Graph coloring with physics-inspired graph neural networks

Martin J. A. Schuetz,[1,2,3] J. Kyle Brubaker,[2] Zhihuai Zhu,[2] and Helmut G. Katzgraber [1,2,3]

[1]*Amazon Quantum Solutions Lab, Seattle, Washington 98170, USA*
[2]*AWS Intelligent and Advanced Compute Technologies, Professional Services, Seattle, Washington 98170, USA*
[3]*AWS Center for Quantum Computing, Pasadena, California 91125, USA*

We show how graph neural networks can be used to solve the canonical graph coloring problem. We frame graph coloring as a multiclass node classification problem and utilize an unsupervised training strategy based on the statistical physics Potts model. Generalizations to other multiclass problems such as community detection, data clustering, and the minimum clique cover problem are straightforward. We provide numerical benchmark results and illustrate our approach with an end-to-end application for a real-world scheduling use case within a comprehensive encode-process-decode framework. Our optimization approach performs on par or outperforms existing solvers, with the ability to scale to problems with millions of variables.

I. INTRODUCTION

The graph coloring problem (GCP) is arguably one of the most famous problems in the field of graph theory [1,2]. Phrased as an optimization problem, the goal is to find an assignment of labels (traditionally referred to as *colors*) to the vertices (nodes) of a graph such that no two adjacent vertices are of the same color, while using the smallest number of colors possible. The convention of using colors dates back to the historic inception of this problem: trying to color a map of the counties of England with the smallest number of colors sufficient to color the map such that no regions sharing a common border would be assigned the same color [1]. Today graph coloring is still an active field of research, with real-world applications across a strikingly wide range of domains, including (for example) the production of sports schedules, the assignment of taxis to customer requests, the creation of timetables at schools and universities, the allocation of computer programming variables to computer registers, air traffic flow management [3], and the game of Sudoku, among others [1].

With online access to first-generation quantum computers steadily expanding, the GCP has recently attracted considerable interest in the broader quantum computing community. In the current era of noisy intermediate-scale quantum (NISQ) devices, typical approaches either involve hybrid quantum-classical algorithms such as the quantum approximate optimization algorithm (QAOA) [4] or quantum annealing [5,6]. Given the low-level access to these devices, the GCP typically has to be cast as a quadratic unconstrained binary optimization problem (QUBO) [7] or, equivalently,

as an Ising Hamiltonian [8], at the expense of increased resource requirements. Specifically, the QUBO description of the GCP with $q > 2$ colors for a graph with n nodes requires $q \times n$ binary variables, or (logical) qubits in the corresponding quantum-native or quantum-inspired approach. In addition, the constraint that each vertex is assigned exactly one color has to be enforced by hand with additional penalty terms [8]. Because of this added overhead due to the binary representation, it would be preferable to tackle the problem in its native mathematical form. In this work we propose the use of graph neural networks to do so, aided by statistical physics concepts.

In the deep learning community, graph neural networks (GNNs) have emerged as a novel class of neural network architectures designed to consume graph structure data [9–16], with the ability to learn effective feature representations of nodes, edges, or even entire graphs. Paradigmatic problems studied with GNNs can be categorized as node classification, link prediction, graph classification, or community detection, among others. Prime examples include the classification of users in social networks [17,18], the prediction of future interactions in recommender systems [19], and the prediction of certain properties of molecular graphs [20,21]. Leaving the details of specific GNN implementations aside (see Refs. [15,22,23] for further details), the underlying theme for GNNs is the implementation of a *message passing* [24] scheme whereby GNNs iteratively update the node (or edge) embeddings by aggregating information from their local neighbors following the topology of the underlying graph. Because of their inherent scalability and graph-based design, GNNs present a platform that can solve the graph coloring problem at scale. We have previously presented a physics-inspired, GNN-based framework to (approximately) solve quadratic unconstrained binary [7] combinatorial optimization problems with up to millions of variables [25]. In this work we natively extend this framework to multicolor decision variables, and show how to solve the graph coloring problem (GCP) without the need for extra penalty terms as needed when using a QUBO-based approach. To this end we
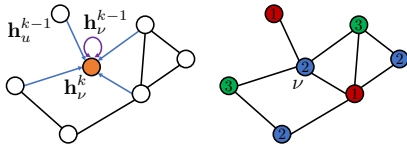
FIG. 1. Schematic illustration of our approach. Following a recursive neighborhood aggregation scheme, the graph neural network is iteratively trained against a loss function based on the Potts model (enforcing different color assignments to adjacent nodes). At training completion, the final values for the soft node assignments at the final graph neural network layer are projected to hard class (color) assignments $\sigma_i = 1, \ldots, q$, as illustrated here for $q = 3$ colors. This solution is optimal as the sample graph contains maximum cliques of size three.

frame the GCP as a multiclass node classification problem and use an unsupervised training strategy based on the Potts model [26], a generalization of the Ising model in statistical physics. For illustration purposes, our approach is schematically depicted in Fig. 1. As discussed in more detail below, generalizations of this approach to applications such as data clustering [27] and community detection [28–30] are straightforward.

The paper is structured as follows. In Sec. II we provide some context for our work, discussing the relevant literature at the cross section between graph coloring and graph neural networks. In Sec. III we describe the basic concepts for our work, with details on the GCP, the physics of the Potts model and its inherent connection to the GCP, and graph neural networks. In Sec. IV we then detail the theoretical framework underlying our approach, providing a comprehensive physics-inspired, GNN-based approach towards solving the GCP and related multicolor optimization problems. Section V outlines an end-to-end application for a real-world scheduling use case, followed by numerical experiments in Sec. VI. Finally, in Sec. VII we draw conclusions and give an outlook on future directions of research.

II. RELATED WORK

In this section we briefly review relevant existing literature, with the goal to provide a detailed context for our work. To keep the scope manageable, we focus on work using GNN-based solution strategies. For an extensive review of the GCP we refer to Ref. [1].

Supervised learning. In Ref. [31] the authors devise a binary classifier to solve the *decision* version of the graph coloring problem, i.e., whether or not a given graph is q colorable. To this end, Lemos *et al.* propose a model that combines a graph neural network with a multilayer perceptron. To train this model a standard binary cross entropy loss function is used, comparing the model's final prediction with the known ground truth for a given GCP instance, as obtained with a complementary CSP solver, albeit for small problem instances only that can be solved exactly. As discussed in Ref. [32], such a supervised approach critically depends on the existence of representative, labeled training data sets with previously optimized hard problem instances, resulting in a somewhat problematic chicken-and-egg scenario. In contrast

to our model, the approach outlined by Lemos *et al.* can underestimate the chromatic number, and—going beyond binary graph classification—requires a heuristic clustering algorithm such as k means in order to provide a constructive coloring solution.

Unsupervised learning. Conceptually, our work is most similar to those approaches that aim to train neural networks in an unsupervised, end-to-end fashion, without the need for labeled training sets. Specifically, Li *et al.* have recently used graph neural networks to solve the GCP following an unsupervised training strategy [33]. With a focus on the formal discriminative power of GNNs for the graph coloring problem and motivated by mere intuition, the authors utilize a loss function which, as we show in this work, follows straightforwardly from the Potts model, and therefore emerges as part of a larger, unifying, *physics-inspired* framework. We also find that our solver improves upon the results of Li *et al.* on several benchmark problems.

