# Algorithmic Concept-Based Explainable Reasoning

**Dobrik Georgiev**[1], **Pietro Barbiero**[1], **Dmitry Kazhdan**[1], **Petar Veličković**[2], **Pietro Liò**[1]

[1]University of Cambridge
[2]DeepMind
{dgg30, pb737, dk525}@cam.ac.uk, petarv@deepmind.com, pl219@cam.ac.uk

## Abstract

Recent research on graph neural network (GNN) models successfully applied GNNs to classical graph algorithms and combinatorial optimisation problems. This has numerous benefits, such as allowing applications of algorithms when preconditions are not satisfied, or reusing learned models when sufficient training data is not available or can't be generated. Unfortunately, a key hindrance of these approaches is their lack of explainability, since GNNs are black-box models that cannot be interpreted directly. In this work, we address this limitation by applying existing work on concept-based explanations to GNN models. We introduce *concept-bottleneck GNNs*, which rely on a modification to the GNN readout mechanism. Using three case studies we demonstrate that: (i) our proposed model is capable of accurately learning concepts and extracting propositional formulas based on the learned concepts for each target class; (ii) our concept-based GNN models achieve comparative performance with state-of-the-art models; (iii) we can derive global graph concepts, without explicitly providing any supervision on graph-level concepts.

## Introduction

Graph neural networks (GNNs) have successfully been applied to problems involving data with irregular structure, such as quantum chemistry (Gilmer et al. 2017), drug discovery (Stokes et al. 2020), social networks (Pal et al. 2020) and physics simulations (Battaglia et al. 2016). One of the latest areas of GNN research focuses on using GNNs for emulation of classical algorithms (Cappart et al. 2021). In particular, this research explored applications of GNNs to iterative algorithms (Veličković et al. 2020b; Georgiev and Liò 2020), pointer-based data structures (Veličković et al. 2020a; Strathmann et al. 2021), and even planning tasks (Deac et al. 2020). Importantly, these works demonstrate that GNNs are capable of *strongly generalising* to input graphs much larger than the ones seen during training.

Unfortunately, in all of the aforementioned cases, these state-of-the-art GNN models are black-boxes, whose behaviour cannot be understood/intepreted directly. In practice, this can lead to a lack of trust in such models, making it challenging to apply and regulate these models in safety-critical

applications, such as healthcare. Furthermore, this lack of interpretability also makes it difficult to extract the knowledge learned by such models, which prevents users from better understanding the corresponding tasks (Adadi and Berrada 2018; Molnar 2020; Doshi-Velez and Kim 2017).

Recent work on Explainable AI (XAI) introduced a novel type of Convolutional Neural Network (CNN) explanation approach, referred to as *concept-based explainability* (Koh et al. 2020; Kazhdan et al. 2020b; Ghorbani et al. 2019; Kazhdan et al. 2021). Concept-based explanation approaches provide model explanations in terms of human-understandable units, rather than individual features, pixels, or characters (e.g., the concepts of a wheel and a door are important for the detection of cars) (Kazhdan et al. 2020b). In particular, work on Concept Bottleneck Models (CBMs) relies on concepts and introduces a novel type of interpretable-by-design CNN, which perform input processing in two distinct steps: computing a set of concepts from an input, and then computing the output label from the concepts (Koh et al. 2020).

In this paper, we apply the idea of CBMs to GNN models, by introducing Concept Bottleneck Graph Neural Networks (CBGNNs). In particular, we rely on the *encode-process-decode* paradigm (Hamrick et al. 2018), and apply concept bottleneck layers before the output of GNN models – see Figure 1. By doing this we are able to extract update/termination rules for the step updates of *step-level* combinatorial optimisation approaches (Veličković et al. 2020b,a; Deac et al. 2020; Strathmann et al. 2021).

Importantly, we show that by relying on a suitable set of concepts and supervising on them, we are capable of deriving the rules of classical algorithms such as breadth-first search (Moore 1959), and Kruskal's algorithm (Kruskal 1956), as well as more advanced heuristics such as parallel graph coloring (Jones and Plassmann 1993). Furthermore, we present an approach to utilise node-level concepts for extracting graph-level rules. Our evaluation experiments demonstrate that all of our extracted rules strongly generalise to graphs of $5\times$ larger size.

To summarise, we make the following contributions in this work:

- Present *Concept Bottleneck Graph Neural Networks* (CBGNN), a novel type of GNN relying on intermediate concept processing. To the best of our knowledge, this is the first work to apply concept bottleneck approaches
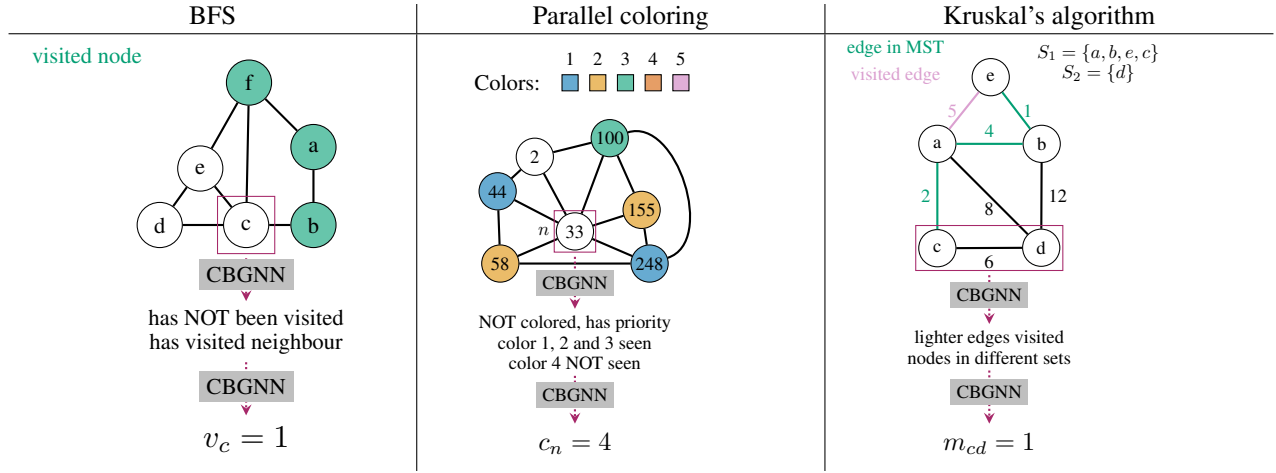
Figure 1: An overview of our Concept Bottleneck Graph Neural Network (CBGNN) approach. Importantly, CBGNN models can be trained to extract concept information for a given task as well as algorithm rules. We give examples of 3 algorithms, showing how CBGNN extract concepts from the input data and then uses these to compute the output.

to GNNs.

