Learning Combinatorial Node Labeling Algorithms

Lukas Gianinazzi[1,2]* , Maximilian Fries[1]*, Nikoli Dryden[1],
Tal Ben-Nun[1], Maciej Besta[1], Torsten Hoefler[1]
[1]Department of Computer Science
ETH Zurich

ABSTRACT

We present a novel neural architecture to solve graph optimization problems where the solution consists of arbitrary node labels, allowing us to solve hard problems like graph coloring. We train our model using reinforcement learning, specifically policy gradients, which gives us both a greedy and a probabilistic policy. Our architecture builds on a graph attention network and uses several inductive biases to improve solution quality. Our learned deterministic heuristics for graph coloring give better solutions than classical degree-based greedy heuristics and only take seconds to apply to graphs with tens of thousands of vertices. Moreover, our probabilistic policies outperform all greedy state-of-the-art coloring baselines and a machine learning baseline. Finally, we show that our approach also generalizes to other problems by evaluating it on minimum vertex cover and outperforming two greedy heuristics.

1 INTRODUCTION

Combinatorial optimization problems on graphs, such as graph coloring and minimum vertex cover, are the subject of numerous works in academia and industry. Graph coloring (GC) has many real-world applications, including scheduling problems [42, 44], register allocation [12, 48], and mobile network autoconfiguration [5]. Minimum vertex cover (MVC) is a well-studied problem in algorithmic graph theory [8, 21, 45] with applications including text summarization [52] and computational biology [2]. For these problems, the common goal is to *assign labels to nodes subject to combinatorial feasibility constraints and costs.* Therefore, we can generalize these problems into a single problem we call *combinatorial node labeling*, which includes many other problems such as traveling salesman [15, 20], maximum cut [32], and list coloring [28].

As combinatorial node labeling is NP-hard [20, 32], many machine learning approaches have been proposed to solve special cases. A learning scheme faces two fundamental challenges when compared to classical heuristics: (1) it should match or outperform heuristics in quality of solution and performance; and (2) it should generalize across graphs of different sizes. Recent work addressed these challenges for the traveling salesman problem [7, 11, 17, 39], influence maximization [41], and minimum vertex cover [14, 38]. For graph coloring, Lemos et al. [37] introduce an estimator for the chromatic number $\chi(G)$ of a graph, but do not construct a coloring and can over- or underestimate $\chi(G)$ (i.e., there may not be a feasible solution with the estimated number of colors).

Existing machine learning methods do not easily generalize to cases where the number of labels is not known in advance, such as graph coloring. To address this limitation, we use policy gradient
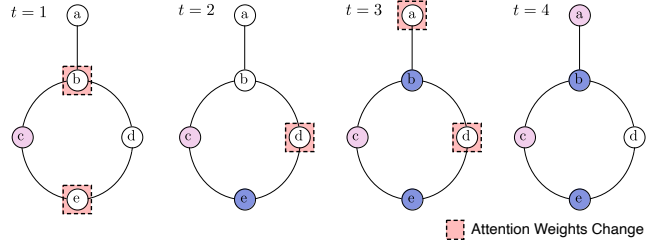

Figure 1: *Spatial locality of the decoding.* **After labeling a node, only its neighbors' attention weights change. The example shows how a graph is** 2**-colored using the vertex order** c, e, b, a, d. **The nodes whose attention weights change have a box around them. For example, when the first node** c **is colored, only its neighbors** b **and** e **receive new attention weights. The figure omits the last step (where** d **is colored).**

reinforcement learning [35, 49] to learn a node ordering and combine this with a simple label rule to label each node according to the ordering. We show that for the chosen label rules, there still exists an order that guarantees an optimal solution. Our model uses a graph neural network (GNN) encoder [54] and an attention-based [36] decoder to assign weights for which node to label next. These weights can either be used greedily or interpreted as probabilities. Our decoder incorporates a *temporal locality* inductive bias, where the selection of a node is conditioned only on the previously labeled node and a global graph context. Further, we also introduce a *spatial locality* inductive bias, whereby labeling a node only impacts the weights of its neighbors. See Figure 1 for an illustration.

We present the combinatorial node labeling framework and evaluate it on the graph coloring and minimum vertex cut problems. We introduce a generic GNN architecture that demonstrates significantly improved results for neural graph coloring and outperforms two state-of-the-art greedy heuristics for minimum vertex cover. A qualitative analysis of the learned heuristics reveals their capability to adapt depending on the properties of the test graph. Our ablation studies show that the introduced temporal and spatial biases improve test scores.

1.1 Related Work

Supervised learning Recent approaches like Joshi et al. [29] and Manchanda et al. [41] obtain good results for influence maximization (IM) and the traveling salesman problem (TSP), respectively. Both approaches use supervised learning. Supervised learning is more sample efficient than reinforcement learning and can lead to overall better results. The fundamental downside is that it is not applicable to every problem. First, it can be difficult to formulate a problem in a supervised manner, since it might have many optimal

solutions (e.g., GC). Second, even if the problem admits a direct supervised formulation, we still need labeled data for training, which can be hard to generate and relies on an existing solver. For IM, the approach of Manchanda et al. [41] shows promising results on graphs much larger than those seen in training. For TSP, the approach of Joshi et al. [29] is very efficient but does not generalize well to graphs larger than those seen in training. Li et al. [38] also use supervised learning and produce good results on minimum vertex cover (MVC), maximum independent set, and maximal clique.

Reinforcement learning Dai et al. [14] provide a general framework for learning problems like MVC and TSP that is trained with reinforcement learning. It shows good results across different graph sizes for the covered problems, but is not fast enough to replace existing approaches. Kool et al. [35] focus on routing problems like TSP and the vehicle routing problem. They outperform Dai et al. on TSP instances of the training size. Unfortunately, their approach does not seem to generalize to graph sizes that are very different from those used for training. Several other reinforcement learning approaches have been proposed and evaluated for TSP [7, 11, 17, 39]. Barrett et al. [6] consider the maximum cut (MaxCut) problem. Huang et al. [26] present a Monte Carlo search tree approach specialized only for graph coloring.

These methods do not address the general node labeling framework, but instead model the solution as a permutation of vertices (e.g., TSP, vehicle routing) or a set of nodes or edges (e.g., MVC, MaxCut).

2 COMBINATORIAL NODE LABELING

We introduce *combinatorial node labeling*, which generalizes many important hard graph optimization problems, including graph coloring (see Appendix D for a list).

We consider an undirected, unweighted, and simple graph $G = (V, E)$ with n nodes in V and m edges in E. We denote the neighbors of a node v by $N(v)$. We assume w.l.o.g. that the graph is connected and hence $m = \Omega(n)$.

A node labeling is a mapping $c : V \rightarrow \{0, \ldots, n\}$. A *partial node labeling* is a mapping $c' : V' \rightarrow \{0, \ldots, n\}$ for any subset of nodes $V' \subseteq V$. A node labeling problem is subject to a feasibility condition and a real-valued *cost function* f. The cost function maps a node labeling c to a real-valued cost $f(c)$. We require that the feasibility condition is expressed in terms of an efficient (polynomial-time computable) *extensibility test* $T : \mathcal{P}(V \times \{0, \ldots, n\}) \times V \times \{0, \ldots, n\} \rightarrow \{0, 1\}$, where $\mathcal{P}$ denotes the powerset. We say the extensibility test passes when it returns 1.

Intuitively, given a partial node labeling $c' : V' \rightarrow \{0, \ldots, n\}$, a node $v \notin V'$, and label ℓ, the extensibility test passes if and only if the partial node labeling c' can be extended by labeling node v with ℓ such that the partial node labeling can be extended into a node labeling. Formally, the extensibility test characterizes the set of *feasible solutions*:

Definition 2.1. A node labeling c is feasible if and only if there exists a sequence of node-label pairs $(v_1, \ell_1), \ldots, (v_n, \ell_n)$ such that for all $i \geq 0$ the extensibility test T satisfies

$$T(\{(v_1, \ell_1), \ldots, (v_i, \ell_i)\}, v_{i+1}, \ell_{i+1}) = 1 \ .$$

The goal of the node labeling problem is to *minimize* the value of the cost function among the feasible node labelings. For consistency, an infeasible node labeling has infinite cost.

