Explaining Graph Neural Networks with Mixed-Integer Programming

Anonymous CVPR submission

Paper ID 16514

Abstract

Graph Neural Networks (GNNs) have become a popular tool for learning from graph-structured data. While they provide state-of-the-art graph learning performance, their lack of transparency hinders our ability to trust and understand them, ultimately limiting the areas where they can be applied. Many explanation methods exist to explain individual predictions made by GNNs, but there are fewer ways to gain more general insight into the patterns they have been trained to identify. Most proposed methods for model-level GNN explanations attempt to generate graphs that exemplify these patterns, but the discreteness of graphs and the nonlinearity of deep GNNs make finding such graphs difficult. In this paper, we formulate the search for an explanatory graph as a mixed-integer programming (MIP) problem, in which decision variables specify the explanation graph and the objective function represents the quality of the graph as an explanation for a GNN's predictions of an entire class in the dataset. This approach, which we call MIPExplainer, allows us to directly optimize over the discrete input space and find globally optimal solutions with a minimal number of hyperparameters. MIPExplainer is able to find accurate explanations on both synthetic and real-world datasets, and outperforms existing methods for generating explanations for GNNs.

1. Introduction

Graph neural networks (GNNs), such as graph convolutional networks (GCN) [12], GraphSAGE [8], and graph attention networks (GAT) [26], provide a family of powerful tools for modelling graphs that learn from both the features contained in nodes and edges and the structure of the graph itself. While they have achieved state-of-the-art performance across a wide range of application domains, their applications are still limited by our inability to explain their behavior. Without being able to explain the patterns GNNs rely on to make predictions, it is impossible to justify their use in applications where trust and safety are important, and there is no way to extract useful information from them. This has motivated a significant amount of research into techniques for GNN explainability.

Although there is research into inherently explainable deep learning, such as ProtGNN [35] and GIB [31] for GNNs, this approach generally comes at the cost of performance. Therefore, most explanation methods are designed to be applied post-hoc to trained models. Elucidating how a GNN "reasons" on graphs presents a completely new set of challenges from deep learning explainability over image and text data. Encodings of graphs lack the same spatial information, and their structure must be represented explicitly, for example by adjacency matrices or edge lists. These representations are inherently discrete, which violates assumptions necessary for the application of commonly-used gradient based methods [5, 19]. The outputs of GNNs are further expected to be permutation equivariant with respect to the ordering of the nodes, which further imposes constraints on generating explanation graphs for a GNN model. A one-size-fits-all strategy may not work with the wide variety of existing graph types (e.g., directed and undirected graphs, graphs with node and edge features) and GNN layers (e.g., convolutions in GCNs, GATs, GINs). Although it can be further extended, our method will focus on unweighted graphs and GraphSAGE-style graph convolutions.

1.1. Related Work

GNN interpretation has been largely focused on instance-level explanation which aims to explain the reasoning behind individual predictions. As identified in [34], at least six categories of instance-level GNN explanation methods have been proposed so far: gradient-based [20], perturbation-based [18, 23, 30, 33], surrogate [27], generation-based [13], decomposition [24], and counterfactual-based [17] methods. These methods do not immediately provide insights into the overall patterns a GNN has identified to make predictions, but it is possible to consolidate instance-level explanations to reveal model-level patterns. For example, we can employ purely statistical methods (e.g., a rank-sum test) to determine whether there are nodes/edges shared by a significant portion of the explanations. A more recent technique is GLGExplainer [2], which finds smaller com-

ponents of the extracted explanations that can be used to build logical expressions consistent with the GNN's predictions (e.g. the GNN predicts a class of graphs given the presence of a certain subgraph and the absence of another subgraph). However, these methods are limited by the scope of the training data, and can be influenced by bias in the dataset. A direct model-level explanation can offer more faithful explanations, and is more useful for determining the degree of bias in the model itself. In this work, we focus on post-hoc model-level explainability for GNNs.

Relatively few methods exist to explain GNNs at the model level. XGNN [32] is the most widely used, and serves as the only baseline in several recent papers that focus on similar objectives [2, 21, 25, 28]. These methods aim to generate graphs that exemplify graph structures used by a trained GNN for making classifications, without straying too far from the distribution of the training data where the model is not well-defined. XGNN involves training a second neural network by reinforcement learning to generate graphs that maximize the original GNN's prediction for a specific class. OrphicX [15] and GEM [14] are two other methods that involve training a second neural network to generate explanations, but they use different objective functions designed to encourage causal explanations. The use of a second black box model in these methods reduces their reliability and makes interpreting the generated explanations significantly more difficult.