- Quantitatively evaluate our approach using three different case-studies (BFS, graph colouring, and Kruskal's), showing that our CBGNN approch is capable of achieving performance on-par with that of existing state-of-the-art

- Qualitatively evaluate our approach, by demonstrating how the concepts utilised by CBGNN models can be used for providing rules summarising the heuristics the CBGNN has learned

## Related Work

**GNN Explainability** Recent work began exploring applications of XAI techniques in the context of GNNs. For instance, work in (Pope et al. 2019; Baldassarre and Azizpour 2019; Schnake et al. 2020) adapt feature-importance gradient-based approaches used for CNN applications (such as Class Activation Mappings, or Layer-wise Relevance Propagation) to GNNs, in order to identify the most important nodes/subgraphs responsible for individual predictions. Alternatively, works in (Ying et al. 2019; Vu and Thai 2020; Luo et al. 2020) focus on more complex approaches unique to GNN explainability, such as those based on mutual information maximisation, or Markov blanket conditional probabilities of feature explanations. Importantly, these works focus on GNN tasks and benchmarks involving social networks, chemistry, or drug discovery, instead of focusing on combinatorial optimisation tasks, which is the focus of this work. Furthermore, these works focus on explaining pre-trained GNNs in a post-hoc fashion, whereas we focus on building GNN models interpretable-by-design. Finally, these works focus on feature-importance-based explanation approaches (i.e. returing relative importance of input nodes/subgraphs), whereas we rely on concept-based explanation approaches instead.

**Concept-based Explainability** A range of existing works have explored various concept-based explanations applied to CNN models. For instance, work in (Ghorbani et al. 2019;

Kazhdan et al. 2020b; Yeh et al. 2019) introduce approaches for extracting concepts from pre-trained CNNs in an unsupervised, or semi-supervised fashion. Work in (Chen, Bei, and Rudin 2020; Koh et al. 2020) rely on concepts for introducing CNN models interpretable-by-design, performing processing in two distinct steps: concept extraction, and label prediction. Other works on concepts include studying the connection between concepts and disentanglement learning (Kazhdan et al. 2021), as well as using concepts for data distribution shifts (Wijaya et al. 2021). Importantly, these works explore concepts exclusively in the context of CNNs, with (Kazhdan et al. 2020a) being the only work exploring concepts in the context of RNN models. In this work, we focus on concept-based explainability for GNNs, where, similar to Koh et al. (2020), the concepts are human-specified.

**Combinatorial Optimisation for GNNs** Following the hierarchy defined in Cappart et al. (2021), our work classifies as a step-level approach. We directly extend on Veličković et al. (2020a,b), therefore we use the models presented in these works as baselines. We *do not* compare our model to an algorithm-level combinatorial optimisation approaches (Xu et al. 2020; Tang et al. 2020; Joshi et al. 2020) or unit-level ones (Yan et al. 2020) for the following reasons: *Algorithm-level approaches usually give one output per data sample* (rather than one output per step), but rules/invariants of a given algorithm come from how the iteration proceeds making algorithm-level combinatorial optimisation less suitable for a concept bottleneck. *Unit-level learning focuses on learning primitive units of computation*, such as taking maximum or merging lists and then combining these manually – having explanations at this level would not be of great benefit. To the best of our knowledge, only Veličković et al. (2020a) attempted to explain GNN predictions, using GNNExplainer (Ying et al. 2019). However, their model (i) was not explainable by design and (ii) required further optimisation *for a single sample* to give a *local explanation*. All other previous works operated in a black-box fashion and did not consider

explainability of the learnt models.

## Methodology

**Encode-process-decode** Following the "blueprint" for neural execution outlined in Veličković et al. (2020b), we model the algorithms by the encode-process-decode architecture (Hamrick et al. 2018) applied on a graph with nodes $V$ and edges (edge index) $E$. For each algorithm $A$, at a (iteration) timestep $t$ of the algorithm, an *encoder network* $f_A$ encodes the algorithm-specific node-level inputs $z_i^{(t)}$ into the latent space. These node embeddings are then processed using the *processor network* $P$, usually a GNN. The processor takes as input the encoded inputs $\mathbf{Z}^{(t)} = \{z_i^{(t)}\}_{i \in V}$ and graph edge index $E$ to produce latent features $\mathbf{H}^{(t)} = \{\mathbf{h}_i^{(t)} \in \mathbb{R}^{|\mathcal{L}|}\}_{i \in V}$, where $|\mathcal{L}|$ is the size of the latent dimension. In contrast with previous work, we calculate algorithm outputs by first passing the latent embeddings through a *decoder network* $g_A'$, which produces concepts for each node $\mathbf{C}^{(t)} = \{c_i^{(t)} \in (0,1)^{|\mathcal{C}|}\}$, where $|\mathcal{C}|$ is number of concepts. The concepts are then passed through a *concept decoder* $g_A$ to produce node-level outputs $\mathbf{Y}^{(t)} = \{\mathbf{y}_i^{(t)}\}$.