Next, we present the two node labeling problems on which we focus in our evaluation.

2.1 Graph coloring (GC)

Definition 2.2. A k-coloring of a graph $G = (V, E)$ is a node labeling $c : V \rightarrow \{1, 2, \ldots, k\}$ such that no two neighbors have the same label, i.e., $\forall \{u, v\} \in E : c(u) \neq c(v)$.

The cost function for GC is the number of distinct labels (or colors) k. Given a partial node labeling $c' : V' \rightarrow \{1, \ldots, k\}$ and any vertex-label pair (v, ℓ), the extensibility test passes for (c', v, ℓ) if and only if the extended partial node labeling $c' \cup (v, \ell)$ is a k- or $(k + 1)$-coloring of the induced subgraph $G[V' \cup \{v\}]$. In particular, the test does not pass when $\ell > k + 1$. The smallest k for which there is a k-coloring of G is the *chromatic number* $\chi(G)$ of G.

2.2 Minimum vertex cover (MVC)

Definition 2.3. A vertex cover of a graph $G = (V, E)$ is a node labeling $c : V \rightarrow \{0, 1\}$ such that every edge is incident to at least one node with label 1, i.e., $\forall \{u, v\} \in E : c(u) = 1 \lor c(v) = 1$.

The cost function for MVC is the number of nodes with label 1. Given a partial node labeling $c' : V' \rightarrow \{0, 1\}$ the extensibility test passes for (c', v, ℓ) if and only if the extended partial node labeling $c' \cup (v, \ell)$ is a vertex cover of the induced subgraph $G[V' \cup \{v\}]$.

3 NODE LABELING POLICIES

Next, we show how every node labeling problem can be formulated as a (finite) Markov decision process (MDP), during which nodes are successively added to a partial node labeling until a termination criterion is met. Then, we discuss how to learn a parameterized policy for this problem. In Section 4, we will present a GNN architecture for parameterizing a policy for such MDPs.

3.1 Node labeling as a Markov Decision Process

We embed the cost function f and the extensibility test into the MDP. Note that we do not require a way to measure the cost of partial node labelings. We formulate the state space, action space, transition function, and reward:

State space A state S represents a partial node labeling. It is a set of pairs $S = V' \times \mathcal{L}$ for a subset of nodes $V' \subseteq V$ and a subset of labels $\mathcal{L} \subseteq \{0, \ldots n\}$. A state is terminal if $V' = V$. Hence, the set of states is the powerset $\mathcal{P}(V \times \{0, \ldots, n\})$ of the Cartesian product of the vertices and labels.

Action space In state S, the set of legal actions are the pairs (v, ℓ) for nodes v and labels ℓ which pass the extensibility test of the problem for the partial node labeling given by S (i.e., $T(S, v, l) = 1$).

Transition function In our case, the transition function $\mathcal{T}$ is deterministic. That is, given the current state S_t and an action (v, ℓ), $\mathcal{T}(S_t, (v, \ell))$ yields the next state $S_{t+1} = S_t \cup \{(v, \ell)\}$.

Reward For a terminal state S representing the node labelling c, the reward is $-f(c)$. For all other states, the reward is 0.

Note that since our tasks are episodic, the return equals the sum of the rewards (specifically the reward received in the terminal state). In particular, we do not use discounting.

In Appendix C.1, we prove that the terminal states of this MDP correspond to the feasible solutions:

Lemma 3.1. *For any node labeling problem, there is an MDP whose terminal states correspond to the feasible solutions with a cost equal to the negative return.*

In the vast majority of reinforcement learning approaches to solve combinatorial graph optimization problems [14, 17, 35, 39], a state S corresponds to a set or sequence of nodes that are already added to a solution set. Instead, in our setting the state represents a partial node labeling. This means that in addition to problems like MVC and TSP, we can also model problems with more than two labels (even when the number of labels is not known in advance). Graph coloring is such a problem.

A *policy* is a mapping from states to probabilities for each action. Note that we can turn a probabilistic policy into a deterministic *greedy* policy by choosing the action with largest probability. Next, we present how to train this policy end-to-end using policy gradients.

3.2 Policy optimization

We train a parameterized node labeling model by policy gradients, specifically REINFORCE [7] with a greedy rollout baseline [35]. At a high level, the algorithm works as follows. We begin by initializing two models, the *current* model and the *baseline* model. For each graph in the batch, the algorithm performs a probabilistic rollout of the policy. The baseline model performs a greedy rollout. The difference between the two costs determines the policy gradient update. After every epoch, we perform a (one-sided) paired t-test over the cost on a challenge dataset to check if the baseline model should be replaced with the current model. See Appendix A.2 for more details.

4 GRAPH LEARNING ARCHITECTURE

Next, we present a graph learning approach to parameterizing policies for node labeling in our MDP framework. Our MDP formulation is modeled after greedy node labeling algorithms. A greedy node labeling algorithm assigns a label in $\{0, \ldots, n\}$ to one node after another based on a problem-specific heuristic. Hence, it can be seen as providing (1) an order on the nodes and (2) a rule to label the next selected node.

We focus on learning an order on the nodes and pick a label that passes the extensibility test according to a fixed rule. The following two lemmas show there exists a simple *label rule* that ensures the optimal solution can be found for GC and MVC (see Appendix C.2 for the proofs):

Lemma 4.1. *For every graph G, there exists an ordering of vertices for which choosing the smallest color that passes the extensibility test colors G optimally.*

Lemma 4.2. *For every graph G, there exists an ordering of vertices for which choosing the label 1 until every vertex in G is adjacent to a node with label 1 produces a minimum vertex cover of G.*
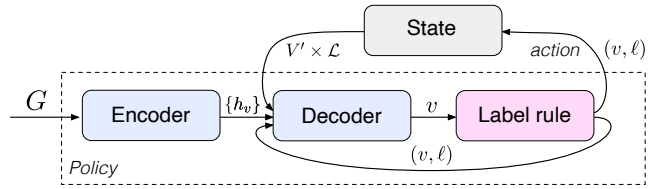

Figure 2: High-level architecture. The encoder (a GNN) reads the input graph and produces embeddings for each node. The decoder takes these embeddings, together with the current state $V' \times \mathcal{L}$ and the last taken action, and outputs the next node. The node v feeds into the label rule, which produces the next action (v, ℓ), leading to the next state $(V' \times \mathcal{L}) \cup \{(v, \ell)\}$. This process repeats until reaching a terminal state (when all nodes are labelled).

We expect similar results can be obtained for most other node labeling problems.

A common strategy in many successful heuristics is to choose the order of the nodes based on their neighborhoods: The ListRight heuristic for MVC [16] assigns a node to the vertex cover based on the assignment of its neighbors. For GC, the DSATUR strategy selects nodes according to their saturation degree [10]. If a new node is selected, only the saturation degree of its neighborhood can change; the others remain unchanged.

Instead of a handcrafted ordering heuristic, we learn to assign weights to each node and choose the nodes according to their weights. To compute these weights, we introduce a novel *spatial locality* inductive bias inspired by the greedy heuristics: labeling a node should only affect the weights of its neighbors. As we will show in Section 5.2, this leads to better test scores compared to the alternatives of updating all or none of the weights when a node is labeled.

4.1 Architecture overview

Our architecture consists of an encoder and a decoder to learn a policy specific the node labeling problem. The encoder learns the local structural information that is important for the problem in the form of a node embedding. It is possible to instantiate any GNN in the encoder.

The *context embedding* encapsulates information about the graph itself (enabling the network to adapt its actions to the graph), the last node that was labeled, and its label. Using only the last node and its label results in a *temporal locality* inductive bias. Adding more nodes into the context embedding provided no benefit (see Section 5.2).

