.
- r-*local equivalent*: Similarly, $\{u, v\}$ is r-*local equivalent* if it is r-local topologically equivalent by π_r and $\boldsymbol{x}_w = \boldsymbol{x}_{\pi_r(w)}$ holds for every $w \in \mathcal{T}_{\mathcal{G}}(u, r)$.

B Proof of Proposition 1

We first recall the proposition.

Proposition 1. *All AC-GNNs cannot discriminate any equivalent node pair.*

Proof. Let π be the automorphism mapping u to v, here, we propose a stronger proposition:

Proposition 5. *Given an AC-GNN and an equivalent node pair $\{u, v\}$ by π, $\boldsymbol{h}_w^i = \boldsymbol{h}_{\pi(w)}^i$ holds for any iteration i and any node $w \in \mathcal{V}$.*

This apparently holds for $i = 0$ since $\boldsymbol{x}_w = \boldsymbol{x}_{\pi(w)}, \forall w \in \mathcal{V}$. Suppose this holds for iteration j, i.e., $\boldsymbol{h}_w^j = \boldsymbol{h}_{\pi(w)}^j, \forall w \in \mathcal{V}$. By definition, AC-GNN $\mathcal{A}$ produces the feature vector $\boldsymbol{h}_v^{j+1}$ of node v in the $(j+1)_{th}$ iteration as follows:

$$\boldsymbol{h}_v^{(j+1)} = \text{COM}^{(j+1)}(\boldsymbol{h}_v^{(j)}, \text{AGG}^{(j+1)}(\{\boldsymbol{h}_u^{(j)} : u \in \mathcal{N}(v)\})). \tag{15}$$

Since an automorphism π remains the set of edges, i.e., $\{u, v\} \in \mathcal{E}$ iff $\{\pi(u), \pi(v)\} \in \mathcal{E}$, the connection relation between two neighbors is preserved after the permutation by π, that is, $\mathcal{N}(\pi(v)) = \{\pi(u), u \in \mathcal{N}(v)\}$ for any $v \in \mathcal{V}$. Then, the input of $\text{AGG}^{(j+1)}$ for $\pi(v)$ is given by $\{\boldsymbol{h}_u^{(j)} : u \in \mathcal{N}(\pi(v))\}$, which is $\{\boldsymbol{h}_{\pi(u)}^{(j)} : u \in \mathcal{N}(v)\}$. Since $\boldsymbol{h}_w^j = \boldsymbol{h}_{\pi(w)}^j, \forall w \in \mathcal{V}$, the input of $\text{AGG}^{(j+1)}$ for v is equal to the one of $\text{AGG}^{(j+1)}$ for v, i.e., $\{\boldsymbol{h}_{\pi(u)}^{(j)} : u \in \mathcal{N}(v)\} = \{\boldsymbol{h}_{\pi(u)}^{(j)} : u \in \mathcal{N}(v)\}$ and makes their output equal, i.e., $\boldsymbol{m}_v^{j+1} = \boldsymbol{m}_{\pi(v)}^{j+1}$. Therefore, the input of $\text{COM}^{(j+1)}$ for v, $(\boldsymbol{h}_v^{(j)}, \boldsymbol{m}_v^{j+1})$, is also equal to the one of $\text{COM}^{(j+1)}$ for $\pi(v)$, which makes the vector features of v and $\pi(v)$ equal after $(j+1)_{th}$ iteration for any node $v \in \mathcal{V}$ and proves the proposition 8. Thus, the AC-GNN $\mathcal{A}$ always produces the same node embeddings for the nodes in the equivalent node pair, which results in the same color.

□

C Proof of Proposition 2

We first recall the proposition.

Proposition 2. *There is a graph in which at least one distinct node pair is not discriminated by any integrated AC-GNN.*

Proof. The proof starts with a simple fact: a classifier $CLS(\cdot)$ always assign two nodes with the same node embedding to the same category.