Where applicable, we also utilise a *termination network* $T_A$ for deciding when to stop the algorithm's execution. However, in contrast with prior work, we observed that training is more stable if we calculate the termination probability based on potential *next step* embeddings (i.e. a belief about what is the state after an iteration has been executed). Additionally we found it insufficient to use the average node embeddings as input to $T_A$ – averaging would obfuscate the signal if there is just a single node which should tell us whether to continue iterating or not. Instead, we opted to use the output of an adapted PrediNet (Shanahan et al. 2020) architecture with one attention head. PrediNet is designed to represent the conjunction of elementary propositions, therefore it can (theoretically) capture the logical bias of the termination rules. The whole idea is visualised in Figure 1 as well as in the equations below (and further detailed in the Appendix):

$$z_i^{(t)} = f_A\left(\mathbf{x}_i^{(t)}, \mathbf{h}_i^{(t-1)}\right) \tag{1}$$

$$\mathbf{H}^{(t)} = P\left(\mathbf{Z}^{(t)}, E\right) \tag{2}$$

$$c_i^{(t)} = \sigma\left(g_A'\left(z_i^{(t)}, \mathbf{h}_i^{(t)}\right)\right) \tag{3}$$

$$\mathbf{y}_i^{(t)} = g_A\left(c_i^{(t)}\right) \tag{4}$$

$$z_i'^{(t)} = f_A\left(\mathbf{y}_i^{(t)}, \mathbf{h}_i^{(t)}\right) \tag{5}$$

$$\mathbf{H}'^{(t)} = P\left(\mathbf{Z}'^{(t)}, E\right) \tag{6}$$

$$\overline{\mathbf{H}^{(t)}} = \text{PrediNet}(\mathbf{H}'^{(t)}) \tag{7}$$

$$\tau^{(t)} = \sigma\left(T_A\left(\overline{\mathbf{H}^{(t)}}\right)\right) \tag{8}$$

where $\sigma$ is a logistic sigmoid function. When using $T_A$, equations 1-8 are repeated if $\tau^{(t)} > 0.5$.

The combination of encoded inputs, together with the latent state of the given node, contains sufficient information

not only about the output at a given step, but also: (i) a node's current state and (ii) observations about other nodes' states in its neighbourhood. If our concepts are engineered to capture some knowledge of either (i) or (ii), then we can extract meaningful algorithm output explanations *without providing any explicit information about how the algorithm works (theorems, invariants, etc.)*

**Explicitly relational GNN architecture** Some graph level tasks (e.g. deciding termination) can be reduced to a logical formula over all nodes – for the algorithms and concepts we consider, termination can be reduced to existence of a node with specific properties. (See graph-level rule extraction). We engineer this logical bias into the termination network $\tau$ by adapting PrediNet (Shanahan et al. 2020) to the graph domain. The PrediNet network architecture learns to represent conjunction/disjunction of elementary propositions and is therefore suitable for the termination task. We list the two minor modifications we made to adapt PrediNet to our tasks in Appendix A.

**Extracting node-level algorithm rules** After the network has been trained deciding node-level formulas for algorithm $A$ is achieved by examining the weights of the concept decoder $g_A$. To achieve this, we used the open-source package `pytorch_explain`[1] (Barbiero et al. 2021) implementing a wide collection of techniques to extract logic-based explanations from concept-bottleneck neural networks (Gori 2017; Ciravegna et al. 2020). The library takes as inputs (i) node-level output decoder weights, (ii) predicted concepts from training data, and (iii) training data ground truth labels, and generates logic formulas in disjunctive normal form as outputs (Mendelson 2009). By construction, the concept decoder $\bar{g}_A \approx g_A$ learnt from concepts $\mathcal{C}$ to outputs $\mathcal{O}$ is a Boolean map. As any Boolean function, it can be converted into a logic formula in disjunctive normal form by means of its truth-table (Mendelson 2009). The weights of the concept decoder $g_A$ are used to select the most relevant concepts for each output task. To get concise logic explanations *when many concepts are required as inputs* (in our experiments this is only the graph coloring task), we add a regularization term in the loss function minimising the $L1$-norm of the concept decoder weights $W$, leading to sparse configurations of $W$. Later, at the training epoch $t_{\text{prune}}$, first-layer weights are pruned concept-wise, i.e. removing *all* the weights departing from the least relevant concepts:

$$\tilde{W}_j^1 = W_j^1 \mathbb{I}_{||W_j^1||_1 \geq \max_i \frac{||W_i^1||_1}{2}}, \qquad \text{for } i = 1, \ldots, |\mathcal{C}| \tag{9}$$

where $\mathbb{I}$ is the indicator function and $W^1$ are weights of the first layer. Further details on logic extraction are provided in Appendix B.

**Extracting algorithm termination rules** When deciding whether to continue execution, we use the fact that a number of graph algorithms continue iterating until a node with a specific combination of concepts exists. Since it is unclear what combination of concepts we should supervise towards, we

---

[1]Apache 2.0 Licence.

| Algorithm | Concepts | Example ground-truth explanations (not provided to the model) |
|---|---|---|
| BFS | $hasBeenVisited$ $(hBV)$ $hasVisitedNeighbours$ $(hVN)$ | $hVN(i) \implies y_i^{(t)} = 1$ $\exists i.\neg hBV(i) \wedge hVN(i) \implies \tau^{(t)} = 1$ |
| Coloring | $isColored$ $(iC)$, $hasPriority$ $(hP)$ $colorXSeen$ $(cXS)$, $X \in \{1,..,5\}$ | $iC(i) \wedge c1S(i) \wedge \neg c2S(i) \implies y_i^{(t)} = 2$ $(\neg iC(i) \wedge hP(i) \wedge c1S(i) \wedge c2S(i)$ $\wedge \neg c3S(i)) \implies y_i^{(t)} = 3$ |
| Kruskal's | $lighterEdgesVisited$ $(lEV)$ $nodesInSameSet$ $(nISS)$ $edgeInMst$ $(eIM)$ | $(lEV(i) \wedge \neg nISS(i) \wedge \neg eIM(i))$ $\implies y_i^{(t)} = 1$ $(nISS(i) \wedge \neg eIM(i))$ $\implies y_i^{(t)} = 0$ |