The decoder uses the node embeddings and the context embedding to select the next node based on attention weights between the node embeddings and the context embedding. After the decoder picks the next node v, the label rule (see Lemma 4.1 and Lemma 4.2) assigns the label ℓ for the node. The policy then takes the action (v, ℓ). Then, the context embedding is updated and the decoder is invoked again until all nodes are labelled. Figure 2 overviews our architecture.

4.2 Node features

Each node v is associated with an input feature vector x_v. Our input features consist of a combination of sine and cosine functions of the node degree, similarly to positional embeddings [53]. This representation ensures that input features are bounded in magnitude even for larger graphs. We subtract the mean node degree from the degrees on the synthetic dense graph instances.

4.3 Encoder

We use a hidden dimension of size d (unless stated otherwise, $d = 64$). The input features are first linearly transformed and then fed into a GNN, which produces, for each node v, a node embedding $h_v \in \mathbb{R}^d$. We use a three-layer Graph Attention Network (GAT) [36, 53, 54], additive multi-head attention with four heads, batch normalization [27] with a skip connection [24] at each encoder layer, and leaky ReLU activations [40].

4.4 Context embedding

Next we describe how to construct the *context embedding*, which is an additional input to the decoder. The context embedding is a function of the output of the encoder and the partial node labeling. Each label ℓ has a *label embedding* h_ℓ, which is a max-pooling over the embeddings of the nodes with the same label ℓ. The graph has a *graph embedding* h_G, which is a max-pooling over all node embeddings. In the context embedding, we introduce a *temporal bias* by only considering the last labeled node (and the graph embedding): Specifically, we denote the node that is labeled in time step t by $v^{(t)}$ and its label by $\ell^{(t)}$. Then, the *context embedding* g_t concatenates the following components: (1) The graph embedding h_G, (2) the embedding $h_{v^{(t-1)}}$ of the last labeled node $v^{(t-1)}$, and (3) the label embedding $h_{\ell^{(t-1)}}$ of the last labeled node's label $\ell^{(t-1)}$.

In the first iteration we use a learned parameter $h^{(0)}$ for components (2) and (3). See Figure 3 for an illustration of how the context embedding changes between time steps.

4.5 Local decoder

The decoder takes as input the node embeddings generated by the encoder and the context embedding and outputs the next node to label. In each time step t, an attention mechanism between the context embedding g_t and each node embedding h_v produces attention weights $a_v^{(t)}$. Here, we introduce a *spatial locality* bias: labeling a node can only affect the attention scores of its neighbors in the next time step. Let V' be the set of nodes already labelled. The attention weight $a_v^{(t)}$ for node v in time step t is given by the *local decoding*. For a node $v \notin V'$:

$$a_v^{(t)} = \begin{cases} C \cdot \tanh\left(\frac{(\Theta_1 g_t)^T (\Theta_2 h_i)}{\sqrt{d}} \right) & v \in \mathcal{N}(v^{(t-1)}) \text{ or } t = 0 \\ a_v^{(t-1)} & v \notin \mathcal{N}(v^{(t-1)}) \end{cases}$$

If $v \in V'$, then the attention weight is $a_v^{(t)} = -\infty$. In the first iteration of the decoder, we calculate the coefficients for each node in the graph. As in Bello et al. [7], we clip the attention coefficients within a constant range $[-C, C]$. In our experiments we set $C = 10$. The learnable parameter matrices are $\Theta_1 \in \mathbb{R}^{d \times 3d}$ and $\Theta_2 \in \mathbb{R}^{d \times d}$. We use scaled dot-product attention [53] (instead of additive
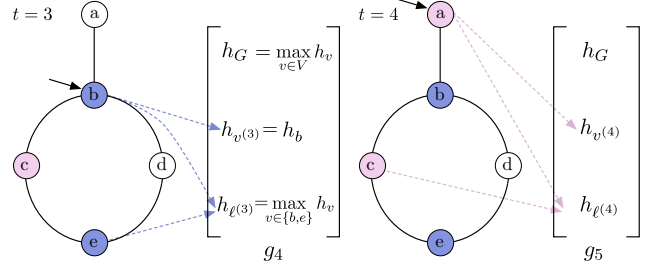

Figure 3: *Temporal locality of the context embedding.* **The context embedding focuses on the last labeled node. It contains the graph embedding and the embeddings of the last labeled node and its label. The example shows two states during graph coloring at time steps $t = 3$ and $t = 4$. At step $t = 3$, the node b has been colored (blue). The context embedding g_4 contains its embedding h_b and the embedding of b's label $\max(h_b, h_e)$. At step $t = 4$, node a is colored (pink). Now, the context embedding g_5 contains $h_{v^{(4)}} = h_a$ and the embedding of a's label $\max(h_a, h_c)$.**

attention) to speed up the decoding. Finally, for each node v we apply a softmax over all attention weights to obtain the probability p_v that node v is labeled next. See Figure 1 for a visualization of the attention weight computation during decoding.

During inference, our *greedy policy* selects the vertices with maximum probability. Our *sampling policy* (for k samples) runs the greedy policy once, then evaluates the learned probabilistic policy k times (selecting a vertex v with the learned probability p_v), returning the best result.

4.6 Number of operations

We express the number of operations (arithmetic operations and comparisons) of the model during inference parameterized by the embedding dimension d, the number of nodes n and the number of edges m. The encoder uses $O(dm + d^2n)$ arithmetic operations and the decoder uses $O(d^2m)$ arithmetic operations, resulting in $O(dm + d^2n + d^2m)$ arithmetic operations, which is *linear in the size of the graph*. To select the action of maximum probability (or sample a vertex), the decoder additionally needs $O(n^2)$ comparison operations (although this could be reduced). As described in Appendix B.3, in practice the d^2m term dominates the runtime for graphs up to $20,000$ vertices and the comparison operations do not dominate the computation.

5 EXPERIMENTS

We evaluate our approach on established benchmarks for graph coloring and minimum vertex cover.

Training We use three different synthetic graph distributions to generate instances for training and validation [3, 19, 55]. We generate 20,000 graphs for training. The graphs have between 20 and 100 nodes. We use Adam with learning rate $\alpha = 10^{-4}$ [34]. The effective batch size is $B = 320$, which comes from using batches of 64 graphs for each node count n and accumulating their gradients. We clip the L2 norm of the gradient to 1, as done in Bello et al. [7].

Table 1: Graph coloring results on the Lemos et al. [37] subset of the COLOR challenge graphs.

	Name	Cost	Wins	Optimal
Classic	Largest First	10.65	50%	45%
	DSATUR	9.85	65%	**50%**
	Smallest Last	10.8	50%	45%
ML	Lemos et al. [37]	N/A	45%	25%
	Ours - Greedy	$10.36^{\pm 0.01}$	55%	**50%**
	Ours - Sampling	$\mathbf{9.65^{\pm 0.04}}$	**70%**	**50%**

We selected these hyperparameters after initial experiments on the validation set. Each model took $15 - 20$ CPU compute node hours to train on a cluster with Intel Xeon E5-2695 v4, 64 GB memory per node. We train each model for 200 epochs with five random seeds and report the standard deviation σ of cost w.r.t. the random seeds as $\pm\sigma$. See Appendix A for more details.

Test Scores In addition to mean cost, we report the ratio of the solution cost to the optimal solution cost (*approximation ratio*). For large graphs, this cannot be computed exactly in a timely manner. In this case, we use the best solution found by an ILP solver within a compute time of 1 hour. To compare with baselines which return infeasible solutions (and hence have ill-defined cost), we report the percentage of *wins* (ties for first place count as wins) and the percentage of instances solved *optimally*. We refer to these metric as 'Wins' and 'Optimal', respectively. We use the model with the lowest cost to compute these percentages.
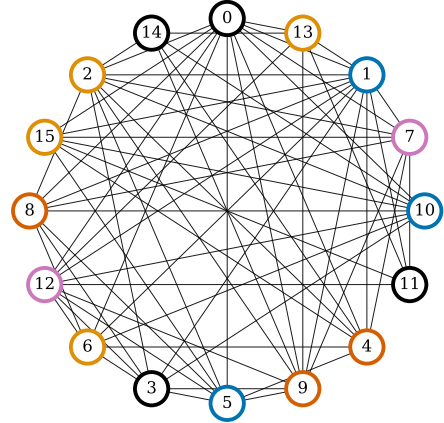
5.1 Results

We compare our approach to existing greedy baselines and machine learning approaches. We focus on other heuristic approaches that return an approximation in polynomial time.