Against this background, our work makes a *physics-inspired* contribution to the emerging cross fertilization between combinatorial optimization and machine learning [34,35]. Specifically, we provide a unified framework that pairs the Potts model [26]—as extensively studied in the context of statistical physics—with deep learning tools in the form of graph neural networks to model and solve a large class of graph-based, multicolor optimization problems such as graph coloring, community detection, or data clustering, all within a completely unsupervised, end-to-end framework.

III. PRELIMINARIES

To set up our notation and terminology, we first provide a formal problem definition for the graph coloring problem (GCP). We then highlight its close connection to the Potts model. Finally, we provide a brief review of graph neural networks.

Graph coloring. We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{1, 2, \ldots, n\}$ and edge set $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}\}$. Given such a graph, in the graph coloring problem we seek to assign an integer $c(v) \in \{1, 2, \ldots, q\}$ to every vertex $v \in \mathcal{V}$, such that (i) the assignment is free of color clashes, i.e., $c(u) \neq c(v) \; \forall (u, v) \in \mathcal{E}$, and (ii) the number of colors q is minimal. We refer to a clash-free coloring using at most q colors as a proper (feasible) q coloring. If such a q coloring can be found, the graph is said to be q colorable. The chromatic number of a graph $\mathcal{G}$, denoted as $\chi = \chi(\mathcal{G})$, with $1 \leqslant \chi \leqslant n$, is the minimum of colors required for a feasible coloring of $\mathcal{G}$. Accordingly, our goal is to find the chromatic number χ, with a coloring where adjacent vertices are assigned to different colors. In general, this problem is computationally hard, with exact algorithms displaying an exponential runtime in the size of the input n. Specifically, it is known to be NP hard to compute the chromatic number χ, typically leaving heuristics such as greedy coloring or tabu-search based methods as the go-to approximation strategies [1].

The graph coloring problem is also closely related to yet another NP-hard combinatorial optimization problem, the minimum clique cover problem (MCC) [36]. In MCC the goal is to partition the nodes of a graph into cliques, with as few

cliques as possible. Conversely, graph coloring provides color classes, i.e., partitions of the vertex set into independent sets (that is subsets with no adjacencies), yielding the following equivalence between clique covers and coloring: Because a subset of vertices is a clique in $\mathcal{G}$ if and only if it is an independent set in the complement of $\mathcal{G}$, a partition of the vertices of $\mathcal{G}$ is a clique cover of $\mathcal{G}$ if and only if it is a coloring of the complement of $\mathcal{G}$. For a given graph $\mathcal{G}$, the smallest number for which a clique cover exists is called the clique cover number.

Potts model. The GCP outlined above is closely related to the standard Potts model, as argued below. In the Potts model every vertex is associated with a spin variable $\sigma_i = 1, \ldots, q$ that can take on q different values. The Hamiltonian for the Potts model can be expressed in compact form as

$$H_{\text{Potts}} = -J \sum_{(i,j) \in \mathcal{E}} \delta(\sigma_i, \sigma_j), \tag{1}$$

where $\delta(\sigma_i, \sigma_j)$ refers to the Kronecker delta, which equals one whenever $\sigma_i = \sigma_j$ and zero otherwise, thus capturing the hard-core spin-spin interactions characteristic for the Potts model [26,37]. Accordingly, if two adjacent spins σ_i and σ_j are in the same state, the energy contribution is $-J$, while it is zero whenever they are in different states. To enforce a feasible coloring of the underlying graph, we consider antiferromagnetic interactions and (if not stated otherwise) set $J = -1$ in the following.

Generalizations to settings with weighted interactions J_{ij} (where some constraints are more important than others) are straightforward. Important applications thereof include data clustering [27] and community detection as captured by the maximization of the modularity parameter [28–30], among others. Specifically, we find that the latter can be described by the generalized Potts Hamiltonian

$$H_{\text{Potts}} = - \sum_{i,j} J_{ij} \delta(\sigma_i, \sigma_j), \tag{2}$$

with interaction strength

$$J_{ij} = \frac{1}{2m} \left(A_{ij} - \frac{d_i d_j}{2m} \right), \tag{3}$$

where A_{ij} refers to the adjacency matrix of the graph, $d_i = \sum_j A_{ij}$ is the degree of node i, and $m = 1/2 \sum_i d_i$. Finally, for two colors ($q = 2$) the Potts model reduces to the MaxCut problem, with Hamiltonian $H_{\text{MaxCut}} = \sum_{i<j} J_{ij} z_i z_j$ with $J_{ij} = A_{ij}/2$ and binary spin variables $z_i \in \{-1, 1\}$ [25], as can be seen by the transformation $\delta(\sigma_i, \sigma_j) \rightarrow (1 + z_i z_j)/2$.

The close connection between the GCP and the Potts model becomes apparent in the (dimensionless) partition function [38] of the Potts model, which allows one to compute most thermodynamic variables of a system through derivatives. For the Potts model it is given by

$$\mathcal{Z} = \sum_{\{\sigma_i\}} \exp[-\beta H_{\text{Potts}}], \tag{4}$$

where $\beta = 1/k_B T$ is the inverse temperature (with $T > 0$) and k_B is the Boltzmann constant. Using the relation $\exp[K\delta(\sigma_i, \sigma_j)] = 1 + [\exp(K) - 1]\delta(\sigma_i, \sigma_j)$, we then find in

generality

$$\mathcal{Z} = \sum_{\{\sigma_i\}} \prod_{(i,j) \in \mathcal{E}} [1 + [\exp(K) - 1]\delta(\sigma_i, \sigma_j)]. \tag{5}$$

In the zero-temperature limit we obtain [39]

$$\mathcal{Z} \rightarrow P_{\mathcal{G}}(q) = \sum_{\{\sigma\}} \prod_{(i,j) \in \mathcal{E}} [1 - \delta(\sigma_i, \sigma_j)] \quad \text{for} \ \ T \rightarrow 0. \tag{6}$$

In the last step we have introduced the chromatic function (polynomial) $P_{\mathcal{G}}(q)$, a central quantity in the theory of graph coloring, thus directly relating the Potts model to graph coloring. In the limit $T \rightarrow 0$ adjacent spins are forced to occupy different states, and the partition function $\mathcal{Z}$ simply reduces to the chromatic function $P_{\mathcal{G}}(q)$ which counts the number of possible q colorings of $\mathcal{G}$ as a function of the number of colors q. The chromatic number $\chi = \min\{q \in \mathbb{N} : P_{\mathcal{G}} > 0\}$ is then the smallest positive integer that is not a zero of the chromatic polynomial.