Table 1: Algorithms and their corresponding concepts. We provide some sample ground truth explanations. Visual examples of how the algorithms work can be seen in Figure 1.

took a full enumeration approach, applied only to the training data, when extracting rules for termination. First, we generate a sample $j$ of the form $(\mathcal{U}_j, \tau'_j)$ from the training set from each step for a given graph. $\tau'_j$ is the ground truth for whether we should keep iterating, and $\mathcal{U}_j = \{\mathbf{c}'_1, \ldots, \mathbf{c}'_k\}$ is a set of all unique concepts combinations,[2] *after the algorithm update state has been performed* (hence $\mathbf{c}'$). Given a set of concept indexes[3] $\mathcal{I} \subseteq \mathcal{P}(\{1..|\mathcal{C}|\})$, where $\mathcal{P}$ denotes power-set, and truth assignment $\mathcal{T} : \{1..|\mathcal{C}|\} \to \{0,1\}$ telling us which concepts *must* be true/false, we check if the following is satisfied:

$$\forall j \left( \tau'_j = 1 \iff (\exists \mathbf{c} \in \mathcal{U}_j. \forall \mathcal{I}_i \in \mathcal{I}. \mathbf{c}_{\mathcal{I}_i} = \mathcal{T}(\mathcal{I}_i)) \right) \tag{10}$$

i.e. we should continue iterating if a special concept combination exists, and we should stop iterating if it does not. We employ a brute-force approach for finding $\mathcal{I}$ and $\mathcal{T}$, breaking ties by preferring smaller $\mathcal{I}$.[4] The complexity of such an approach is exponential, but if the concept bottleneck is carefully engineered, the number of necessary concepts, and/or the number of concepts in the special combination will be small, making the computation feasible.

*More importantly*, if the same enumeration approach was applied to the raw node input data a $\mathcal{I}/\mathcal{T}$ combination *may not exist*. For example, the node-level inputs for the BFS task on each step do not tell us which nodes have visited neigbours (crucial for deciding termination). Additionally, if we have larger number of input features, the brute-force approach *may not be computationally feasible* – the combinations scale exponentially with the number of node features and concepts are one way to reduce this number.

---

[2] $k$ may vary across samples

[3] $\{a..b\}$ denotes the set of integers from $a$ to $b$

[4] In our case this broke all ties, but, if necessary, one can add tie-break on truth assignment, by e.g. choosing assignments with more true/false values

## Experimental Setup

**Algorithms considered**   We apply our GNN to the following algorithms: *breadth-first search* (BFS), *parallel coloring* (Jones and Plassmann 1993), a graph coloring heuristic, and *Kruskal's minimum spanning tree* (MST) algorithm (Kruskal 1956). BFS is modelled as a binary classification problem where we predict whether a node is visited or not, parallel coloring – as a classification task over the classes of possible node colors, plus one class for uncolored nodes. Kruskal's is modelled as two tasks trained in parallel – one, as a classification task to choose the next edge to be considered for the MST and one to help us decide which nodes belong to the same set. As the original Kruskal's algorithm executes for $|E|$ steps, *we do not learn termination for the MST task* and stick to a fixed number of steps. We show how we model inputs/outputs for all algorithms in Appendix C.

Importantly, all algorithms we experimented with posses the following two properties: (i) node/edge outputs are discrete and can be described in terms of concepts; (ii) continuing the execution can be reduced to the existence of a node with a specific combination of features. Examples of classical algorithms that *do not* fall into this category are the class of shortest path algorithms: to explain such algorithms, we would need to use arithmetic (e.g. minimum, sum) for the rules – something that concepts cannot directly capture. We leave explanation of such algorithms for future work.

To generate our concepts, we took into account what properties of the nodes/neighbourhood the algorithm uses, but we did not provide any details to the model *how* to use them. Table 1 gives more details on what concepts we chose and some example explanations. We give an outline how one can use these concepts for explaining the algorithms, in Appendix D.

**Data generation**   Following prior research on the topic of neural execution (Veličković et al. 2020b), for BFS we generate graphs from a number of categories – ladder, grid, Erdős-Rényi (Erdős and Rényi 1960), Barabási-Albert (Albert and

Barabási 2002), 4-Community graphs, 4-Caveman graphs and trees. For the coloring task, we limit the number of colors to 5 and then generate graphs where *all* nodes have fixed degree 5. This made the task both challenging (i.e. there are occassions where 5 colors are necessary) and feasible (we can generate graphs that are 5-colorable). Training data for these tasks graph size is fixed at 20 and we test with graph sizes of 20, 50 and 100 nodes. For Kruskal's algorithm, we reused most graph categories for the BFS task, except the last three where the graph is either a tree or is not connected. Due to GPU memory constraints, training MST on graphs of size 20 required reducing the batch size by a factor of 4 and making the training very time consuming. Therefore, for the MST task we reduced the size of the training graphs to 8. Testing is still performed on graphs of size 20, 50 and 100. For all tasks we did a 10:1:1 train:validation:testing split.[5] The data for each task is generated by the corresponding deterministic algorithm. More details about the data generation are present in Appendix E.

**Architectures tested** We decided to choose message-passing neural networks (Gilmer et al. 2017) with the max aggregator for the main skeleton of our processor (GNN) architecture as this type of GNN is known to align well with algorithmic execution (Veličković et al. 2020b; Georgiev and Liò 2020; Veličković et al. 2020a). However, due to the non-linear nature of some of the tasks (parallel coloring) and the added concept bottleneck we found it beneficial to add a hidden layer to some of the encoders and decoders, rather than simply model them as an affine projection.

The Kruskal's algorithm consists of several steps – masking out visited edges, finding the minimal edge from the unmasked and checking if two nodes are in the same set and unifying if they are not. The architecture for this algorithm, follows the main ideas of Figure 1, to implement them we combine the architecture of Yan et al. (2020) for the first two steps and Veličković et al. (2020a) for the third step. More details can be found in Appendix F.