5.1.1 Graph Coloring. **Greedy baselines** *Largest-First* greedily colors nodes in decreasing order of degree. *Smallest-Last* [43] colors the nodes in reverse degeneracy order, which guarantees that when a node is colored, it will have the smallest possible number of neighbors that have been already colored. Smallest-Last guantees a constant number of colors for certain families of graphs, such as Barabási-Albert graphs [3] and planar graphs [43]. *DSATUR* [10] selects vertices based on the largest number of distinct colors in its neighborhood. DSATUR is exact on certain families of graphs, e.g., bipartite graphs [10].
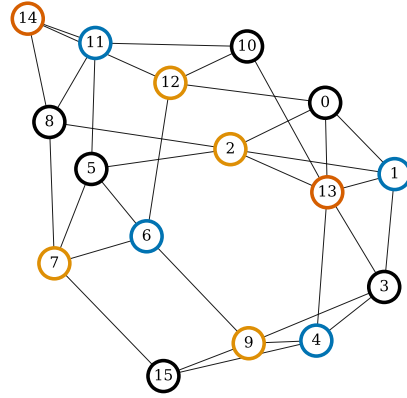
Machine learning baseline We compare our approach with the chromatic number estimator of Lemos et al. [37]. It does not guarantee that the solution is feasible, meaning that it can both under- and overestimate the chromatic number. We use the values reported by the original paper.

Results on COLOR benchmark We evaluate our results on the same subset of the COLOR02/03 benchmark [1] as Lemos et al. [37], consisting of 20 instances of size between 25 and 561 vertices. See Table 1 for the results.
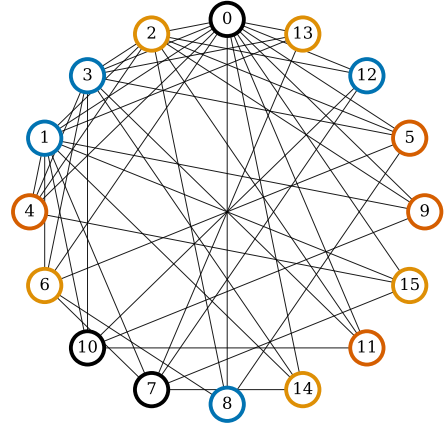
Our greedy policy outperforms both Largest-First and Smallest-Last and is tied with DSATUR for the most graphs solved optimally. When sampling (100 samples) is used to evaluate the policy, our

(a) Erdős-Rényi graph.

(b) Watts-Strogatz graph.

(c) Barabási-Albert graph.

Figure 4: Example colorings produced by our learned heuristic. Node borders indicate the colors. Numbers on the nodes indicate the order in which the heuristic labels them.

Table 2: Comparison of MVC approaches on dense ER graphs with edge-probability 0.15.

	Name	Cost	Approx. Ratio
ML	Li et al. [38]	**212.296**	1.0594
	S2V-DQN	N/A	1.1208
	Ours - Greedy	$221.52^{\pm1.11}$	1.0510
	Ours - 10 Samples	$220.27^{\pm1.20}$	**1.0443**

Table 3: Comparison of MVC approaches on BA graphs with average degree 4.

	Name	Cost	Approx. Ratio
ML	Li et al. [38]	**131.62**	**1.0084**
	S2V-DQN	N/A	1.0099
	Ours - Greedy	$133.78^{\pm0.07}$	1.0234
	Ours - 10 Samples	133.39±0.05	1.0202

Table 4: Comparison of MVC heuristics on the `memetracker` graph.

	Name	Cost	Approx. Ratio
Classic	MVCApprox	666	1.408
	MVCApprox-Greedy	578	1.222
ML	Li et al. [38]	475	1.0042
	S2V-DQN†	**474**	**1.002**
	Ours - Greedy	$484^{\pm2.10}$	1.0273

† trained and evaluated on the same graph.

model outperforms all baselines in *both mean cost and win percentage* and is also tied for the most graphs solved optimally. The approximation ratio is 1.25 and 1.13 for our greedy and sampling policies, resp.

Qualitative Results Figure 4 presents typical examples of the learned coloring heuristic on the training distribution graphs. We can observe that the heuristic generally picks higher degree, centrally located nodes first. However, if several nodes have the same degree, it favors coloring neighboring nodes subsequently. This happens in the WS graphs, see Figure 4b. The learned heuristic can consistently color the WS graphs with 4 colors, which matches the Smallest-Last heuristic. We conclude that the learned heuristic captures complex aspects of the graph extending beyond simple degree-based decisions and considers the graph's local neighborhood structure.

5.1.2 Minimum Vertex Cover. **Greedy baselines** MVCApprox iteratively picks an edge that does have one of its endpoints labeled with 1 and labels both endpoints with 1. MVCApprox-Greedy proceeds similarly, but greedily selects the edge with maximum sum of the degrees of its endpoints. Both algorithms guarantee a 2-approximation [46].

Machine learning baselines *S2V-DQN* is a *Q*-learning based approach [14]. We use the values reported in the original paper. Li et al. [38] present a tree-search based approach trained in a supervised way. In contrast to S2V-DQN, it samples multiple solutions, then verifies if they are feasible. The time to construct a feasible solution varies depending on the instance. We use the publicly available code and pretrained model from the authors in our experiments. We run Li et al.'s code until it finds a feasible solution, and allow it to sample more solutions if it is below the time budget of 30 seconds per graph.

Results on in-distribution graphs We evaluate and compare our approach for MVC with S2V-DQN [14] and Li et al. [38] on the same dataset of generated graphs as. Dai et al. [14]. It consists of 16000 graphs from two distributions, Erdős-Rényi (ER) [19] and Barabási-Albert (BA) [3], of sizes varying from 20 to 600 nodes. We use the results reported by Dai et al. [14] on their model trained on $40 - 50$ nodes, except for the graphs with less than 40 nodes, for which no data is available for this model. Hence we use their model trained on $20 - 40$ nodes on these smaller graphs. See Table 2 for the results on ER graphs and Table 3 for the results on BA graphs.

On ER graphs, our model achieves the closest average approximation ratio, followed by Li et al. [38]. On the BA graphs, our model is about 2.3% away from optimal. In comparison, the two machine learning baselines are slightly less than 1% away from optimal.

Results on real-world graph We evaluate our approach on the `memetracker` graph from Dai et al. [14] which has 960 vertices and 4,888 edges. Note that the S2V-DQN model is trained on subgraphs from the same graph, giving it an advantage over the other models that have not seen the graph during training. See Table 4 for the results. Our approach took less than 1 second to find a vertex cover, whereas Li et al. [38] took 25 minutes to find a solution on the same machine.