Graph neural networks. Graph neural networks are an emergent family of neural networks that extend the standard deep learning toolbox to graph data [40]. While convolutional neural networks are well defined only over rigid, grid-structured data (such as images), and recurrent neural networks are built for sequences of data (such as text), the GNN formalism provides a general framework for defining neural networks on graph-structured data [40]. With permutation invariance (under the arbitrary labeling of nodes) built in by design, GNNs offer a scheme to generate node representations that incorporate the topology of the graph. The common theme to any type of GNN is that it implements some form of neural *message passing*, whereby messages (in the form of vectors) are exchanged between the nodes of the graph to iteratively update the internal representations of the graph's nodes [24]. More formally, for a given input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ along with any relevant node features $\mathbf{X} \in \mathbb{R}^{d_0 \times n}$, a GNN can be used to generate node embeddings $\mathbf{p}_v, \forall v \in \mathcal{V}$ [40]. This is done iteratively as follows: Consider hidden embedding vectors $\{\mathbf{h}_v^k\}$ representing each node $v \in \mathcal{V}$. In each iteration k, every embedding vector $\mathbf{h}_v^k$ is updated based on information inferred from the corresponding local neighborhood, denoted as $\mathcal{N}_v = \{u \in \mathcal{V} | (u, v) \in \mathcal{E}\}$. At layer (iteration) $k = 0$, the initial representations $\mathbf{h}_v^0 \in \mathbb{R}^{d_0}$ are usually derived from the node's labels or given input features of dimensionality d_0 [41]. This single-layer update can then be formalized as

$$\mathbf{m}_v^k = \text{AGGREGATE}_\theta^k \left(\{\mathbf{h}_u^{k-1} | u \in \mathcal{N}_v\} \right),$$
$$\mathbf{h}_v^k = \text{UPDATE}_\theta^k \left(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k \right), \tag{7}$$

for the GNN layers (iterations) $k = 1, \ldots, K$, with AGGREGATE($\cdot$) and UPDATE($\cdot$) referring to some (typically parametrized) differentiable functions [40]. See Refs. [15,16,22,23] for several popular design choices such as graph convolutional networks (GCNs). In Eq. (7), the vector $\mathbf{m}_v^k$ represents the kth layer *message* for node $v = 1, \ldots, n$ as aggregated from the corresponding local graph neighborhood $\mathcal{N}_v$. At each iteration k, every node aggregates information from its local neighborhood, and as these iterations progress each node embedding encapsulates a larger receptive field within the graph. Specifically, after k iterations every node embedding contains information

about its k-hop neighborhood, with the final output (after K iterations of message passing) defined as $\mathbf{p}_\nu = \mathbf{h}_\nu^K$. This output can then be used for prediction tasks, such as node classification. To optimize the predictive power of this approach, the (parametrized) final node embeddings $\mathbf{p}_\nu = \mathbf{h}_\nu^K(\theta)$ are fed into a problem-specific loss function, with some form of stochastic gradient descent optimizing the weight parameters θ of the network.

IV. THEORETICAL FRAMEWORK

In this section we discuss in detail the theoretical framework underlying our work. We show how to solve the GCP using GNNs, with the antiferromagnetic Potts model providing a canonical choice for the loss function controlling the unsupervised GNN training process.

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{1, 2, \ldots, n\}$ and edge set $\mathcal{E} = \{(i, j) : i, j \in \mathcal{V}\}$. Given such graph, our goal is to assign colors to the nodes of the graph in such a way that adjacent nodes are assigned different colors and the number of colors used is minimal. To this end we associate a discrete variable (spin) $\sigma_\nu = 1, \ldots, q$ with every vertex $\nu \in \mathcal{V}$, thereby assigning one of q possible states (colors) to every node in the graph. To enforce a valid coloring we consider the standard Potts spin model [26] with antiferromagnetic interactions as given in Eq. (1); this model gives no energy contribution to neighboring spins with different colors, but penalizes color clashes with a positive energy offset. The ground-state energy is then zero if and only if the graph is q colorable, thus providing a good cost function for encoding the GCP. To make the GCP compatible with our GNN-based approach we first reformulate the Potts model (1) in terms of one-hot-encoded variables $\hat{\mathbf{y}}_i$ as

$$H_{\text{Potts}} = -J \sum_{(i,j) \in \mathcal{E}} \hat{\mathbf{y}}_i^\mathsf{T} \cdot \hat{\mathbf{y}}_j. \tag{8}$$

Here the variable $\hat{\mathbf{y}}_i$ describes the class assignment for node $i \in \mathcal{V}$ within a q-dimensional unit vector where all components are zero except for one (set to 1) and signals the color assignment as $\sigma_i = \sum_{\alpha=1}^q \alpha \hat{\mathbf{y}}_i^{[\alpha]}$, with $\hat{\mathbf{y}}_i^{[\alpha]}$ denoting the αth component of $\hat{\mathbf{y}}_i$, and by definition $\sum_\alpha \hat{\mathbf{y}}_i^{[\alpha]} = 1$. Next, generalizing our approach as detailed in Ref. [25] to multiclass node classification problems, we apply a relaxation strategy to the problem Hamiltonian H_{Potts} to generate a differentiable loss function $\mathcal{L}(\theta)$ with which we perform unsupervised training on the multicolor node representations of the GNN. To this end we replace the (hard) one-hot-encoded decision vectors $\hat{\mathbf{y}}_i$ with corresponding (soft) normalized assignments $\mathbf{p}_i(\theta) \in [0, 1]^q$, letting $\hat{\mathbf{y}}_i \rightarrow \mathbf{p}_i(\theta)$. In our approach, these soft assignments $\mathbf{p}_i(\theta)$ are generated by our GNN ansatz as final node embeddings $\mathbf{p}_i = \mathbf{h}_i^K \in [0, 1]^q$ at layer K, after the application of a standard softmax activation function, and used as an input for the Potts-like loss function $\mathcal{L}(\theta)$ given by

$$H_{\text{Potts}} \longrightarrow \mathcal{L}_{\text{Potts}}(\theta) = -J \sum_{(i,j) \in \mathcal{E}} \mathbf{p}_i^\mathsf{T} \cdot \mathbf{p}_j. \tag{9}$$

To arrive at the predicted soft assignments $\mathbf{p}_i$ for all nodes $i = 1, \ldots, n$, the GNN follows a standard recursive neighborhood aggregation scheme [24,42], where each node $\nu =$
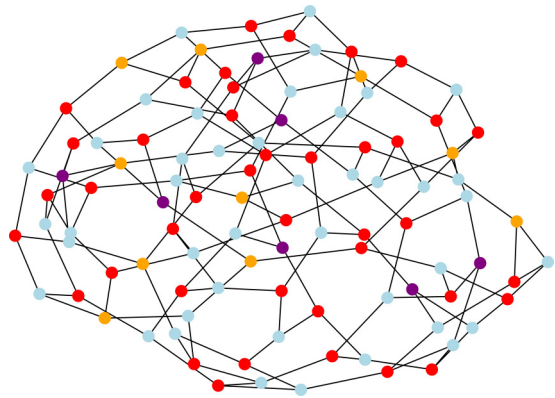

FIG. 2. Example 4-coloring solution to the graph coloring problem for a random 3-regular graph with $n = 100$ nodes. At training completion the GNN provides color (class) assignments to each vertex. The optimization problem is to assign the colors in a way that adjacent nodes must be assigned different colors, while using the smallest number of colors possible (corresponding to the antiferromagnetic ground state of the underlying Potts model).