Other recent methods avoid training a second neural network in various ways. GNNInterpreter [28] and GraphEx [22] assume that the graphs in the dataset are sampled from a set of underlying distributions parameterized by continuous latent parameters. GNNInterpreter defines an objective function similar to XGNN during training, maximizing a target class's logit while penalizing the distance between the embedding of the generated graph and the mean embedding of the training data to keep explanations in-distribution, and learns parameters through Monte Carlo gradient estimation. GraphEx attempts to solve this problem using maximum likelihood estimation, which does not require gradients from the model and only relies on its predictions of graphs in the dataset. PAGE [25] also maximizes predicted class probability, but restricts explanations to common subgraphs in the target class.

In the past, discrete optimization techniques have been applied to deep neural networks to solve inverse problems, such as in [3] and [1]. Finding a graph that maximizes the GNN output is a type of inverse problem. However, these existing methods were only defined for layers consisting of a linear transformation with ReLU activations, and rely on costly automated bound-tightening procedures to achieve feasible runtimes. This work will reformulate the optimization problem for explainability, generalize the application of mixed-integer programming on standard neural networks to a range of GNN architectures, and derive explicit bounds for continuous variables that can be calculated with a single forward pass through the network.

1.2. Current Issues and Our Contributions

There are several common problems among existing approaches for model-level GNN explanation. For one, they all have many hyperparameters that can change the generated explanation graphs. Generating a high-quality explanation requires specific settings for these hyperparameters, but the more times a user runs the same algorithm with different hyperparameters, the more likely they are to generate spurious explanations. In cases where hyperparameters modify the objective function used to calculate the quality of explanations (e.g. weights for regularization terms), it is impossible to quantitatively compare the quality of the generated explanations, even though they may be extremely different. This problem is exasperated by the fact that several of these methods rely on stochastic gradient optimization to optimize their explanations. Due to the stochastic nature of this approach and the necessity of setting a maximum number of iterations, the objective value might be far from the global optimum when an explanation graph is generated, without any way of knowing that this is the case. Thus, the corresponding explanation may not represent the information that the objective function was designed to extract.

In this paper, we propose a new explanation method based on mixed-integer programming (MIP) for finding class-representative input graphs. This has several benefits over existing approaches. (1) Unlike any existing method, we can directly optimize over the discrete space of possible input graphs, without any restrictions on types of node and edge features. The only assumption we make about the space of graphs is a bounded size, and we do not require any assumptions about its underlying distribution. (2) This approach has a minimal number of hyperparameters that influence the explanation (only the size of the explanation must be specified), facilitating the application of our approach and mitigating the effects of bias when analyzing the results. (3) We prove that our MIP formulation has a globally optimal solution, and in many cases we can find and verify this solution. In cases where this is intractable, we can place an upper bound on the optimal solution, guaranteeing the quality of the generated explanation.

2. MIPExplainer

Our model-level explanation also seeks to optimize an input graph $G = (X, A)$ so that the GNN predicts a certain class with the highest possible probability, where X contains the d attributes for each node as row vectors for N nodes in the graph and $A = (a_{ij})$ represents the N by N adjacency matrix. We focus on the case of a binary adjacency matrix where $a_{ij} \in \{0, 1\}$, so that $a_{ij} = 1$ indicates there is an

edge between nodes i and j. Let a GNN realize a function $f_c(G, \theta)$ that maps G to the probabilities of several classes indexed by c and θ contains all the learned parameters in the GNN. During the GNN training, G is given and θ needs to be determined, whereas in many model explanation methods, θ has been fixed, and we optimize G (i.e., X and A) to maximize $f_c(G, \theta)$ (or a related objective).

The proposed MIP will optimize G in terms of the values of A and X. Each layer of the GNN imposes a set of constraints in the MIP. We add decision variables to represent the output of each layer, which are constrained to the correct values in terms of the decision variables representing the output of the previous layer. For example, for a fully connected layer, a new matrix of decision variables Y' will be added to the model and constrained with $Y' = WY' + b$, where Y' are the decision variables representing the outputs of a previous layer and W and b are the model's learned parameters. Since the outputs of subsequent layers are constrained exactly, ultimately all of the constraints define the feasible region of X and A. We place constraints on nodes and edges (or entries of A) so that the derived explanation forms a connected graph with valid features, and we can further constrain X and A to reduce the number of candidate solutions for a single graph since the GNN is permutation equivariant with respect to the order of nodes. In the subsequent sections, we provide a detailed derivation of our MIP formulation by discussing the objective function and the various constraints.