Qualitative Results; Figure 5 shows typical results of our learned minimum vertex cover heuristics. On the ER graphs, we can see that the heuristic does not always start with the highest degree node. In contrast, on the BA graphs, the heuristic has a strong preference to start with the highest degree node. In contrast to the classic greedy heuristics (and our learned graph coloring heuristic), the learned MVC heuristics seldomly pick neighboring nodes subsequently.
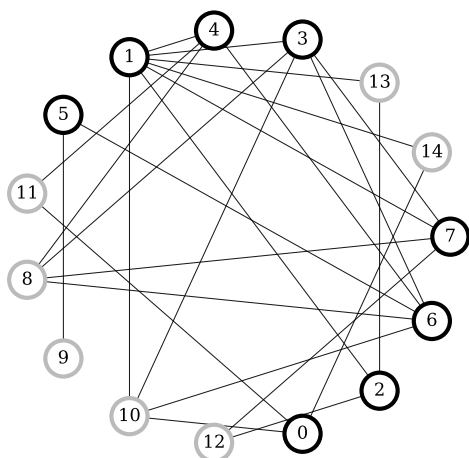
5.2 Investigation of locality

5.2.1 Spatial locality. We test the inductive biases we made regarding locality of the decoder. First, we compare against a variant of the decoder that never updates the attention weights (called *static decoding*) and a decoder that always updates all of the attention weights (called *global decoding*).
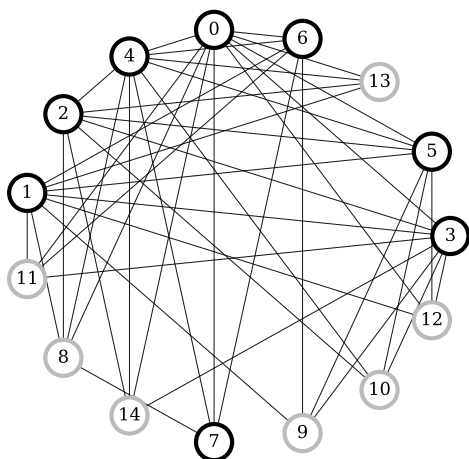
Static Decoding never recomputes the attention weights. For node a node i that is not yet labeled, its weight is:

$$a_i = C \cdot \tanh\left(\frac{(\Theta_1 \mathbf{g_0})^T (\Theta_2 h_i)}{\sqrt{d}}\right)$$

Static decoding uses $O(d^2 n + m + n^2)$ operations, which are fewer than those of local update decoding when $m \gg d^2 n$. With static decoding, the model is essentially a GNN with a special node-readout function.

(a) Erdős-Rényi graph.

(b) Barabási-Albert graph.

Figure 5: Examples of covers produced by our learned heuristic: nodes with a bold black border are in the cover. The numbers indicate the order in which nodes are labelled. Note that as soon as a cover is found, the order of the nodes is irrelevant: they are all labeled to be outside the cover.

We train graph coloring models with static decoding. On the Lemos et al. [37] subset of the COLOR challenge graphs, static decoding achieves a worse mean cost of $10.74^{\pm 0.12}$ when using the greedy policy.

Global Decoding recomputes the attention weights in each time step t. For a node i that is not yet labeled, its weight is:

$$a_i^{(t)} = C \cdot \tanh\left(\frac{(\Theta_1 \mathbf{g_t})^T (\Theta_2 h_i)}{\sqrt{d}}\right)$$

Global decoding uses $O(d^2 n^2)$ operations, which is at least a d^2 factor more than local update decoding for not too dense graphs ($m \ll n^2/d^2$). When there are only two labels (as for MVC), global decoding is very similar to the Kool et al. [35] model. The difference

to Kool et al. [35] is that they use additional attention layer to compute a new context embedding from g_t. Then, in each decoding step, they apply an attention mechanism between the context node and all the nodes that are not yet taken.

Global decoding also achieves a worse mean cost of $10.71^{\pm 0.05}$ (greedy policy) on the Lemos et al. [37] graph coloring test set.

5.2.2 Temporal locality. We varied the size of the context embedding (i.e., the number of nodes and their labels that contribute to it). Increasing the context size does not significantly improve the test score on graph coloring. For graph coloring, a context of size two and three results in a mean cost of $10.49^{\pm 0.12}$ and $10.42^{\pm 0.12}$, respectively, for the greedy policy.

6 CONCLUSION

We presented a unifying framework for learning efficient heuristics to node labeling problems. Since such problems underly many practical applications, providing a comprehensive framework for them broadens the reach and benefits of machine learning. This work contributes to the goal of replacing hand-crafted heuristics with *learned* heuristics tailored to the problems at hand. We demonstrated excellent results on graph coloring and minimum vertex cover as example problems. Future work could include generalizing the architecture to handle weighted graphs and edge labeling problems. In contrast to previous work, our work extends beyond tasks that are simpler to formulate, like minimum vertex cover, to more challenging problems like graph coloring. Formulating a new learning problem in our framework requires a cost function, a labeling rule, and an extensibility test. Our architecture benefits from two inductive biases: a spatial and a temporal bias. These biases allow our model to have nearly-linear operation number scaling and outperform several baselines.

REFERENCES

[1] Computational symposium on graph coloring and generalizations (COLOR02), Ithaca, NY, 7-8 September 2002, 2002. URL https://mat.tepper.cmu.edu/COLOR02/.

[2] Abu-khzam, F., Collins, R., Fellows, M., Langston, M., Suters, W., and Symons, C. Kernelization algorithms for the vertex cover problem: Theory and experiments. pp. 62–69, 01 2004.

[3] Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, Jan 2002. ISSN 1539-0756. doi: 10.1103/revmodphys.74.47. URL http://dx.doi.org/10.1103/RevModPhys.74.47.

[4] Arora, S., Rao, S., and Vazirani, U. V. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009. doi: 10.1145/1502793.1502794. URL https://doi.org/10.1145/1502793.1502794.

[5] Bandh, T., Carle, G., and Sanneck, H. Graph coloring based physical-cell-id assignment for LTE networks. In *Proceedings of the International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly, IWCMC 2009, Leipzig, Germany, June 21-24, 2009*, pp. 116–120, 2009. doi: 10.1145/1582379.1582406. URL https://doi.org/10.1145/1582379.1582406.

[6] Barrett, T. D., Clements, W. R., Foerster, J. N., and Lvovsky, A. Exploratory combinatorial optimization with reinforcement learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 3243–3250, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5723.

[7] Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, 2017. URL https://openreview.net/forum?id=Bk9mxlSFx.

[8] Bhattacharya, S., Henzinger, M., and Nanongkai, D. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on*

Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19, pp. 470–489, 2017. doi: 10.1137/1.9781611974782.30. URL https://doi.org/10.1137/1.9781611974782.30.

[9] Bodlaender, H. L. Discovering treewidth. In *SOFSEM 2005: Theory and Practice of Computer Science, 31st Conference on Current Trends in Theory and Practice of Computer Science, Liptovský Ján, Slovakia, January 22-28, 2005, Proceedings*, pp. 1–16, 2005. doi: 10.1007/978-3-540-30577-4_1. URL https://doi.org/10.1007/978-3-540-30577-4_1.

[10] Brélaz, D. New methods to color vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979. doi: 10.1145/359094.359101. URL https://doi.org/10.1145/359094.359101.

[11] Cappart, Q., Moisan, T., Rousseau, L., Prémont-Schwarz, I., and Ciré, A. A. Combining reinforcement learning and constraint programming for combinatorial optimization. *CoRR*, abs/2006.01610, 2020. URL https://arxiv.org/abs/2006.01610.

[12] Chaitin, G. J. Register allocation & spilling via graph coloring. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, Boston, Massachusetts, USA, June 23-25, 1982*, pp. 98–105, 1982. doi: 10.1145/800230.806984. URL https://doi.org/10.1145/800230.806984.

[13] Cowen, L. J., Cowen, R., and Woodall, D. R. Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. *J. Graph Theory*, 10 (2):187–195, 1986. doi: 10.1002/jgt.3190100207. URL https://doi.org/10.1002/jgt.3190100207.

[14] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *CoRR*, abs/1704.01665, 2017. URL http://arxiv.org/abs/1704.01665.

[15] Dantzig, G. B., Fulkerson, D. R., and Johnson, S. M. Solution of a large-scale traveling-salesman problem. *Oper. Res.*, 2(4):393–410, 1954. doi: 10.1287/opre.2.4.393. URL https://doi.org/10.1287/opre.2.4.393.

[16] Delbot, F. and Laforest, C. A better list heuristic for vertex cover. *Inf. Process. Lett.*, 107(3-4):125–127, 2008. doi: 10.1016/j.ipl.2008.02.004. URL https://doi.org/10.1016/j.ipl.2008.02.004.

[17] Drori, I., Kharkar, A., Sickinger, W. R., Kates, B., Ma, Q., Ge, S., Dolev, E., Dietrich, B., Williamson, D. P., and Udell, M. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *19th IEEE International Conference on Machine Learning and Applications, ICMLA 2020, Miami, FL, USA, December 14-17, 2020*, pp. 19–24, 2020. doi: 10.1109/ICMLA51294.2020.00013. URL https://doi.org/10.1109/ICMLA51294.2020.00013.