$1, 2, \ldots, n$ collects information (encoded as feature vectors) of its neighbors to compute its new feature vector $\mathbf{h}_\nu^k$ at layer $k = 0, 1, \ldots, K$. Similar to Ref. [25], the node embeddings $\mathbf{h}_\nu^0$ are initialized randomly. After k iterations of aggregation, a node is represented by its transformed feature vector $\mathbf{h}_\nu^k$, which captures the structural information within the node's k-hop neighborhood [14]. For the multiclass node classification task at hand we use convolutional aggregation steps, followed by the application of a nonlinear softmax activation function with the dimensionality set by the number of colors q, thereby providing one-hot-encoded q-dimensional soft (probabilistic) node assignments $\mathbf{p}_\nu = \mathbf{h}_\nu^K \in [0, 1]^q$, with the softmax function automatically ensuring normalization as $\sum_{\alpha=1}^q \mathbf{p}_\nu^{[\alpha]} = 1$. By virtue of this built-in normalization and in stark contrast to any QUBO-based approach [7,8,43], we do not have to add additional terms to the loss function to enforce a one-hot constraint that drives the solution towards one *unique* color assignment per node. Conversely, once the unsupervised training process has completed, we apply a simple projection heuristic to map the soft assignments $\mathbf{p}_\nu$ to hard class variables $\sigma_\nu = 1, \ldots, q$ using, for example, $\sigma_\nu = \arg\max(\mathbf{p}_\nu)$ to find the class (color) with the largest predicted probability, thus providing unique color assignments for every node. As shown in Fig. 1, the final color assignment $\{\sigma_\nu\}$ can then be visualized as a q coloring of the graph. For further illustration, an example 4-coloring solution (as implemented with this approach) for a random 3-regular graph with $n = 100$ vertices is shown in Fig. 2. Far beyond this sample scale, the scalability inherent to GNNs opens up the possibility of studying unprecedented problem sizes with hundreds of millions of nodes when leveraging distributed training in a mini-batch fashion on a cluster of machines as demonstrated recently in Ref. [44].

Our approach features several hyperparameters, including the number of layers K, the dimensionality of the embedding vectors $\mathbf{h}_i^k$, and the learning rate β, which can be optimized via hyperparameter optimization techniques. In particular, the number of colors q can be seen as a GCP-specific hyperparameter that together with the graph $\mathcal{G}$ defines the
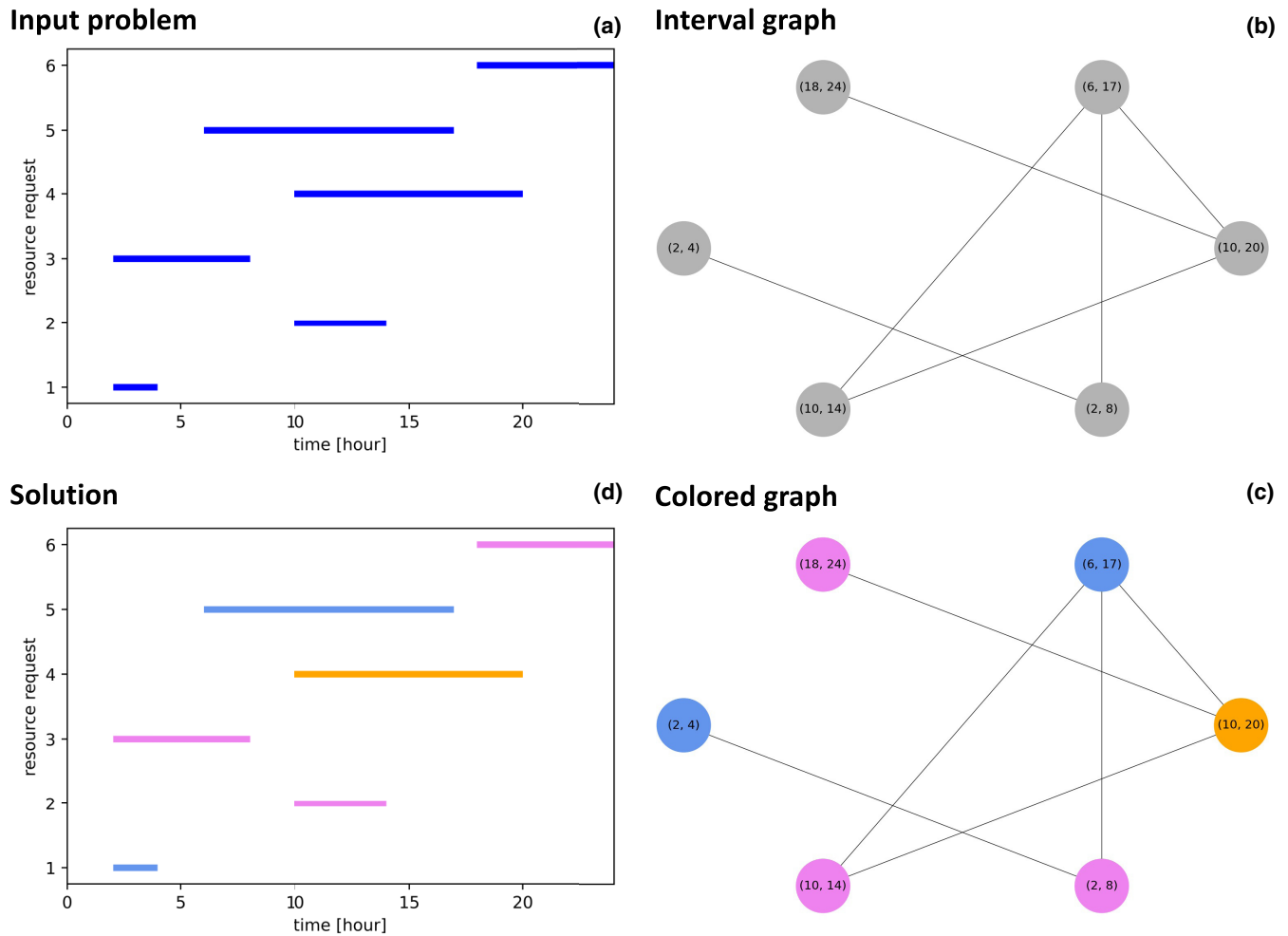
FIG. 3. Example end-to-end application of graph coloring for a task scheduling problem. (a) The problem is specified in terms of a schedule detailing six resource requests (vertical axis) as a function of time, spread out over the course of 24 h (horizontal axis). (b) *Encoding*: The problem is encoded in the form of an *interval graph* where every node represents one request labeled by the corresponding time interval, and edges refer to clashes within the resource requests whenever two requests overlap in time. (c) *Processing*: We solve the graph coloring problem on this interval graph using a graph neural network with a Potts-type loss function as detailed in the main text. Once the algorithm has converged, we obtain a graph colored with the smallest number of color clashes for the given number of colors. In this example we find a feasible coloring with $\chi = 3$ colors as expected based on the clique of size three. (d) *Decoding*: Finally the proposed colors are mapped back to the original resource requests. In this example we find that three resources are sufficient in order to satisfy all requests.

input pair $(\mathcal{G}, q)$ for the decision problem whether or not $\mathcal{G}$ allows for a q coloring. To identify the chromatic number χ, one can perform, for example, a naive search by sequentially checking if $\mathcal{G}$ is q colorable for $q = 1, 2, \ldots$, or use a binary search to cut down the average number of calls required logarithmically. Alternatively, one can try to solve the graph coloring problem (i.e., the search for a feasible coloring) in parallel with the minimization of colors used by adding a corresponding regularization term to the loss function (such as $\sim q^2$). However, such a term should not overpower the regular Potts-like term $\mathcal{L}_{\text{Potts}}(\theta)$, as to not drive the overall solution towards an infeasible coloring.