As we extend on the models in (Veličković et al. 2020b,a) we use these models as baselines.

**Experimental details** We train our models using teacher forcing (Williams and Zipser 1989) for a fixed number of epochs (500 for BFS, 3000 for the parallel coloring, 100 for Kruskal's). When testing BFS/parallel coloring, we pick the model with the lowest sum of validation losses and when testing Kruskal's – the model with highest last-step accuracy. For training we use Adam optimizer (Kingma and Ba 2015) with initial learning rate of 0.001 and batch size 32. We optimise the sum of losses on the concept, output and termination (except for Kruskal's, see above) predictions – for more details on how we define our losses see Appendix G. We evaluate the ability to strongly generalise on graphs with sizes 50 and 100. Standard deviations are obtained over 5 runs. For parallel coloring we add L1 regularisation and pruning on epoch 2000 to obtain higher quality explanations since every combination of (concepts, output) pair may not

---

[5]When working with multiple graph categories, the ratio is preserved across each category

| Metric | $|V| = 20$ | $|V| = 50$ | $|V| = 100$ |
|---|---|---|---|
| MSA | $99.09_{\pm0.86}\%$ | $98.74_{\pm0.44}\%$ | $97.92_{\pm1.50}\%$ |
| LSA | $99.25_{\pm0.56}\%$ | $99.17_{\pm0.20}\%$ | $99.13_{\pm0.29}\%$ |
| TA | $98.79_{\pm0.86}\%$ | $96.79_{\pm1.53}\%$ | $95.08_{\pm2.89}\%$ |
| MSA | $99.71_{\pm0.11}\%$ | $99.23_{\pm0.21}\%$ | $98.92_{\pm0.59}\%$ |
| LSA | $99.69_{\pm0.13}\%$ | $99.17_{\pm0.23}\%$ | $99.10_{\pm0.22}\%$ |
| TA | $99.61_{\pm0.18}\%$ | $99.02_{\pm0.43}\%$ | $98.59_{\pm0.77}\%$ |
| F MSA | $99.71_{\pm0.12}\%$ | $99.24_{\pm0.21}\%$ | $98.93_{\pm0.59}\%$ |
| F LSA | $99.69_{\pm0.13}\%$ | $99.16_{\pm0.22}\%$ | $99.08_{\pm0.19}\%$ |
| F TA | $99.51_{\pm0.17}\%$ | $99.02_{\pm0.43}\%$ | $98.48_{\pm0.74}\%$ |
| *C MSA | $99.85_{\pm0.05}\%$ | $99.60_{\pm0.10}\%$ | $99.45_{\pm0.29}\%$ |
| *C LSA | $99.72_{\pm0.07}\%$ | $99.35_{\pm0.23}\%$ | $99.42_{\pm0.09}\%$ |

Table 2: Parallel coloring accuracies over 5 runs – standard (above the line) versus bottlenecked model (below the line). *MSA* is mean-step accuracy, *LSA* is last-step accuracy, *F* denotes formula-based, *C* denotes concepts' metrics

be observed during training. Libraries, code, and computing details are described in Appendix L. All hyperparameters were tuned manually.

**Metrics** We use a variety of metrics, such as *mean-step accuracy* (average accuracy of per-step outputs), *last-step accuracy* (average accuracy of final algorithm outputs) and *termination accuracy* (average accuracy of predicting termination). Similarly, from the predicted and ground-truth concepts we compute: *concepts mean-step accuracy* and *concepts last-step accuracy* as well *formula mean-step accuracy*, *formula last-step accuracy* and *formula termination accuracy*. The last three are derived by applying the *extracted formulas* to the *predicted concepts* for predicting the output/termination instead of using the respective neural network. The motivation behind is that if we achieve high concept accuracies and high formula accuracies then the formulas are likely to be representing the underlying algorithm (or data) accurately.

**Qualitative analysis** We provide several qualitative experiments: (i) We fit a decision tree (DT) for the $\mathcal{C} \to \mathcal{O}$ task ($\mathcal{C} \to \mathcal{T}$ is not possible, due to DTs working on fixed size node-level features). Concepts and targets are obtained from the *ground truth* concepts and target classes of *all training data nodes* at each step for each graph. (ii) We also plot the concepts last/mean step accuracy vs epoch for each concept and provide further analysis on which concept the networks find the most difficult. (iii) We provide sample target class explanations for each algorithm.

**Code and data** All our training and data generation code is available at https://github.com/HekpoMaH/algorithmic-concepts-reasoning.

## Results and Discussion

**Concept accuracies** As can be seen from Tables 2&3 and Table 6, Appendix H (metrics with an asterisk) we are able to learn concepts with high accuracy (99% and higher accuracy for BFS and parallel coloring). Results show that GNNs are capable of producing high-level concepts, capturing either
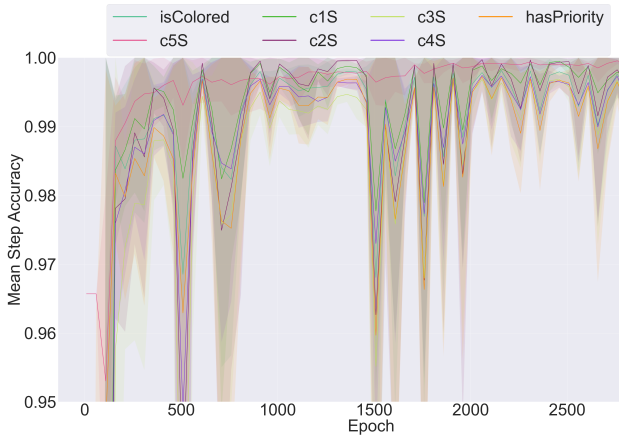
(a) Concept mean-step accuracy
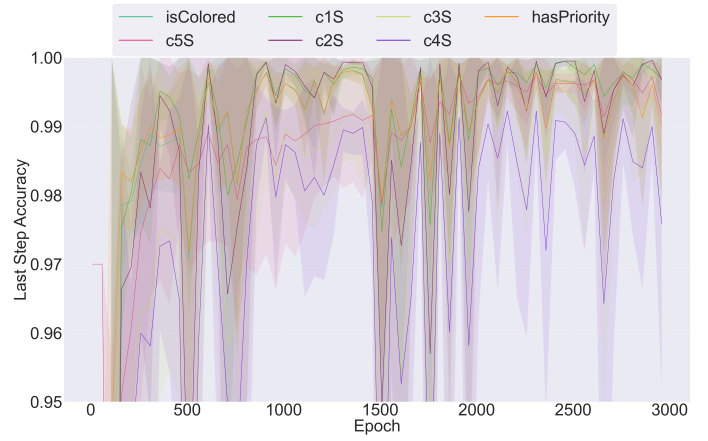
(b) Concept last-step accuracy