[18] Erdős, P. and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[19] Erdős, P. and Rényi, A. On the evolution of random graphs. *Transactions of the American Mathematical Society*, 286:257–257, 1984.

[20] Garey, M. R. and Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990. ISBN 0716710455.

[21] Ghaffari, M., Jin, C., and Nilis, D. A massively parallel algorithm for minimum weight vertex cover. In *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pp. 259–268, 2020. doi: 10.1145/3350755.3400260. URL https://doi.org/10.1145/3350755.3400260.

[22] Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J. (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

[23] Harary, F. and Melter, R. A. On the metric dimension of a graph. *Ars Combinatoria*, 2(191-195):1, 1976.

[24] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

[25] Hedetniemi, S. T. and Laskar, R. C. Bibliography on domination in graphs and some basic definitions of domination parameters. *Discret. Math.*, 86(1-3):257–277, 1990. doi: 10.1016/0012-365X(90)90365-O. URL https://doi.org/10.1016/0012-365X(90)90365-O.

[26] Huang, J., Patwary, M. M. A., and Diamos, G. F. Coloring big graphs with alphagozero. *CoRR*, abs/1902.10162, 2019. URL http://arxiv.org/abs/1902.10162.

[27] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

[28] Jensen, T., Jensen, T., and Toft, B. *Graph Coloring Problems*. A Wiley interscience publication. Wiley, 1995. ISBN 9780471028659. URL https://books.google.ch/books?id=YfZQAAAAMAAJ.

[29] Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *CoRR*, abs/1906.01227, 2019. URL http://arxiv.org/abs/1906.01227.

[30] Karger, D. R. and Stein, C. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. doi: 10.1145/234533.234534. URL https://doi.org/10.1145/234533.234534.

[31] Karger, D. R., Motwani, R., and Ramkumar, G. D. S. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997. doi: 10.1007/BF02523689. URL https://doi.org/10.1007/BF02523689.

[32] Karp, R. Reducibility among combinatorial problems. In Miller, R. and Thatcher, J. (eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum Press, 1972.

[33] Kernighan, B. W. and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x. URL https://doi.org/10.1002/j.1538-7305.1970.tb01770.x.

[34] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[35] Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[36] Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K., and Koh, E. Attention models in graphs: A survey. *ACM Trans. Knowl. Discov. Data*, 13(6):62:1–62:25, 2019. doi: 10.1145/3363574. URL https://doi.org/10.1145/3363574.

[37] Lemos, H., Prates, M. O. R., Avelar, P. H. C., and Lamb, L. C. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. *CoRR*, abs/1903.04598, 2019. URL http://arxiv.org/abs/1903.04598.

[38] Li, Z., Chen, Q., and Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 537–546, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/8d3bba7425e7c98c50f52ca1b52d3735-Abstract.html.

[39] Ma, Q., Ge, S., He, D., Thaker, D., and Drori, I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In *AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications*, 2020.

[40] Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Citeseer, 2013.

[41] Manchanda, S., Mittal, A., Dhawan, A., Medya, S., Ranu, S., and Singh, A. GCOMB: learning budget-constrained combinatorial algorithms over billion-sized graphs. In *Advances in Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e7532dbeff7ef901f2e70daacb3f452d-Abstract.html.

[42] Marx, D. Graph colouring problems and their applications in scheduling. *Periodica Polytechnica Electrical Engineering*, 48:11–16, 2004.

[43] Matula, D. W. and Beck, L. L. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983. doi: 10.1145/2402.322385. URL https://doi.org/10.1145/2402.322385.

[44] Myszkowski, P. B. Solving scheduling problems by evolutionary algorithms for graph coloring problem. In *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, pp. 145–167. 2008. doi: 10.1007/978-3-540-78985-7_7. URL https://doi.org/10.1007/978-3-540-78985-7_7.

[45] Onak, K., Ron, D., Rosen, M., and Rubinfeld, R. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pp. 1123–1131, 2012. doi: 10.1137/1.9781611973099.88. URL https://doi.org/10.1137/1.9781611973099.88.

[46] Papadimitriou, C. and Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. 01 1982. ISBN 0-13-152462-3. doi: 10.1109/TASSP.1984.1164450.

[47] Robson, J. M. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986. doi: 10.1016/0196-6774(86)90032-5. URL https://doi.org/10.1016/0196-6774(86)90032-5.

[48] Smith, M. D., Ramsey, N., and Holloway, G. H. A generalized algorithm for graph-coloring register allocation. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation 2004, Washington, DC, USA, June 9-11, 2004*, pp. 277–288, 2004. doi: 10.1145/996841.996875. URL https://doi.org/10.1145/996841.996875.

[49] Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[50] Tarjan, R. E. and Trojanowski, A. E. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977. doi: 10.1137/0206038. URL https://doi.org/10.1137/0206038.

[51] Tomita, E. and Seki, T. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete Mathematics and Theoretical Computer Science, 4th International Conference, DMTCS 2003, Dijon, France, July 7-12, 2003. Proceedings*, pp. 278–289, 2003. doi: 10.1007/3-540-45066-1_22. URL https://doi.org/10.1007/3-540-45066-1_22.

[52] ul Islam, A. and Kalita, B. Application of minimum vertex cover for keyword – based text summarization process. 2017.

[53] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[54] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

[55] Watts, D. J. and Strogatz, S. H. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998. ISSN 0028-0836. doi: 10.1038/30918. URL http://dx.doi.org/10.1038/30918.

Table 5: The graph parameters for training and validation.

BA	ER	S-ER	WS
$\delta = 4$	$p = 0.15$	$p = p_{s-er}$	$k = 5,$
			$q = 0.1$

A TRAINING

A.1 Data Generation

We use four different synthetic graph distributions to generate instances for training and validation. All graphs are generated via the Python NETWORKX library [22].

Barabási-Albert Model [3] The Barabási-Albert (BA) Model generates random scale-free networks. Similar to real-world networks BA graphs grow by preferential attachment, i.e., a new node is more likely to link to more connected nodes. The model is parameterized by one parameter δ, which dictates the average degree.

Erdős-Rényi Model [19] An Erdős-Rényi (ER) graph $G(n, p)$ has n nodes and each edge exists independently with probability p. The expected number of edges is $\binom{n}{2}p$.

Watts-Strogatz Model [55] Watts-Strogatz (WS) graphs were developed to overcome the shortcomings of ER graphs when modeling real world graphs. In real networks we see the formation of local clusters, i.e., the neighbors of a node are more likely to be neighbors. For parameters k and q, a WS graph is built as follows: build a ring of n nodes. Next, connect each node to its k nearest neighbors. Finally, replace each edge $\{u, v\}$ by a new edge $\{u, w\}$ (chosen uniformly at random) with probability q.

A.1.1 Training set parameters. See Table 5 for the parameters of the graph distributions used during training. Note that for BA and ER graphs, the parameters match those used in the Dai et al. [14] test set (see Table 2 and Table 3). We also consider sparse ER graphs (S-ER), for we set the edge probability such that graphs have expected average degree close to $\Delta = 7.5$ when n is small but remain connected with high probability when n is large. This means that

$$p_{s-er} = \min\left(1, \max\left(\frac{\Delta}{n}, (1 + \epsilon)\frac{\ln n}{n}\right)\right) , \tag{1}$$

for a small ϵ, which we set to 0.2 in our experiments. The formula is derived from the connectivity threshold of ER graphs [18].

For graph coloring, we train on a hybrid dataset consisting of an equal proportion of BA, S-ER, and WS graphs. For minimum vertex cut, we train on a dataset consisting of BA graphs, a dataset consisting of ER graphs, and a hybrid dataset consisting on a combination of the two (in equal proportion). We use the in-distribution models for the evaluation on the synthetic test instances and the hybrid model for the memetracker graph. During training, we use an equal proportion of graphs with $n \in \{20, 40, 50, 70, 100\}$ nodes.