V. INDUSTRY APPLICATIONS

The graph coloring problem is known to describe many real-world applications, in particular in scheduling and allocation problems [1]. Prominent examples include timetabling

problems or frequency assignment problems, relevant to the planning of wireless communication services [1,45]. To illustrate both our GNN-based approach as well as the real-world applicability of the graph-coloring problem, we now discuss an end-to-end application for a canonical scheduling use case. We do so within a comprehensive three-step encode-process-decode approach in which we (i) first phrase the use case as a graph coloring problem (encoding), (ii) we then solve this problem using our GNN-based approach (processing), and finally (iii) decode the coloring solution to an actual solution for the use case at hand (decoding). For the sake of this illustrative example we consider a small problem instance as illustrated in Fig. 3. More thorough numerical benchmarks are presented in Sec. VI.

We consider a scenario involving the scheduling of tasks with given start and end times, with applications in car sharing, taxi companies, aircraft assignments, etc. Specifically, we face n resource requests (or bookings) with a start time

indicating when the resource will be needed and an end time indicating the resource is available; see Fig. 3(a) for an example problem with $n = 6$ resource requests. The problem is then to assign resources (e.g., cars) to these requests (e.g., bookings) in the most efficient way involving the smallest number of resources needed. As illustrated in Fig. 3(a), typically some requests will overlap in time leading to request clashes that cannot be satisfied by the same resource. As commonly done in resource allocation problems and scheduling theory, this situation can conveniently be described with the help of an undirected *interval graph* in which a vertex is introduced for every request, with edges connecting vertices whose requests overlap. Figure 3(b) displays an encoding of the problem with a graph made of six vertices and six edges, including a clique of size three. While inexpensive, special-purpose algorithms exist for interval graphs [1], we can then solve the graph coloring problem on this interval graph (in the same way as any other GCP) using our general-purpose GNN-based approach. To this end, we run unsupervised multiclass classification directly on the interval graph with $n = 6$ nodes and final softmax nonlinearity of dimension $q = 3$, as opposed to QUBO-based approaches involving $q \times n = 18$ binary variables [4]. For the sample problem illustrated in Fig. 3(c) we obtain a feasible coloring using just three colors ($\chi = 3$). Finally, as shown in Fig. 3(d), we decode this coloring to the corresponding assignment in which three resources are used to satisfy all six requests.

VI. NUMERICAL EXPERIMENTS

We now turn to systematic numerical experiments using standard benchmark problems for graph coloring. In particular, we provide results for the publicly available COLOR data set [46], as well as well-known citation data sets (Cora [47], Citeseer [48], and Pubmed [49]) often used for graph-based benchmark experiments. The former provide small and medium-sized dense problem instances with relatively large known chromatic numbers ($\chi \sim 10$), while the latter are large, but sparse real-world graphs (which, for the purpose of graph coloring, we consider as undirected graphs, dismissing any potential node or edge features). Our basic GNN architecture is very similar to the one detailed in Ref. [25], except for the dimension of the final GNN layer set here to the number of colors q. We provide results for two standard types of GNN architectures, that is graph convolutional networks (GCN) [15] as well as GraphSAGE [13]. Model configurations (hyperparameters) are detailed in the Appendix. We compare our results (as given by the cost directly reported by the Potts Hamiltonian) to previously published results sourced from Ref. [33], including results based on the tabu-search based heuristic called Tabucol [50], a local search algorithm which tracks single moves within a tabu list. We complement these with our own benchmark results obtained with Tabucol and a *greedy* coloring algorithm. The latter parses through the graph's vertices one by one according to some vertex ordering and greedily assigns the first available color. If no available color can provide a feasible coloring, yet another color is expensed, thus (by design) always providing a feasible coloring with a corresponding upper bound on the chromatic number denoted as $\bar{\chi}_{\text{greedy}}$. Here we have implemented a greedy algorithm with largest-first ordering strategy as further detailed in Refs. [1,51]. Our greedy results for $\bar{\chi}_{\text{greedy}}$ largely agree with results presented in Ref. [52].

For a given graph and a fixed number of colors q, we report the total number of color clashes as achieved with our physics-inspired GNN solvers (dubbed PI-GCN and PI-SAGE, respectively), and we assess the solution quality with the normalized error $\epsilon = H_{\text{Potts}}/|\mathcal{E}|$ quantifying the number of color clashes normalized by the number of edges $|\mathcal{E}|$. Accordingly, the quantity $\Xi = 1 - \epsilon$ can be regarded as the coloring accuracy achieved (that is the number of edges without coloring conflicts divided by the total number of edges). In addition, we report upper bounds on the chromatic number, denoted as $\bar{\chi}_{\text{GNN}}$. To this end we have implemented a simple search as well as randomized post-processing heuristic. For a given GNN solution, the latter tries to remove remaining color clashes at the expense of one additional color, by randomly going through existing clashes, and randomly assigning the new color to one of the two nodes at hand. This process is repeated till a feasible coloring (with zero cost) has been found. Our method has been implemented in python, leveraging the open-source libraries Deep Graph Library [53] and PyTorch (for GNN handling), and NetworkX (for graph handling). All reported experiments have been run on p3.2×large AWS EC2 instances, with 1 GPU, 8 virtual CPUs, 81 GiB memory, 16 GiB GPU memory, with 2.3 GHz (base) and 2.7 GHz (turbo) Intel Xeon E5-2686 v4 processors.

COLOR graphs. We study several benchmark instances from the COLOR data set [46] which can be categorized as follows [46]:

(i) Book graphs: For a given work of literature, a graph is created with each node representing a character. Two nodes are connected by an edge if the corresponding characters encounter each other in the book. This type of graph is publicly available for Tolstoy's Anna Karenina (anna), and Hugo's Les Misérables (jean), among others.

(ii) Myciel graphs: This family of graphs is based on the Mycielski transformation. The Myciel graphs are known to be difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size [46].

(iii) Queens graphs: This family of graphs is constructed as follows. Given an n by n chessboard, a queens graph is a graph made of n^2 nodes, each corresponding to a square of the board. Two nodes are then connected by an edge if the corresponding squares are in the same row, column, or diagonal. In other words, two nodes are adjacent if and only if queens placed on these two nodes can attack each other in a single move. In all cases, the maximum clique in the graph is no more than n, and the coloring value is lower bounded by n.