2.1. Objective Function

Following the paradigm established by existing methods, an objective function for explanations typically contains two parts: a term related to class prediction and a regularizer that enforces the generated explanations to be in-distribution. In XGNN, the explanation generator is penalized during training for actions that violate manually-defined sets of rules, such as the maximum number of bonds that can be formed with a certain atom in a molecule. In GNNInterpreter, the embedding of the explanation graph needs to be close to the average embedding of graphs in the training set. While these regularization strategies may help confine the explanation graph to a region of the input space where the model is well-defined, they cannot guarantee the quality of the explanation. While regularization terms can normally be balanced through some tuning procedure, this is impossible without knowing the ground-truth explanations for the GNN already, and attempting to determine the weights by judging the generated graphs qualitatively increases the likelihood of mistakenly accepting spurious explanations. Therefore, we do not apply any regularization in the objective function during our experiments, but the proposed method is able to incorporate commonly used regularizers if desired.

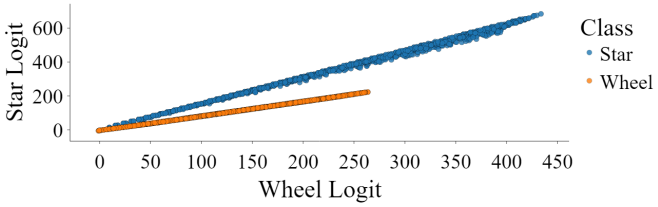

Figure 1. Logits for the Star and Wheel Graphs in the Shapes Dataset

While maximizing a single logit while disregarding the logits of other classes in the denominator (e.g., as done by GNNInterpreter) is possible, this may lead to low quality explanations in some circumstances. Predictions are made based on the difference between the logits, and the absolute value of a single logit may be unrelated to the prediction of the network. To illustrate this, after training a GNN to classify star graphs and wheel graphs of varying sizes (a task defined in [28]), we plotted the logits it assigned to the training data for both classes in Figure 1. Note that the maximum logit for the wheel class is actually assigned to a correctly-classified star graph. Thus, simply maximizing the logit for wheels will not produce an effective explanation for the wheel class.

To accurately find class-discriminative information, we should maximize the *difference* between the logit of the target class and the logit of the other classes. Maximizing the normalized probability, as done by XGNN, is possible but can lead to numerical instability when maximizing because improvements get exponentially smaller as the magnitude of the logits increases. We can form an objective function as a linear combination of all logits but with a positive coefficient only for the target class, but at the risk of a single negative logit dominating the objective value. To mitigate this problem, we can maximize the difference between the logit of the target class and the maximum of the other classes. In our observation, the latter is more effective, so we focus on discussing the following objective function:

$$\max_{G} \left(f_c(G, \theta) - \max_{i \neq c}(f_i(G, \theta)) \right), \tag{1}$$

where f_i denotes the ith output of the GNN before the application of the softmax function for classification.

2.2. Constraints

We make one crucial assumption about the node features, that their values are bounded by a constant M. We do not make assumptions on the node features or their distribution. We also require that the number of nodes in the explanation n is fixed in advance, which is the only hyperparameter that changes the optimization problem being solved.

If GraphSAGE convolution layers are employed in the GNN, the updated node representations X' after each layer are calculated from existing node representations X with the formula

$$X' = \sigma(XW_1 + \text{Aggregation}(X)W_2 + b) \quad (2)$$

where the aggregation is commonly realized by summing over neighbors' attributes, i.e., $\text{Aggregation}(X) = AX$.

Assume that a GNN model has L_c GraphSAGE-based convolution layers with sum aggregations and ReLU activations, followed by a global feature-wise sum pooling layer and L_f fully connected (FC) layers with ReLU activations. In total, there are $L = L_c + 1 + L_f$ layers (indexed as $L_i, i \in 1, \ldots, L$). We will use the following notation: $W_j^{(i)}$, matrices of scalars, denote the GNN's matrices of learned weights in layer i. The vectors of scalars $b^{(i)}$ denote the GNN's learned bias vectors in layer i. For notation convenience, we also denote $X^{(0)} = X$, where x_{ij} is the jth feature of node i.

We will also add the following decision variables to our formulation, which will be constrained to the correct values based on X and A: $\Phi^{(i)}$ represents the output of layer i before the activation function, $X^{(i)}$ represents $ReLU(\Phi^{(i)})$, the output of layer i. We also represent $ReLU(-\Phi^{(i)})$ by $B^{(i)}$, while $Z^{(i)}$ are binary indicators representing the truth value of $\Phi^{(i)} > 0$ elementwise[1]. d is an indicator vector representing whether each element of $\Phi^{(L)}$ is the maximum element in the output of layer L not including the target class (i.e., dimension j of d is 1 when dimension j of $\Phi^{(L)}$ is the maximum, while the rest are all 0's), and y represents the value of the maximum output of the GNN that is not for the target class. Aside from the binary variables A, $Z^{(i)}$, and d, all other variables are continuous.