Figure 2: Concept accuracies per epoch of the parallel coloring algorithm. (1 point every 50 epochs). cXS is $colorXSeen$. Note the y axis scale. It can be observed that the $hasPriority$ concept is one of the worst performing concepts. This leads to nodes being colored in a different order and therefore lower last-step concept accuracy for concepts related to colors. Standard deviation obtained from 5 runs.

| Metric | $|V| = 20$ | $|V| = 50$ | $|V| = 100$ |
|--------|-----------|-----------|------------|
| MSA | $96.75_{\pm 0.15}\%$ | $95.41_{\pm 0.09}\%$ | $94.68_{\pm 0.10}\%$ |
| LSA | $93.70_{\pm 0.33}\%$ | $90.10_{\pm 2.80}\%$ | $86.69_{\pm 4.28}\%$ |
| MSA | $96.93_{\pm 0.13}\%$ | $95.86_{\pm 0.37}\%$ | $95.27_{\pm 0.59}\%$ |
| LSA. | $94.00_{\pm 0.24}\%$ | $92.20_{\pm 0.52}\%$ | $91.29_{\pm 0.86}\%$ |
| F MSA | $96.79_{\pm 0.37}\%$ | $95.77_{\pm 0.31}\%$ | $95.25_{\pm 0.54}\%$ |
| F LSA. | $93.70_{\pm 0.71}\%$ | $91.92_{\pm 0.47}\%$ | $91.15_{\pm 0.60}\%$ |
| *C MSA | $97.91_{\pm 0.08}\%$ | $97.21_{\pm 0.22}\%$ | $96.80_{\pm 0.35}\%$ |
| *C LSA. | $99.56_{\pm 0.29}\%$ | $99.49_{\pm 0.49}\%$ | $97.09_{\pm 0.29}\%$ |

Table 3: Kruskal's algorithm accuracies over 5 runs – standard (above the line) versus bottlenecked model (below the line). Abbreviation definitions same as Table 2

node or neighbourhood information, for these algorithmic tasks and the learned concept extractors strongly generalise – concept accuracy does not drop even for $5\times$ larger graphs.

**Parallel algorithms: BFS and coloring** For the BFS task both the baseline and bottlenecked model perform optimally in line with the state of the art. We therefore present results from the BFS task in Appendix H. Results from the parallel coloring task are shown in Table 2. Apart from the high accuracy achieved, our results show that: (i) the bottleneck doesn't have a major impact on the final model accuracy – original metrics[6] remain the same or are better for both algorithms; (ii) we are able to learn concepts accurately and (iii) the extracted rules are accurate – applying them to the accurately predicted concepts in order to produce output has no significant negative effect on the predictive accuracy of our model – formula based accuracies do not deviate more than 5-6% than the original metrics.

---

[6]namely mean-, last-step accuracy and termination accuracy

**Qualitative analysis: decision trees** We visualise the fitted decision trees (DTs) for each algorithm in Appendix I. In all cases the logic of the DT follows the logic of the original algorithm. Additionally, the leaf nodes of all decision trees contain samples from a single class showing that concepts were capable of capturing the complexity of the algorithm.

**Qualitative analysis: concept learning curves** We present per concept learning curves for the parallel coloring in Figure 2 and for Kruskal's in Figure 3: (i) Parallel coloring exhibits many occasions where there are drops of concept accuracy across almost all concepts. If we observe more carefully Figure 2a, we will notice that they coincide with a drop of the accuracy of $hasPriority$ concept. This drop also explains the lower last-step concept accuracy – changing the coloring order early on may produce quite different final coloring. To confirm this observations, we trained an oracle model that is always provided with the correct value for $hasPriority$. Such oracle model achieved almost perfect concept accuracy – we provide a plot of the concept learning curves in Appendix J; (ii) The concept instability was present only in the beginning for Kruskal's, but it converged to a stable solution. The reason $edgeInMst$ concept remained with the lowest last-step accuracy is that the overall last-step accuracy of the model is lower.

**Qualitative analysis: explanations** We list examples of obtained explanations in Table 4 and present *all* explanations obtained from the algorithms in in Appendix K. The extracted rules show that concepts are one way to extract accurate representation of the rules of the algorithm. E.g. we can (re)infer from the listed rules for parallel coloring that for getting a given color that color should not be seen in the neighbourhood and colors coming before that one have already been seen.