A.2 Policy Optimization

We train our model with REINFORCE with a greedy rollout baseline Kool et al. [35]. The details follow. We denote the cost of labeling the graph G_i in the order given by the sequence of nodes π by $\mathcal{L}(\pi, G_i)$. A model M is parameterized by parameters θ. On a graph G_i, the model returns a sequence of nodes π and an associated probability p_θ. The probability p_θ is the product of all action probabilities that led to the sequence of nodes π. We write $p_\theta, \pi \leftarrow M_\theta(G_i)$ when the policy is evaluated deterministically and $p_\theta, \pi \sim M_\theta(G_i)$ when the policy is evaluated probabilistically.

The complete training procedure is given in Algorithm 1. Note that Algorithm 1 follows from the textbook REINFORCE with a baseline [49] by factoring the probability of reaching a terminal state and using that the rewards are 0 in our MDP except when reaching a terminal state. Unlike Kool et al. [35], we do not use warmup epochs where an exponential moving average baseline is used in the first epochs of training.

B ADDITIONAL RESULTS

B.1 Results by graph size

Our test and validation data already include graphs larger than those seen in training. In the main paper (Tables 1-3), we report the average across the graph sizes. See Table 6 for a breakdown of Table 2, where we show how the approximation ratio varies with the test instance size on minimum vertex cover (MVC). For graph coloring (GC), about a third of the test instances have more vertices than seen in training. The average approximation ratio on those instances is 1.18 and 1.14 for the Greedy and the Sampling policies, respectively. This is comparable to the results on the overall test set (1.25 and 1.13 for the Greedy and the Sample policy, respectively).

B.2 Validation Results

We compare the cost of the learned heuristic for different parameters of the training. The validation set consists of 600 graphs with n nodes for $n \in \{20, 50, 100, 200, 400, 600\}$.

Algorithm 1 Policy Training with REINFORCE + Baseline.

Input: number of epochs E, batch size B, dataset D_{train}, earing rate α

Initialize model M_θ and baseline model M_θ^{BL}

$D_{\text{challenge}} \leftarrow$ Sample new challenge dataset

for epoch $= 1, \ldots, E$ **do**

    for batch in D_{train} **do**

      $[\ p_{\theta,i}, \pi_i \sim M_\theta(G_i)\ \text{for } G_i \text{ in } batch\ ]$ // Sample from policy

      $[\ p_{\theta,i}^{BL}, \pi_i^{BL} \leftarrow M_{\theta^{BL}}(G_i)\ \text{for } G_i \text{ in } batch\ ]$ // Greedy baseline

      $\nabla_\theta J(\theta) = \frac{1}{B} \sum_{i=1}^{B} (\mathcal{L}(\pi_i \mid G_i) - \mathcal{L}(\pi_i^{BL} \mid G_i))\ \nabla_\theta \log(p_{\theta,i})$ // Policy gradient

      $\theta \leftarrow Gradient\ Descent(\theta, \nabla_\theta J(\theta), \alpha)$

    end for

                                                         //Challenge the baseline

    if $OneSidedPairedTTest(M_\theta, M_\theta^{BL}, D_{\text{challenge}}) < 0.05$ **then**

      $\theta^{BL} \leftarrow \theta$

      $D_{\text{challenge}} \leftarrow$ Sample new challenge dataset

    end if

end for

Table 6: Cost and approximation ratio of our approach for MVC on ER graphs (10 samples) from Table 2 by instance size.

Vertices	15-20	40-100	100-300	300-500	500-600
Cost	8	43.8	178.4	384.9	540
Appr. Ratio	1.012	1.042	1.048	1.055	1.054

Table 7: Validation Cost for GC.

Training Distr.	Cost S-ER	Cost WS	Cost BA
S-ER+WS+BA	$5.32^{\pm 0.05}$	$4.01^{\pm 0.00}$	$5.50^{\pm 0.04}$

Table 8: Validation Cost for MVC.

Training Distribution	Cost ER	Cost BA
ER	$\mathbf{223.56^{\pm 0.11}}$	$218.07^{\pm 4.36}$
BA	$223.76^{\pm 0.43}$	$\mathbf{199.44^{\pm 12.02}}$
ER+BA	$223.98^{\pm 0.26}$	$208.18^{\pm 6.24}$

B.2.1 Graph coloring. Table 7 shows the validation cost on the three training distribution for the configuration used in the experiments. Over all three distributions, the mean validation cost is $4.95^{\pm 0.02}$.

    Number of attention heads We varied the number of attention heads (among 1, 2, 4) while keeping the dimension per head to 16. For graph coloring, this results in a mean validation cost of $5.29^{\pm 0.02}$, $4.98^{\pm 0.01}$, and $4.95^{\pm 0.02}$, respectively. Hence, 4 attention heads (overall hidden dimension 64) yields the lowest mean validation cost for graph coloring.

    Learning rate With a larger learning rate of $\alpha = 10^{-3}$, the mean validation cost for graph coloring is significantly worse, namely $5.22^{\pm 0.002}$. A smaller learning rate of $\alpha = 10^{-5}$ leads to a mean validation cost of $5.02^{\pm 0.001}$, which is slightly worse than the cost for $\alpha = 10^{-4}$.

B.2.2 Minimum vertex cover. Table 8 shows the validation results for training on either only one distribution and evaluating on ER and BA graphs. Training on a mixture ER and BA graphs leads to worse validation cost on BA graphs compared to training only on BA graphs. Training on ER graphs exclusively without BA graphs leads to a slight cost improvement on ER graphs.
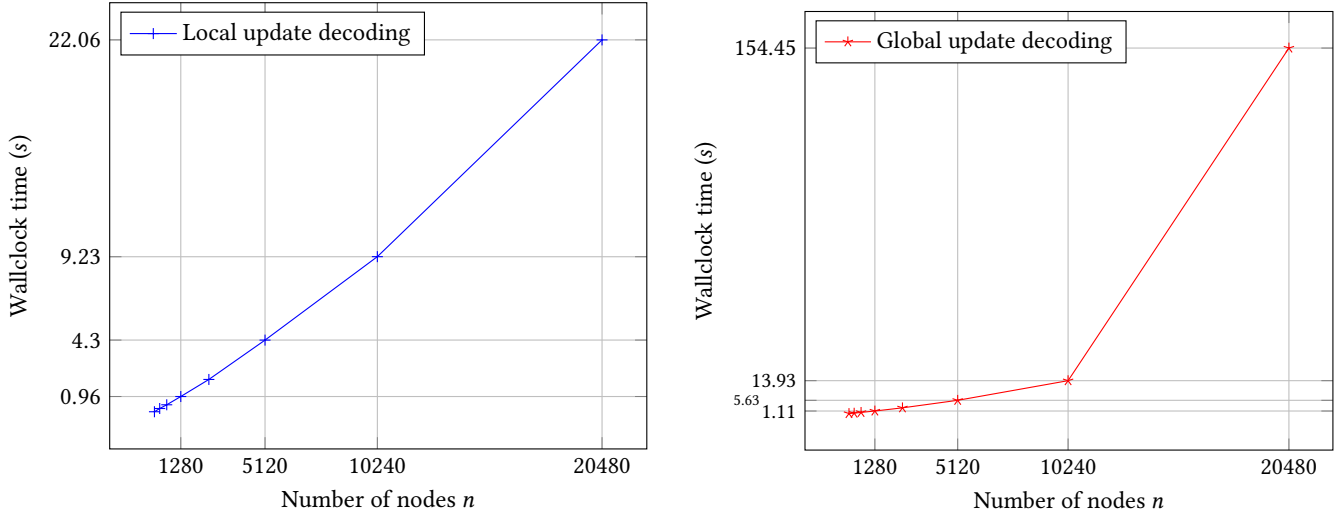
Figure 6: Runtime scaling of graph coloring inference on ER graphs for local and global decoding.