Our numerical results are summarized in Table I, and example solutions are displayed in Fig. 4. We consistently find sub-one-percent normalized errors (i.e., $\epsilon < 1\%$) across all COLOR instances, some of which have been deemed as hard [54,55], with the GraphSAGE-based architecture typically outperforming the GCN-based baseline architecture. This observation appears to be in agreement with existing literature [41] showing that GCN architectures tend to be more susceptible to oversquashing (bottleneck) effects than other GNN architectures. In this work, with its inherent neighborhood sampling strategy, GraphSAGE is seen be more robust

TABLE I. Numerical results for COLOR graphs [46]. For a given number of colors q, we report the cost $= H_{\text{Potts}}$, that is the number of conflicts in the best coloring result, as achieved with our physics-inspired GNN solvers (PI-GCN and PI-SAGE), together with results for the Tabucol algorithm, as partially sourced from Ref. [33]. Upper bounds on the chromatic number χ as found by a greedy algorithm as well as PI-SAGE are reported as $\bar{\chi}_{\text{greedy}}$ and $\bar{\chi}_{\text{GNN}}$, respectively. Best results are marked in boldface. The last column gives the normalized error ϵ (for the best PI-GNN result) specifying the relative fraction of edges with color clashes. Example solutions are displayed in Fig. 4. Further details are provided in the main text.

Graph	Nodes	Edges	Density	Colors q	$\bar{\chi}_{\text{greedy}}$	$\bar{\chi}_{\text{GNN}}$	Tabucol	Tabucol [33]	GNN [33]	PI-GCN	PI-SAGE	Error ϵ
anna	138	493	5.22%	11	11	11	**0**	**0**	1	1	**0**	0.00%
jean	80	254	8.04%	10	10	10	**0**	**0**	**0**	**0**	**0**	0.00%
myciel5	47	236	21.83%	6	6	6	**0**	**0**	**0**	**0**	**0**	0.00%
myciel6	95	755	16.91%	7	7	7	**0**	**0**	**0**	**0**	**0**	0.00%
queen5-5	25	160	53.33%	5	5	5	**0**	**0**	**0**	**0**	**0**	0.00%
queen6-6	36	290	46.03%	7	8	7	**0**	**0**	4	1	**0**	0.00%
queen7-7	49	476	40.48%	7	9	7	**0**	10	15	8	**0**	0.00%
queen8-8	64	728	36.11%	9	10	10	**0**	8	7	6	1	0.14%
queen9-9	81	1056	32.59%	10	12	11	**0**	5	13	13	1	0.09%
queen8-12	96	1368	30.00%	12	13	12	**0**	10	7	10	**0**	0.00%
queen11-11	121	1980	27.27%	11	15	14	20	33	33	37	**17**	0.86%
queen13-13	169	3328	23.44%	13	17	17	35	42	40	61	**26**	0.78%

to potential oversquashing effects as relevant for the larger and dense COLOR instances. This increased performance of PI-SAGE comes at a price of extended training times, with per-epoch training times being $\sim$5–50$\times$ longer than for PI-GCN on the same graph. Whereas the PI-GCN model takes anywhere from $\sim$0.167 to 2 h to train, the PI-SAGE model takes anywhere from $\sim$1 to 8 h to train for the COLOR graphs considered. With potentially multiple factors contributing to this disparity, a more detailed analysis of this observation is left for future research. Finally, we find that PI-SAGE performs on par with Tabucol across the COLOR instances. In addition, the estimated chromatic numbers found with PI-SAGE are on par or better than the greedy baseline results. For example, for queen7-7 we find $\bar{\chi}_{\text{GNN}} = 7$, while $\bar{\chi}_{\text{greedy}} = 9$. For solutions with nonzero cost we find that a simple post-processing heuristic can provide a fully purified (feasible) solution at the expense of a small number of colors, thereby providing a simple estimate for the chromatic number. For

example, solutions with just one remaining color clash, as is the case for queen8-8 and queen9-9, are trivial to purify at the expense of one color (yielding estimates of $\bar{\chi}_{\text{GNN}} = q + 1$). For a larger number of color clashes, as is the case for queen11-11 and queen13-13, several iterations of this simple post-processing routine may be necessary till a feasible coloring is found. While further improvements may be possible through additional GNN runs at colors $q + 1, q + 2, \ldots$, we observe on-par or better performance compared to the greedy
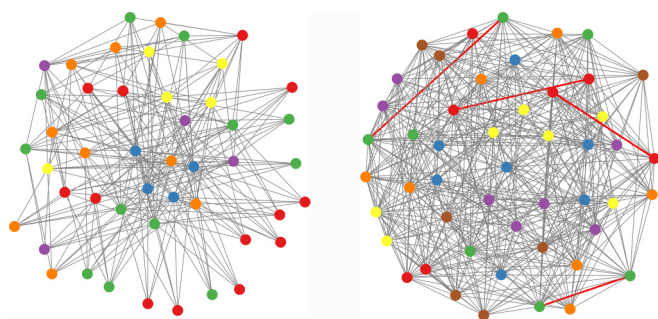

FIG. 4. Example solutions to the graph coloring problem for the myciel5 (left) and queen7-7 (right) graphs from the COLOR data set with $q = 6$ and $q = 7$ colors, respectively. The solution for myciel5 corresponds to a feasible coloring with normalized error $\epsilon = 0\%$, whereas the solution for queen7-7 represents an infeasible, but low-energy solution with $\epsilon = 0.84\%$ for which the remaining four color clashes have been highlighted with bold (conflicting) edges. Further details are provided in Table I and the main text.
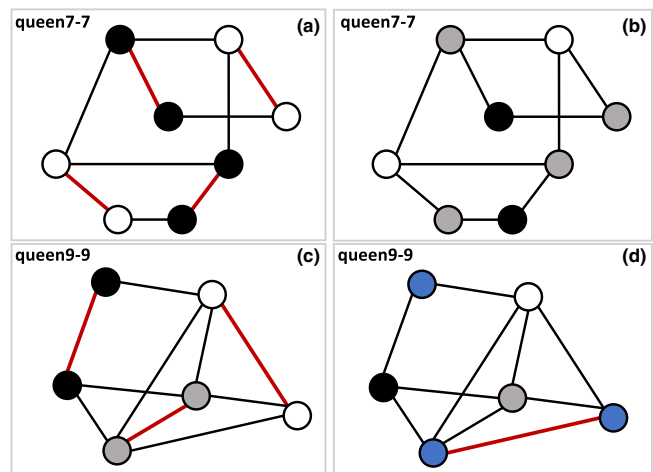

FIG. 5. Illustration of our simple post-processing heuristic to estimate upper bounds on the chromatic number. Color clashes have been highlighted with bold (conflicting) edges. Panels on the left display example original (infeasible) solutions found by PI-GNN (not corresponding to the best solutions found with PI-GNN). Panels on the right show the results of a simple post-processing strategy. For the queen7-7 example this randomized post-processing heuristic can remove all color clashes and provide a feasible coloring, at the expense of one additional color. For the queen9-9 instance, one color clash is still outstanding after application of one iteration of this purification protocol. Further details are provided in the main text.

TABLE II. Numerical results for citation graphs [47–49]. Further details are provided in the caption of Table I and in the main text.