We additionally observed, that as the number of concepts

| Algorithm | Thing to explain | Explanation |
|---|---|---|
| BFS | $n$ is visited | $hasVisitedNeighbours(n)$ |
| | continue execution | $\exists n.\neg hasBeenVisited(n) \wedge hasVisitedNeighbours(n)$ |
| parallel coloring | $n$ has color 2 | $(isColored(n) \wedge \neg hasPriority(n) \wedge c1S(n) \wedge \neg c2S(n))\vee$ <br> $(hasPriority(n) \wedge c1S(n) \wedge \neg c2S(n) \wedge \neg isColored(n))$ |
| | $n$ has color 5 | $(isColored(n) \wedge \neg hasPriority(n) \wedge c1S(n)$ <br> $\wedge c2S(n) \wedge c3S(n) \wedge c4S(n))\vee$ <br> $(hasPriority(n) \wedge c1S(n) \wedge c2S(n)$ <br> $\wedge c3S(n) \wedge c4S(n) \wedge \neg isColored(n))$ |
| | continue execution | $\exists n.\neg isColored(n)$ |
| Kruskal's | $e$ not in MST | $(nISS(e) \wedge \neg eIM(e)) \vee (\neg lEV(e) \wedge \neg eIM(e))$ |
| | $e$ in MST | $(lEV(e) \wedge nISS(e) \wedge eIM(e)) \vee (lEV(e) \wedge \neg nISS(e) \wedge \neg eIM(e))$ |

Table 4: Sample explanations for each algorithm *obtained from the learned model. $cXS$* denotes $colorXSeen$, $nISS$ is $nodesInSameSet$, $lEV$ is $lighterEdgesVisited$, $eIM$ is $edgeInMst$.



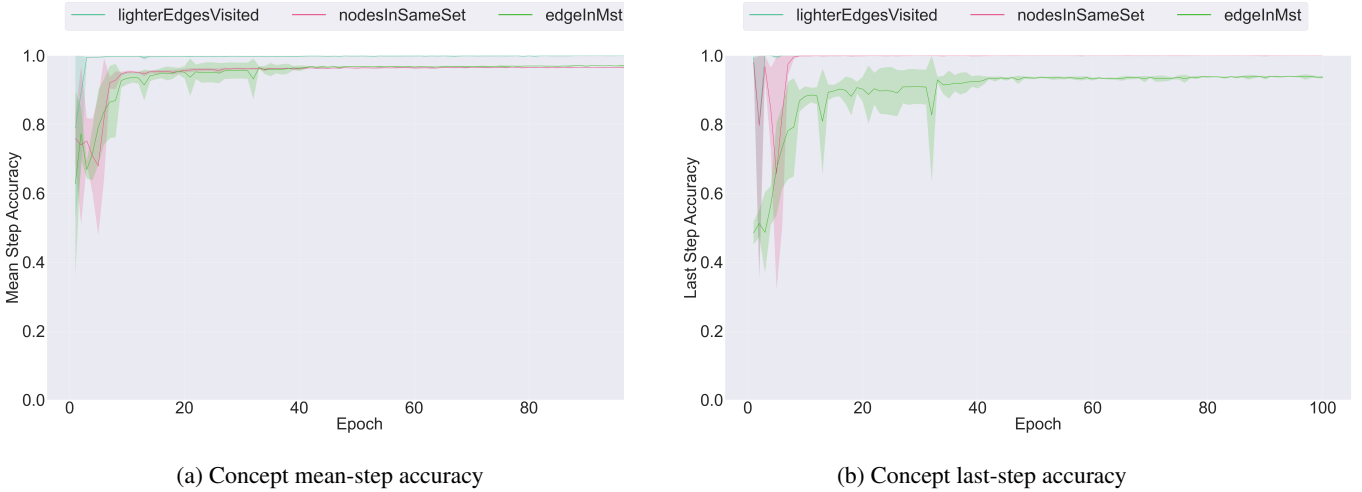(a) Concept mean-step accuracy



(b) Concept last-step accuracy

Figure 3: Concept accuracies per epoch of the Kruskal's algorihm on graphs with 20 nodes. (1 point per epoch). After an initial instability concepts are consistently accurate. Standard deviation obtained from 5 runs.

increases, if we need shorter and more general rules we need more and more data. One way to alleviate such problem is L1 regularisation and pruning – we additionally perform an ablation study in Appendix K showing that without regularisation rules are still usable (giving good formula accuracy) *but are less general.*

## Conclusions

We presented concept-based reasoning on graph algorithms through Concept Bottleneck Graph Neural Networks. We demonstrated through the surveyed algorithms, that we can accurately learn node-level concepts without impacting performance. Moreover, by examining training data and model weights, we are capable of explaining each node-level output classes with formulas based on the defined concepts. Concepts also allow us perform a *unsupervised* rule extraction of certain graph-level tasks, such as deciding when to terminate.

Extracted rules are interpretable and applying them does not heavily impact accuracy.

## Broader Impact

Our work evaluates the extent to which concept bottlenecks can be applied to GNNs for explaining algorithms and hence has no specific ethical risks associated. However, our approach is not restricted only to algorithms and can be applied to other graph domains. GNNs have already seen a lot of success on various real-world settings including, but not limited to: bioinformatics/computational medicine, social networking applications, physics and quantum chemistry simulations. Applying CBGNNs on each of the aforementioned domains carries the ethical risks present in the domain.

6691

# References

Adadi, A.; and Berrada, M. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access*, 6: 52138–52160.

Albert, R.; and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1): 47.

Baldassarre, F.; and Azizpour, H. 2019. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*.

Barbiero, P.; Ciravegna, G.; Georgiev, D.; and Giannini, F. 2021. PyTorch, Explain! A Python library for Logic Explained Networks. ArXiv preprint, arXiv:2105.11697.

Battaglia, P. W.; Pascanu, R.; Lai, M.; Rezende, D. J.; and Kavukcuoglu, K. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In Lee, D. D.; Sugiyama, M.; von Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 4502–4510.

Cappart, Q.; Chételat, D.; Khalil, E.; Lodi, A.; Morris, C.; and Veličković, P. 2021. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*.

Chen, Z.; Bei, Y.; and Rudin, C. 2020. Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, 2(12): 772–782.

Ciravegna, G.; Giannini, F.; Melacci, S.; Maggini, M.; and Gori, M. 2020. A Constraint-Based Approach to Learning and Explanation. In *AAAI*, 3658–3665.