B.3 Runtime Scalability

Figure 6 shows how the runtime of our graph coloring model scales with the size of the graph on S-ER graphs with the edge probability p as in validation according to Equation (1). Each datapoint is the mean of 10 samples. We can see that while our model with the local decoding has nearly linear runtime scaling, the global decoding (from Section 5.2.1) does not scale well to graphs with more than 10,000 nodes. In particular, it takes less than 1 second to color ER graphs with 1,280 vertices (> 5,000 edges) and less than 22.1 seconds to color graphs with 20,480 vertices (> 100,000 edges). If the local decoding is replaced with globally updating the attention weights, the runtime increases significantly to more than 154 seconds to color the same graphs with 20,480 vertices.

Similar results hold for MVC: it takes less than 1 second to compute a vertex cover for a graph with 1,000 vertices and less than 40 seconds to compute a vertex cover for a graph with 20,480 vertices (> 100,000 edges). With global decoding, it takes 180.9 to compute a vertex cover.

C ADDITIONAL PROOFS

C.1 The node labeling MDP

PROOF OF LEMMA 3.1. Consider a sequence of actions $(v_1, \ell_1), \ldots, (v_n, \ell_n)$ ending in a terminal state. For all t, the prefix $(v_1, \ell_1), \ldots, (v_t, \ell_t)$ of this sequence corresponds to a partial node labeling c' (by viewing the sequence of node-label pairs as describing a function from nodes to labels). By construction of the MDP, labeling node v_{i+1} with ℓ_{t+1} passes the extensibility test for c'. Hence the node labelling c represented by $(v_1, \ell_1), \ldots, (v_n, \ell_n)$ is feasible. By constriction, the return of the episode is $-f(c)$, where $f(c)$ is the cost of node labeling c.

Conversely, consider a feasible solution c with cost $f(c)$. Then, by definition of feasibility (Section 5.2), there is a sequence (v_1, ℓ_1), $\ldots, (v_n, \ell_n)$ of node-label pairs such that for all $t \geq 0$ the partial node labeling given by $(v_1, \ell_1), \ldots, (v_t, \ell_t)$ passes the extensibility test for node v_{t+1} and label ℓ_{t+1}. Hence, the sequence of node-label pairs is also a sequence of actions in the MDP leading to a terminal state. The return for this episode is $-f(c)$. □

C.2 Optimality of the labeling rule

PROOF OF LEMMA 4.1. Let G be some graph with chromatic number $\chi(G) = k$ and c^* be a mapping that colors G optimally.

We partition V into color classes $C_i = \{v \mid c^*(v) = i\}$ such that all nodes with color i are in C_i. Now, we build an ordering by consecutively taking all nodes from C_1, then all nodes from C_2 and so on. Choosing the smallest color that passes the extensibility test will produce an optimal coloring for such an order of nodes. This coloring might be different from the one of c^*. This is, because a node in C_i might have no conflicts with some color $j < i$ and therefore this node will be assigned color j. However, by assumption all nodes that have color j in c^* are already colored and a node from C_i can have at most $i - 1$ conflicting colors in its neighborhood. Hence, the algorithm will produce a coloring of at most k colors, which we assumed to be optimal. □

PROOF OF LEMMA 4.2. Let S be the set over nodes with label 1 in a minimum vertex cover of G. Order these nodes first (in an arbitrary relative order), then order the remaining nodes in $V - S$ after these nodes (in an arbitrary relative order). Labeling the nodes in this order produces a minimum vertex cover of G. □

Table 9: Node labeling problems which partition the nodes into 2 or more sets.

Problem	Extensibility Test $T(V' \times \mathcal{L}, v, l)$	Cost function f
Balanced k-partition [33]	There are no more than $\lceil \frac{n}{k} \rceil$ nodes with the same label and at most k labels.	$\sum_{\{u,v\} \in E, l(u) \neq l(v)} w(u,v)$
Balanced $(k, 1+\epsilon)$-partition [33]	There are no more than $\lceil \frac{n(1+\epsilon)}{k} \rceil$ nodes with the same label and at most k labels.	$\sum_{\{u,v\} \in E, l(u) \neq l(v)} w(u,v)$
Minimum k-cut [30]	$k - |V| - |V'| - 1 \leq |\mathcal{L} \cup \{v\}|$ and $|\mathcal{L} \cup \{v\}| \leq k$	$\sum_{\{u,v\} \in E, l(u) \neq l(v)} w(u,v)$
Clique cover [32]	Every label induces a clique	Number of labels
Domatic number [25]	Every label induces a dominating set of $G[V' \cup \{v\}]$	Negative number of labels
Graph coloring [28, 32]	No neighbor of v has label l	Number of labels
Graph co-coloring [28]	The nodes with label l induce an independent set in G or the complement of G	Number of labels
k-defective coloring [13]	No node has more than k neighbors with label l	Number of labels

Table 10: Node labeling problems where the labels encode a sequence of nodes.

Problem	Extensibility Test $T(V' \times \mathcal{L}, v, l)$	Cost function f
Traveling salesman problem [15]	$l = \max(\mathcal{L}) + 1$ and v is a neighbor of the node in $\mathcal{L}$ with label $\max(\mathcal{L})$	$\sum_{(u,v) \in E, l(v)=l(u)+1} w(u,v)$
Tree decomposition [9]	$l = \max(\mathcal{L}) + 1$	For a node v_i with label i, add edges to G until v_i forms a clique with its higher-labelled neighbors. The cost is the largest number of higher-labelled neighbors in the augmented graph [9].
Longest path [31]	$l = \max(\mathcal{L}) + 1$	Maximum number of nodes with consecutive labels that induce a path

D LIST OF COMBINATORIAL NODE LABELING PROBLEMS

We provide an extensive list of classic graph optimization problems framed as node labeling problems. Note that there can be multiple equivalent formulations. For some problems, we consider a *weighed graph* G with weight function $w : E \mapsto \mathbb{R}^+$, we write $w(u,v)$ the weight of an edge $\{u,v\}$. For a set of nodes S, we denote the subgraph of G induced by S with $G[S]$.

The problems in Table 9 require a partition of the nodes as their solution. These can be represented as node labeling problems by giving each partition its unique label. For many of the problems, the number of used labels determines the cost function.

The problems in Table 10 require a path (or a sequences of nodes) as their solution, which we represent as node labeling problems by having the label indicate the position in the path (or sequence).

The problems in Table 11 require a set of nodes as their solution. These can be represented as node labeling problems by giving the nodes in the solution set the label 1 and the nodes not in the solution set the label 0. The cost function is closely related to the number of nodes with label 1 for most of these problems.

Table 11: Node labeling problems with binary labels. *For all these problems,* the extensibility test passes only if the label is 0 or 1 (and the additional requirements listed below are satisfied).

Problem	Extensibility Test $T(V' \times \mathcal{L}, v, l)$	Cost function f
Maximum cut [32]	At least one node has label 1	$-\|\{\{u, v\} \in E, l(u) \neq l(v)\}\|$
Sparsest cut [4]	At least one node has label 1	$\frac{\|\{\{u,v\} \in E, \, l(u) \neq l(v)\}\|}{\|\{v \in V, \, l(v)=1\}\|}$
Maximum independent set [47, 50]	The subgraph induced by the nodes with label 1 is an independent set	$-\|\{v \in V, l(v) = 1\}\|$
Minimum vertex cover (node cover) [32]	The subgraph induced by the nodes with label 1 is a vertex cover of $G[V' \cup \{v\}]$	$\|\{v \in V, l(v) = 1\}\|$
Maximum clique [51]	The subgraph induced by the nodes with label 1 is a clique	$-\|\{v \in V, l(v) = 1\}\|$
Minimum feedback node set [32]	$G[\{u \in V' \cup \{v\}, l(u) = 0\}]$ is a forest	$\|\{v \in V, l(v) = 1\}\|$
Metric dimension [23]	The nodes in $V' \cup \{v\}$ are uniquely identified by their distances to nodes with label 1	$\|\{v \in V, l(v) = 1\}\|$
Minimum dominating set [25]	The nodes with label 1 form a dominating set of $G[V' \cup \{v\}]$	$\|\{v \in V, l(v) = 1\}\|$
Minimum connected dominating set [25]	The nodes with label 1 form a connected dominating set of $G[V' \cup \{v\}]$	$\|\{v \in V, l(v) = 1\}\|$