Consider the following graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which two connected nodes u and v share the same neighborhood except each other, i.e., $\mathcal{N}(u) \backslash \{v\} = \mathcal{N}(v) \backslash \{u\}$. The node pair $\{u, v\}$ is distinct since they are connected. It follows that the inputs for the node features of u and v after iteration k are exactly the same since $\mathcal{N}(u) \cup \{u\} = \mathcal{N}(u) \backslash \{v\} \cup \{u, v\} = \mathcal{N}(v) \backslash \{u\} \cup \{u, v\} = \mathcal{N}(v) \cup \{v\}$. Therefore, the outputs are the same, which means that $\boldsymbol{h}_u^{(j)} = \boldsymbol{h}_v^{(j)}$ holds for any iteration k and any aggregation and combine functions $\text{AGG}(\cdot), \text{COM}(\cdot)$. Combining with the fact that $CLS(\boldsymbol{h}_u) = CLS(\boldsymbol{h}_v)$ if $\boldsymbol{h}_u = \boldsymbol{h}_v$, the proof is finished. □

D PROOF OF COROLLARY 1

We first recall the corollary.

Corollary 1. *A local coloring method is non-optimal in the random d-regular graph as $d \to \infty$.*

Proof. A random d-regular graph $\mathcal{G}_d^n$ is a graph with n nodes and each node pair is connected with a probability d/n. We start the proof from the following non-trivial proposition:

Proposition 6 (Rahman et al. (2017))**.** *The largest density of factor of i.i.d. independent sets in a random d-regular graph is asymptotically at most $(\log d)/d$ as $d \to \infty$. The density of the largest independent sets in these graphs is asymptotically $2(\log d)/d$.*

The proposition above limits the size of an independent set produced by a local method for the random d-regular graph with an upper bound, $n(\log d)/d$ as $d \to \infty$. Given an upper bound of the independent set, the following corollary on the graph coloring problem is introduced:

Corollary 2. *The lower bound of k with a zero conflict constraint obtained by a local coloring method for the random d-regular graph is $d/\log d$ as $d \to \infty$.*

The proof is based on the Proposition 6: if a local coloring method f obtains a smaller k', s.t. $k' < d/\log d$ by coloring $\mathcal{G}_d^n$ without conflict using k' colors, all node sets classified by the node color will be independent sets and the size of the maximum one will be larger than $(n \log d)/d$, a contradiction with Proposition 6.

The Corollary 2 reveals the lower bound of k by local methods for a random d-regular graph. Another important observation of k by Achlioptas & Naor (2004) specifies that exact value of the chromatic number (i.e., the minimum k) of a random d-regular graph. The proposition is described as follows:

Proposition 7 (Achlioptas & Naor (2004))**.** *Let t_d be the smallest integer t such that $d < 2t \log t$. The chromatic number of a random d-regular graph is either t_d or $t_d + 1$.*

It follows directly from Corollary 2 and Proposition 7 that, we can finish the proof of Corollary 1 by showing that the lower bound of k by local methods is always greater than the exact chromatic number:

$$d/\log d > t_d + 1 \text{ for } d \to \infty. \tag{16}$$

Let $f(t) = 2t \log t$ and define t_0 s.t. $d = f(t_0) = 2t_0 \log t_0$. Since t_d is the smallest integer t such that $d < f(t)$, we have $f(t_0) = d \geq f(t_d - 1)$. Since f is monotonically increasing, $t_0 \geq t_d - 1$ and thus $d/\log d - t_d - 1 \geq d/\log d - t_0 - 2$ always holds. Let $d = 2t_0 \log t_0$, we further transform the objective to:

$$d/\log d - t_d - 1 \geq d/\log d - t_0 - 2 = \frac{2t_0 \log t_0}{\log(2t_0 \log t_0)} - t_0 - 2 > 0, \text{ for } d, t_0 \to \infty.$$

we first prove that

$$\begin{aligned} &\frac{\log t_0}{\log(2t_0 \log t_0)} > 2/3 \\ \Rightarrow &3 \log t_0 > 2 \log(2t_0 \log t_0) \\ \Rightarrow &3 \log t_0 > 2(1 + \log t_0 + \log(\log t_0)) \\ \Rightarrow &\log t_0 > 2 + 2 \log(\log t_0) \text{ when } t_0 \to \infty. \end{aligned}$$

The above inequality holds obviously. Following the objective, we have:

$$\frac{2t_0 \log t_0}{log(2t_0 \log t_0)} - t_0 - 2 > \frac{4}{3} t_0 - t_0 - 2 > 0 \text{ when } t_0 \to \infty.$$

Therefore, we finish the proof.

□

E PROOF OF COROLLARY 2

We first recall the corollary.