Deac, A.; Velickovic, P.; Milinkovic, O.; Bacon, P.; Tang, J.; and Nikolic, M. 2020. XLVIN: eXecuted Latent Value Iteration Nets. *CoRR*, abs/2010.13146.

Doshi-Velez, F.; and Kim, B. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

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

Georgiev, D.; and Liò, P. 2020. Neural Bipartite Matching. In *Graph Representation Learning and Beyond (GRL+) workshop*.

Ghorbani, A.; Wexler, J.; Zou, J.; and Kim, B. 2019. Towards automatic concept-based explanations. *arXiv preprint arXiv:1902.03129*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 1263–1272.

Gori, M. 2017. *Machine Learning: A constraint-based approach*. Morgan Kaufmann.

Hamrick, J. B.; Allen, K. R.; Bapst, V.; Zhu, T.; McKee, K. R.; Tenenbaum, J.; and Battaglia, P. W. 2018. Relational inductive bias for physical construction in humans and machines. In *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*.

Jones, M.; and Plassmann, P. 1993. A Parallel Graph Coloring Heuristic. *SIAM J. Sci. Comput.*, 14: 654–669.

Joshi, C. K.; Cappart, Q.; Rousseau, L.; Laurent, T.; and Bresson, X. 2020. Learning TSP Requires Rethinking Generalization. *CoRR*, abs/2006.07054.

Kazhdan, D.; Dimanov, B.; Jamnik, M.; and Liò, P. 2020a. MEME: Generating RNN Model Explanations via Model Extraction. *arXiv preprint arXiv:2012.06954*.

Kazhdan, D.; Dimanov, B.; Jamnik, M.; Liò, P.; and Weller, A. 2020b. Now You See Me (CME): Concept-based Model Extraction. *arXiv preprint arXiv:2010.13233*.

Kazhdan, D.; Dimanov, B.; Terre, H. A.; Jamnik, M.; Liò, P.; and Weller, A. 2021. Is Disentanglement all you need? Comparing Concept-based & Disentanglement Approaches. *arXiv preprint arXiv:2104.06917*.

Kingma, D. P.; and Ba, J. 2015. 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*.

Koh, P. W.; Nguyen, T.; Tang, Y. S.; Mussmann, S.; Pierson, E.; Kim, B.; and Liang, P. 2020. Concept Bottleneck Models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 5338–5348. PMLR.

Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1): 48–50.

Luo, D.; Cheng, W.; Xu, D.; Yu, W.; Zong, B.; Chen, H.; and Zhang, X. 2020. Parameterized explainer for graph neural network. *arXiv preprint arXiv:2011.04573*.

Mendelson, E. 2009. *Introduction to mathematical logic*. CRC press.

Molnar, C. 2020. *Interpretable machine learning*. Lulu. com.

Moore, E. F. 1959. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, 285–292.

Pal, A.; Eksombatchai, C.; Zhou, Y.; Zhao, B.; Rosenberg, C.; and Leskovec, J. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In Gupta, R.; Liu, Y.; Tang, J.; and Prakash, B. A., eds., *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 2311–2320. ACM.

Pope, P. E.; Kolouri, S.; Rostami, M.; Martin, C. E.; and Hoffmann, H. 2019. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10772–10781.

Schnake, T.; Eberle, O.; Lederer, J.; Nakajima, S.; Schütt, K.; Müller, K.; and Montavon, G. 2020. Higher-order explanations of graph neural networks via relevant walks. *arXiv: 2006.03589*.

Shanahan, M.; Nikiforou, K.; Creswell, A.; Kaplanis, C.; Barrett, D. G. T.; and Garnelo, M. 2020. An Explicitly Relational Neural Network Architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 8593–8603. PMLR.

Stokes, J. M.; Yang, K.; Swanson, K.; Jin, W.; Cubillos-Ruiz, A.; Donghia, N. M.; MacNair, C. R.; French, S.; Carfrae, L. A.; Bloom-Ackermann, Z.; Tran, V. M.; Chiappino-Pepe, A.; Badran, A. H.; Andrews, I. W.; Chory, E. J.; Church, G. M.; Brown, E. D.; Jaakkola, T. S.; Barzilay, R.; and Collins, J. J. 2020. A Deep Learning Approach to Antibiotic Discovery. *Cell*, 180(4): 688–702.e13.

Strathmann, H.; Barekatain, M.; Blundell, C.; and Veličković, P. 2021. Persistent Message Passing. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*.

Tang, H.; Huang, Z.; Gu, J.; Lu, B.-L.; and Su, H. 2020. Towards Scale-Invariant Graph-related Problem Solving by Iterative Homogeneous GNNs. *Advances in Neural Information Processing Systems*, 33.

Veličković, P.; Buesing, L.; Overlan, M. C.; Pascanu, R.; Vinyals, O.; and Blundell, C. 2020a. Pointer Graph Networks. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Veličković, P.; Ying, R.; Padovano, M.; Hadsell, R.; and Blundell, C. 2020b. Neural Execution of Graph Algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Vu, M. N.; and Thai, M. T. 2020. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *arXiv preprint arXiv:2010.05788*.

Wijaya, M. A.; Kazhdan, D.; Dimanov, B.; and Jamnik, M. 2021. Failing Conceptually: Concept-Based Explanations of Dataset Shift. *arXiv preprint arXiv:2104.08952*.

Williams, R. J.; and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2): 270–280.

Xu, K.; Li, J.; Zhang, M.; Du, S. S.; Kawarabayashi, K.; and Jegelka, S. 2020. What Can Neural Networks Reason About? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yan, Y.; Swersky, K.; Koutra, D.; Ranganathan, P.; and Hashemi, M. 2020. Neural Execution Engines: Learning to Execute Subroutines. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Yeh, C.-K.; Kim, B.; Arik, S. O.; Li, C.-L.; Pfister, T.; and Ravikumar, P. 2019. On completeness-aware concept-based explanations in deep neural networks. *arXiv preprint arXiv:1910.07969*.

Ying, R.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32: 9240.