Graph	Nodes	Edges	Density	Colors q	$\bar{\chi}_{greedy}$	$\bar{\chi}_{GNN}$	Tabucol	Tabucol [33]	GNN [33]	PI-GCN	PI-SAGE	Error ϵ
Cora	2708	5429	0.15%	5	5	5	**0**	31	3	1	**0**	0.00%
Citeseer	3327	4732	0.09%	6	6	6	**0**	6	3	1	**0**	0.00%
Pubmed	19717	44338	0.02%	8	8	9	NA	NA	35	**13**	17	0.03%

baseline already with this simple post-processing only. The core of this post-processing routine is illustrated in Fig. 5.

Citation graphs. Next we provide results for publicly available, real-world citation graphs, with up to $n \sim 2 \times 10^4$ nodes. While Cora and Citeseer refer to networks of computer science publications (with nodes representing publications and edges referring to citations), the Pubmed citation network is a set of articles related to diabetes from the PubMed database [47–49]. Following Ref. [33], the number of available colors q for the Cora, Citeseer, and Pubmed graphs has been set to 5, 6 and 8, respectively. The results of our analysis are displayed in Table II, with the greedy coloring algorithm providing optimal baseline results for these *sparse* instances [with graph densities in the range $\sim$(0.02–0.15)%]. We find that the basic PI-GCN solver displays consistent, small errors $\epsilon \sim 10^{-4}$, close to the global optimum. For the Cora and Citeseer graphs PI-GCN finds solutions with just one single color clash within $\sim 5 \times 10^3$ edges, while PI-SAGE finds optimal solutions at zero cost. Note that local optimality has been verified for these solutions through a series of simple local spin flips. Similarly, even for the largest instance (Pubmed with $n \sim 2 \times 10^4$ nodes) we obtain a small error of $\epsilon \sim 3 \times 10^{-4}$, while Tabucol fails to color the graph within a 24 h time limit [33]. Conversely, we find that PI-GCN converges for the Cora and Citeseer instances in $\sim$5 to 40 min, respectively, while Pubmed takes $\sim$6.7 h for training completion. The comparatively long training time for Pubmed is arguably due to separate logic being used to calculate the loss function: Because of memory constraints on the training instances, here we implemented sparse tensor calculations to ensure we would avoid memory overload, at the expense of training time. A more thorough analysis together with the investigation of

warm-starting (transfer learning) strategies is left for future research.

Overall, we find that the general-purpose PI-GNN solver shows the potential to provide competitive coloring results compared to established, state-of-the-art heuristics such as the Tabucol algorithm [50] or greedy coloring algorithms; in particular for large graphs (where Tabucol runtimes become extensively long) or dense graphs (where greedy algorithms may show performance drops). However, with the possibility to scale to problems with millions of nodes [25], as well as the ability to solve other multiclass problems such as community detection or data clustering within the very same framework.

VII. CONCLUSION AND OUTLOOK

In summary, we have shown how graph neural networks can be used to solve graph coloring problems using insights from statistical physics as our guiding principle. In our approach we frame graph coloring as a multiclass (multicolor) node classification problem, with the Potts model providing a canonical choice for the loss function with which we train the GNN. Natively extending our framework as presented in Ref. [25], we apply a relaxation strategy to the Potts model by dropping integrality constraints on the decision variables in order to generate a differentiable loss function with which we perform unsupervised training on the node representations of the GNN. The GNN is then trained to generate soft assignments to predict the likelihood of belonging in one of q classes, for each vertex in the graph. In post training we use simple projection heuristics to find a coloring solution consistent with the original problem.

TABLE III. Hyperparameters for results on COLOR graphs. All hyperparameters refer to our PI-SAGE architecture. All runs used the AdamW optimizer with default values (other than learning rate, as per the table).

Graph	Colors q	Embedding d_0	Layers K	Hidden dims $[d_1, d_2,...]$	Learning rate β	Dropout	Number epochs	Patience
anna	11	43	1	[22]	0.03507	0.3298	100 000	500
jean	10	50	1	[62]	0.01663	0.3185	100 000	500
myciel5	6	16	1	[18]	0.01333	0.3964	100 000	500
myciel6	7	8	1	[22]	0.01779	0.2225	100 000	500
queen5-5	5	77	1	[32]	0.02988	0.3784	100 000	500
queen6-6	7	20	1	[12]	0.05105	0.3425	100 000	500
queen7-7	7	67	1	[12]	0.02175	0.2339	100 000	500
queen8-8	9	32	1	[10]	0.02728	0.2878	100 000	500
queen8-12	12	107	1	[23]	0.01730	0.1796	100 000	500
queen9-9	10	109	1	[16]	0.02636	0.3257	100 000	500
queen11-11	11	75	1	[25]	0.04600	0.2974	100 000	500
queen13-13	13	112	1	[199]	0.14426	0.1571	100 000	500

TABLE IV. Hyperparameters for results on citations graphs. All hyperparameters refer to our PI-GCN architecture. All runs used the Adam optimizer with default values (other than learning rate, as per the table).

Graph	Colors q	Embedding d_0	Layers K	Hidden dims $[d_1, d_2,...]$	Learning rate β	Dropout	Number epochs	Patience
Cora	5	2342	1	[3496]	0.00556	0.0148	100 000	500
Citeseer	6	5127	1	[2472]	0.00983	0.0161	100 000	500
Pubmed	8	5137	1	[6082]	0.02966	0.1715	100 000	500

Finally, we highlight possible extensions of research going beyond our present work. First, using the unifying framework established by the Potts model, it would be interesting to apply our approach to other multiclass problems such as community detection, data clustering, and the minimum clique cover problem. Beyond that, one could apply our approach to other large-spin (nonbinary) problems such as, for example, variations of the Blume-Capel model which can be seen as a generalization of QUBO to large-spin (i.e., multicolor) variables. Furthermore, one could systematically study the potential existence of sharp coloring thresholds and the onset of hardness with associated critical graph connectivities, for both families of random graphs as done in Ref. [37] but also more structured real-world problems. Finally, there are several ways to potentially boost the performance of our GNN-based optimizer. For example, one could explore alternative GNN implementations, potentially in combination with graph rewiring techniques, as recently proposed and analyzed in Ref. [56], thereby decoupling the training graph from the original problem graph and providing additional GNN design choices. In addition, post training one could replace the simple deterministic (argmax) projection scheme used here with more sophisticated strategies such as local search routines that further refine the mapping from soft (probabilistic) class assignments to hard (integer) variables.

An end-to-end open source demo version of the code implementing our approach has been made publicly available at [57].

APPENDIX: HYPERPARAMETERS FOR PI-GNN ON BENCHMARK INSTANCES

In this Appendix we provide details for the specific model configurations (hyperparameters) as used to solve the COLOR and citation graph instances with our physics-inspired GNN solver (PI-GNN). The results achieved with these model configurations are displayed in Tables I and II; the corresponding hyperparameters are given in Tables III and IV, respectively.

[1] R. M. R. Lewis, *A Guide to Graph Colouring - Algorithms and Applications* (Springer, Heidelberg, 2016).

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[3] N. Barnier and P. Brisset, Graph coloring for air traffic flow management, Ann. Operations Res. **130**, 163 (2004).