Corollary 2. *AC-GNN is a local coloring method.*

Proof. Given an AC-GNN $\mathcal{A}$ with L layers, let's consider a L-local equivalent node pair $\{u, v\}$ in $\mathcal{G}$ by an L-local automorphism π_L, which means that two rooted subtrees $\mathcal{T}_{\mathcal{G}}(u, L)$ and $\mathcal{T}_{\mathcal{G}}(v, L)$ are isomorphic and $\boldsymbol{x}_w = \boldsymbol{x}_{\pi_r(w)}$ holds for every $w \in \mathcal{T}_{\mathcal{G}}(u, r)$. Since two rooted subtrees are isomorphic, the WL test (Weisfeiler & Lehman, 1968) decides $\mathcal{T}_{\mathcal{G}}(u, L)$ and $\mathcal{T}_{\mathcal{G}}(v, L)$ are isomorphic and assigns the same color to w and $\pi_L(w)$ for any $w \in \mathcal{T}_{\mathcal{G}}(u, L)$. To connect the WL test with AC-GNN, the following proposition is used:

Proposition 8 (Morris et al. (2019); Barceló et al. (2019); Xu et al. (2018))**.** *If the WL test assigns the same color to two nodes in a graph, then every AC-GNN maps the two nodes into the same node embedding.*

Therefore, $\mathcal{A}$ maps the u and v into the same node embedding. It follows that $\mathcal{A}$ is L-local and thus local. □

F PROOF OF THEOREM 2

We first recall the theorem.

Theorem 2. *Let $\mathcal{A}$ be a simple AC-GNN and both input and output of each layer in $\mathcal{A}$ be the probability distribution $\boldsymbol{h} \in \mathbb{R}^k$ of k colors, $\mathcal{A}$ is color equivariant if and only if the following conditions hold:*

- *For any layer i, all the off-diagonal elements of $\boldsymbol{C}^{(i)}$ are tied together and all the diagonal elements are equal as well. That is,*

$$\boldsymbol{C}^{(i)} = \lambda_C^{(i)} \boldsymbol{I} + \gamma_C^{(i)} (\boldsymbol{1}\boldsymbol{1}) \quad \lambda_C^{(i)}, \gamma_C^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^T \in \mathbb{R}^k. \tag{17}$$

- *For any layer i, all the off-diagonal elements of $\boldsymbol{A}^{(i)}$ are also tied together and all the diagonal elements are equal as well. That is,*

$$\boldsymbol{A}^{(i)} = \lambda_A^{(i)} \boldsymbol{I} + \gamma_A^{(i)} (\boldsymbol{1}\boldsymbol{1}) \quad \lambda_A^{(i)}, \gamma_A^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^T \in \mathbb{R}^k. \tag{18}$$

- *For any layer i, all elements in $\boldsymbol{b}^{(i)}$ are equal. That is,*

$$\boldsymbol{b}^{(i)} = \beta^{(i)} \boldsymbol{1} \quad \beta^{(i)} \in \mathbb{R} \;\; \boldsymbol{1} = [1, ..., 1]^T \in \mathbb{R}^k. \tag{19}$$

Proof. Let $\text{AGG}^{(i)}$ and $\text{COM}^{(i)}$ be the aggregation and combination functions in the i_{th} layer of $\mathcal{A}$. $\mathcal{A}$ is color equivalent if and only if all functions in $\{\text{AGG}^{(i)}, \text{COM}^{(i)} : i \in 1, ..., L\}$ are color equivalent. the aggregation function is color equivalent clearly and thus we are left to consider the color equivalence of combination functions. Considering the definition of color equivalent in Definition 4, the color equivalence of combination function $\text{COM}^{(i)} = \sigma(\boldsymbol{x}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)})$ is given by:

$$\sigma(\boldsymbol{x}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)})\boldsymbol{P} = \sigma(\boldsymbol{x}\boldsymbol{P}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{P}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)}). \tag{20}$$

$\text{COM}^{(i)}$ is color equivalent if and only if the equation above holds for any permutation matrix $\boldsymbol{P} \in \mathbb{R}^{k \times k}$ and $\boldsymbol{x}, \boldsymbol{y}$. We first find three special cases, which correspond to three conditions respectively:

Case 0. When $\boldsymbol{y} = \vec{0}$, we have that $\sigma(\boldsymbol{x}\boldsymbol{C}^{(i)})\boldsymbol{P} = \sigma(\boldsymbol{x}\boldsymbol{P}\boldsymbol{C}^{(i)})$ holds for any $\boldsymbol{P}$ and $\boldsymbol{x}$. That is, $\boldsymbol{x}(\boldsymbol{C}^{(i)}\boldsymbol{P} - \boldsymbol{P}\boldsymbol{C}^{(i)}) = \vec{0}$ always holds, which reveals that $\boldsymbol{C}^{(i)}\boldsymbol{P} = \boldsymbol{P}\boldsymbol{C}^{(i)}$. $\boldsymbol{C}^{(i)}\boldsymbol{P} = \boldsymbol{P}\boldsymbol{C}^{(i)}$ holds for any $\boldsymbol{P}$ follows that $C^{(i)}_{m,m} = C^{(i)}_{n,n}$ and $C^{(i)}_{m,n} = C^{(i)}_{m,n}$ for any $m, n \in \{1, ..., k\}$. Therefore, all the off-diagonal elements of $\boldsymbol{C}^{(i)}$ are tied together and all the diagonal elements are equal as well.

Case 1. When $\boldsymbol{x} = \vec{0}$, we can prove that all the off-diagonal elements of $\boldsymbol{A}^{(i)}$ are tied together and all the diagonal elements are equal as well following the similar induction in case 1.

Case 2. When $\boldsymbol{x} = \boldsymbol{y} = \vec{0}$, we have that $\sigma(\boldsymbol{b}^{(i)})\boldsymbol{P} = \sigma(\boldsymbol{b}^{(i)})$ holds for any $\boldsymbol{P}$. Therefore, all elements in $\boldsymbol{b}^{(i)}$ are equal.

After proving that these conditions are necessary for a color equivalent $\mathcal{A}$, we proceed to prove that the conditions are already sufficient. Let $\boldsymbol{C}^{(i)} = \lambda_C^{(i)}\boldsymbol{I} + \gamma_C^{(i)}(\mathbf{11})$, $\boldsymbol{A}^{(i)} = \lambda_A^{(i)}\boldsymbol{I} + \gamma_A^{(i)}(\mathbf{11})$ and $\boldsymbol{b}^{(i)} = \beta^{(i)}\mathbf{1}$, $\text{COM}^{(i)}$ is then calculated by:

$$\begin{aligned}
\text{COM}^{(i)}\boldsymbol{P} &= \sigma(\boldsymbol{x}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)})\boldsymbol{P} \\
&= \sigma(\boldsymbol{x}\lambda_C^{(i)}\boldsymbol{I}\boldsymbol{P} + \boldsymbol{x}\gamma_C^{(i)}(\mathbf{11})\boldsymbol{P} + \boldsymbol{y}\lambda_A^{(i)}\boldsymbol{I}\boldsymbol{P} + \boldsymbol{y}\gamma_A^{(i)}(\mathbf{11})\boldsymbol{P} + \beta^{(i)}\mathbf{1}\boldsymbol{P}) \\
&= \sigma(\boldsymbol{x}\boldsymbol{P}\lambda_C^{(i)}\boldsymbol{I} + \boldsymbol{x}\boldsymbol{P}\gamma_C^{(i)}(\mathbf{11}) + \boldsymbol{y}\boldsymbol{P}\lambda_A^{(i)}\boldsymbol{I} + \boldsymbol{y}\boldsymbol{P}\gamma_A^{(i)}(\mathbf{11}) + \beta^{(i)}\mathbf{1}) \\
&= \sigma(\boldsymbol{x}\boldsymbol{P}\boldsymbol{C}^{(i)} + \boldsymbol{y}\boldsymbol{P}\boldsymbol{A}^{(i)} + \boldsymbol{b}^{(i)}). \qquad (21)
\end{aligned}$$

Therefore, $\text{COM}^{(i)}$ is color equivalent if and only if the conditions hold, which completes the proof. □

G Proof of Proposition 4

We first recall the proposition.

Proposition 4. *The cost monotonically decreases under specific parameters* $\lambda_C^{(i)} = 1, \gamma_C^{(i)} = 0, \lambda_A^{(i)} = -\delta, \gamma_A^{(i)} = 0$.

Proof. Consider the derivatives of the cost function with respect to time t. Let $C_{v,j}$ denote the cost induced by node v and j.

$$\begin{aligned}
\frac{\partial E}{\partial t} &= \sum_{v\in\mathcal{V}} \frac{\partial E_v}{\partial \boldsymbol{h}_v} \cdot \frac{\partial \boldsymbol{h}_v}{\partial t} = \frac{1}{2}\sum_{v\in\mathcal{V}}\sum_{j\in\mathcal{N}'(v)} \frac{\partial C_{v,j}}{\partial \boldsymbol{h}_v} \cdot \frac{\partial \boldsymbol{h}_v}{\partial t} \\
&= \frac{1}{2}\sum_{v\in\mathcal{V}}\sum_{j\in\mathcal{N}'(v)} \boldsymbol{h}_j \cdot \frac{\partial \boldsymbol{h}_v}{\partial t} \\
&= \frac{1}{2}\sum_{v\in\mathcal{V}}\sum_{j\in\mathcal{N}'(v)} \boldsymbol{h}_j[(\lambda_C^{(i)} - 1)\boldsymbol{h}_v + \gamma_C^{(i)}(\mathbf{11})\boldsymbol{h}_v + \boldsymbol{m}_v^{(i)}\lambda_A^{(i)} + \boldsymbol{m}_v^{(i)}\gamma_A^{(i)}(\mathbf{11})] \\
&= \frac{1}{2}\sum_{v\in\mathcal{V}} \boldsymbol{m}_v^{(i)}[(\lambda_C^{(i)} - 1)\boldsymbol{h}_v + \gamma_C^{(i)}(\mathbf{11})\boldsymbol{h}_v + \boldsymbol{m}_v^{(i)}\lambda_A^{(i)} + \boldsymbol{m}_v^{(i)}\gamma_A^{(i)}(\mathbf{11})]. \qquad (22)
\end{aligned}$$

To ensure the function E to monotonically decrease, a straightforward way is to guarantee the coefficients of the polynomial above satisfy a set of constraints. Obviously, the choice of $\lambda_C^{(i)} = 1, \lambda_A^{(i)} = -\delta, \gamma_C^{(i)} = \gamma_A^{(i)} = 0$ monotonically decreases the cost function. □

H Details on the experimental setting and results

H.1 Models

GDN. We select a depth following the trend of the graph size, as stated in Loukas (2019). Specifically, the layer number is 10 for the layout testing dataset and 100 for other larger datasets. The trainable GDN is represented by "GDN-train" following a same training strategy with other AC-GNN variations. If not specified, the result of GDN shown in Section 5 stands for the training-free GDN . The step-rate δ is 0.5 and dropedge rate p is 0.5.

GNN-GCP. We trained the GNN-GCP model for 4300 epochs achieving around 70% training accuracy which showed no significant discrepancies from the results in the original paper, in terms of chromatic number predictions.

Tabucol. Following the original configuration, the Tabucol algorithm is assigned with iteration limit of 1000 (or the time limit of 24 hours) and the number of miscolored node pairs is returned if the algorithm fails to find a perfect coloring assignment within the limit.

Other AC-GNN variations. If not specified, all AC-GNN variations have 2 layers and the hidden dimension is 64. We trained the models for 200 epochs and the 10-layer GCN is trained for 30 epochs, in which the epoch number is selected as the one with the best validation results. We use the SGD optimizer with learning rate 0.005 and momentum 0.1 under the layout training dataset.

H.2 DATASETS

Layout. The number of available colors is set to 3. In the comparison with the AC-GNN variations, 80% randomly selected samples is separated into the training dataset (for trainable models) and the testing dataset contains the remaining samples.

Random graphs. The number of available colors is set from 2 to 5 since it is randomly generated. The settings to generate the random graphs are: random power-law tree (γ=3), Watts- Strogatz small-world (k = 4, p = 0.25) and Holme and Kim model (m = 4, p = 0.1), To fit the problem context and prevent miscalculations, we removed all self-loops in the testing graphs and added duplex connections between connected nodes.

Citation datasets. The number of available colors of Cora, Citeseer, and Pubmed is set to 5, 6 and 8 respectively.

The chromatic numbers of graphs in Random and Citation datasets are obtained from the CSP Solver[2].

[2]https://developers.google.com/optimization/cp/cp_solver