[4] Y.-H. Oh, H. Mohammadbagherpoor, P. Dreher, A. Singh, X. Yu, and A. J. Rindos, Solving multi-coloring combinatorial optimization problems using hybrid quantum algorithms, arXiv:1911.00595.

[5] O. Titiloye and A. Crispin, Quantum annealing of the graph coloring problem, Discrete Optimization **8**, 376 (2011).

[6] B. Pokharel, Z. Gonzalez Izquierdo, P. Aaron Lott, E. Strbac, K. Osiewalski, E. Papathanasiou, A. Kondratyev, D. Venturelli, and E. Rieffel, Inter-generational comparison of quantum annealers in solving hard scheduling problems, arXiv:2112.00727.

[7] F. Glover, G. Kochenberger, and Y. Du, Quantum bridge analytics I: A tutorial on formulating and using QUBO models, 4OR **17**, 335 (2019).

[8] A. Lucas, Ising formulations of many NP problems, Front. Phys. **2**, 5 (2014).

[9] M. Gori, G. Monfardini, and F. Scarselli, in *2005 IEEE International Joint Conference on Neural Networks* (IEEE, Palm Springs CA, 2005), Vol. 2, p. 729.

[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, The graph neural network model, IEEE Trans. Neural Networks **20**, 61 (2008).

[11] A. Micheli, Neural network for graphs: A contextual constructive approach, IEEE Trans. Neural Networks **20**, 498 (2009).

[12] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2015), p. 2224.

[13] W. Hamilton, Z. Ying, and J. Leskovec, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2017), p. 1024.'

[14] K. Xu, H. Weihua, J. Leskovec, and S. Jegelka, in *International Conference on Learning Representations* (OpenReview.net, 2019).

[15] T. N. Kipf and M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, in *ICLR* (OpenReview.net, 2017), https://openreview.net/forum?id=SJU4ayYgl.

[16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, A comprehensive survey on graph neural networks, arXiv:1901.00596.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena, in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, 2014), p. 701.

[18] Z. Sun, Z. H. Deng, J.-Y. Nie, and J. Tang, in *International Conference on Learning Representations* (OpenReview.net, 2018).

[19] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, arXiv:1806.01973.

[20] A. Strokach, D. Becerra, C. Corbi-Verge, A. Perez-Riba, and P. M. Kim, Fast and flexible protein design using deep graph neural networks, Cell Syst. **11**, 402 (2020).

[21] T. Gaudelet, B. Day *et al.*, Utilising graph machine learning within drug discovery and development, arXiv:2012.05716.

[22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, in *International Conference on Learning Representations* (IEEE, New York, 2016).

[23] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, in *International Conference on Learning Representations* (OpenReview.net, 2018).

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, in *Proceedings of the 34th International Conference on Machine Learning-Volume* (JMLR, 2017), Vol. 70, p. 1263.

[25] M. J. A. Schuetz, J. K. Brubaker, and H. K. Katzgraber, Combinatorial optimization with physics-inspired graph neural networks, Nat. Mach. Intell. **4**, 367 (2022).

[26] F. Y. Wu, The Potts model, Rev. Mod. Phys. **54**, 235 (1982).

[27] M. Blatt, S. Wiseman, and E. Domany, Superparamagnetic Clustering of Data, Phys. Rev. Lett. **76**, 3251 (1996).

[28] S. Fortunato, Community detection in graphs, Phys. Rep. **486**, 75 (2010).

[29] S. Fortunato and D. Hric, Community detection in networks: A user guide, Phys. Rep. **659**, 1 (2016).

[30] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E **69**, 026113 (2004).

[31] H. Lemos, M. Prates, P. Avelar, and L. Lamb, Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems, arXiv:1903.04598.

[32] N. Karalias and A. Loukas, Erdoes goes neural: An unsupervised learning framework for combinatorial optimization on graphs, arXiv:2006.10643.

[33] W. Li, R. Li, Y. Ma, S. On Chan, and B. Yu, Rethinking Graph Neural Networks for Graph Coloring (ICLR 2021 Conference Withdrawn Submission).

[34] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, End-to-end constrained optimization learning: A survey, arXiv:2103.16378.

[35] Q. Cappart, D. Chetelat, E. Khalil, A. Lodi, C. Morris, and P. Velickovic, Combinatorial optimization and reasoning with graph neural networks, arXiv:2102.09544.

[36] R. M. Karp, *Complexity of Computer Computations* (Plenum, New York, 1972), Chap. Reducibility among Combinatorial Problems, p. 85.

[37] L. Zdeborová and F. Krzakala, Phase transitions in the coloring of random graphs, Phys. Rev. E **76**, 031131 (2007).

[38] J. M. Yeomans, *Statistical Mechanics of Phase Transitions* (Oxford University Press, Oxford, 1992).

[39] F. Y. Wu, Potts model and graph theory, J. Stat. Phys. **52**, 99 (1988).

[40] W. L. Hamilton, Graph representation learning, Synthesis Lectures on Artificial Intelligence and Machine Learning **14**, 1 (2020).

[41] U. Alon and E. Yahav, in *ICLR* (OpenReview.net, 2021).

[42] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, in *International Conference on Machine Learning (ICML)* (PMLR, 2018), p. 5453.

[43] G. Kochenberger, F. Glover, B. Alidaee, and C. Rego, An unconstrained quadratic binary programming approach to the vertex coloring problem, Ann. Oper. Res. **139**, 229 (2005).

[44] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, DistDGL: Distributed graph neural network training for billion-scale graphs, arXiv:2010.05337.

[45] A. Eisenblaetter, M. Groetschel, and A. M. C. A. Koster, Frequency planning and ramifications of coloring, Discuss. Math. Graph Theory **22**, 51 (2002).

[46] M. Trick, COLOR Dataset, URL (2002), https://mat.tepper.cmu.edu/COLOR02/.

[47] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, Automating the construction of internet portals with machine learning, Inform. Retrieval **3**, 127 (2000).

[48] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, Collective classification in network data, AI Mag. **29**, 93 (2008).

[49] G. Namata, B. London, L. Getoor, and B. Huang, in *10th International Workshop on Mining and Learning with Graphs* (2012), Vol. 8, p. 249.

[50] A. Hertz and D. de Werra, Using tabu search techniques for graph coloring, Computing **39**, 345 (1987).

[51] A. Kosowski and K. Manuszewski, Graph Colorings (American Mathematical Society, 2004), Chap. Classical Coloring of Graphs, pp. 2–19.

[52] M. Aslan and N. A. Baykan, A performance comparison of graph coloring algorithms, Int. J. Intell. Syst. Appl. Eng. **4**, 1 (2016).

[53] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai *et al.*, Deep graph library: A graph-centric, highly-performant package for graph neural networks, arXiv:1909.01315.

[54] S. Gualandi and F. Malucelli, Exact solution of graph coloring problems via constraint programming and column generation, INFORMS J. Comput. **24**, 81 (2012).

[55] D. Kirovski and M. Potkonjak, in *Design Automation Conference* (Association for Computing Machinery, 1998), p. 427.

[56] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, Understanding over-squashing and bottlenecks on graphs via curvature, arXiv:2111.14522.

[57] https://github.com/amazon-research/gcp-with-gnns-example.