To constrain $\Phi^{(i)}$ for the convolutional layers ($1 \leq i \leq L_c$):

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + AX^{(i-1)}W_2^{(i)} + b^{(i)}. \quad (3)$$

For the pooling layer $i = L_c + 1$ (with $\mathbf{1}$ representing a vector of 1s):

$$\Phi^{(i)} = \mathbf{1}^T\Phi^{(i-1)}, \quad (4)$$

and for the fully connected layers ($L_c + 1 < i \leq L$):

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + b^{(i)}. \quad (5)$$

To constrain $X^{(i)}$ for all layers except the pooling and readout layers ($0 < i \leq L - 1, i \neq L_c + 1$), we encode the ReLU output as follows:

$$X^{(i)} - B^{(i)} = \Phi^{(i)}, \quad (6)$$
$$X^{(i)} \leq MZ^{(i)}, \quad (7)$$
$$B^{(i)} \leq M(1 - Z^{(i)}), \quad (8)$$
$$0 \leq X^{(i)}, B^{(i)} \leq M. \quad (9)$$

For the pooling layer, we simply have that $X^{(L_c+1)} = \Phi^{(L_c+1)}$. To constrain d_j and y:

$$y \geq X_{\neq c}^{(L)}, \quad (10)$$
$$y \leq X_{\neq c}^{(L)} + (\max(U_{X_{\neq c}^{(L)}})\mathbf{1} - L_{X_{\neq c}^{(L)}})(1 - d), \quad (11)$$
$$\sum_j d_j = 1, d_j \in \{0, 1\}, \quad (12)$$

where $L_{X_{\neq c}^{(L)}}$ and $U_{X_{\neq c}^{(L)}}$ represent lower and upper bounds for the decision variables in $X^{(L)}$ excluding the output of class c. A method to calculate these bounds based on the bounds of the input will be discussed in a later section. Most of these constraints are linear in terms of the decision variables except Eq.(3) where decision variables A and $X^{(i)}$ multiply to form quadratic terms. Because A is binary, these terms can be equivalently reformulated into linear functions. There are several ways to perform the linearization of quadratic terms with both continuous and binary variables. We describe one such method by change of variables [10]. For a given binary variable $a \in A$ and a continuous variable $x \in X^{(i)}$ bounded by M, let $e = a \times x$ be a new intermediate decision variable. Let $E^{(i)}$ be the matrix of $AX^{(i)}$ where entries are all calculated by summing the corresponding e's. Constraints in Eq.(3) can be rewritten as follows with additional bound constraints:

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + E^{(i)}W_2^{(i)} + b^{(i)}, \quad (13)$$
$$-Ma \leq e \leq Ma, \quad (14)$$
$$x - M(1 - a) \leq e \leq x + M(1 - a). \quad (15)$$

Now, our MIP has been transformed into a problem that maximizes Eq.(1) which can be calculated as $X_c^{(L)} - y$ subject to constraints Eq.(13-15) and Eq.(4-12). Note that this MIP has a concave objective function and all linear constraints when integrality is relaxed, so it is a mixed-integer linear program (MILP).

2.3. Constraints on A and X

Additional constraints can be placed on A and X when generating explanations. For example, when the input space is actually graphs with one-hot features, we can constrain the sum of each row of $X^{(0)}$ to be equal to 1, and X can also be defined with binary or integer decision variables when appropriate. If the input graph is undirected, we can add the

[1]For elements of $\Phi^{(i)}$ exactly equal to 0, the corresponding values of $Z^{(i)}$ can still be 0, but this will not affect the computation.

constraints $a_{ij} = a_{ji}$ for all i, j with $0 \leq ij < n$ and $i < j$. We can prevent self-loops in the explanation by constraining the diagonal elements of A to be 0.

We also impose a partial ordering on the graph nodes $\text{node}_1, \ldots, \text{node}_n$ to ensure that the explanation graph is connected, or in the case of directed graphs, weakly connected (i.e. connected ignoring the directionality of the edges). We require that there is at least one edge between node_i and the set of nodes $\{\text{node}_j \mid i > j\}$. This becomes a constraint on A, and specifically for each i with $0 \leq i < n$, we add the constraint $\sum_{\{j|i>j\}}(a_{ij} + a_{ji}) \geq 1$. These constraints also partially alleviate the effect of equivariance (where the same graph can have many different A), because they reduce the MIP's feasible region, but not the set of candidate graphs. This can easily be proven by showing that for any (weakly) connected graph, the nodes can be ordered in a way that satisfies these constraints by running depth-first-search (DFS) on such a graph ignoring edge directions. Starting at an arbitrarily-chosen node. The ith node found by DFS must have been found from one of the 1 through $i-1$ nodes.

2.4. Generalizing to more GNNs

Many highly performant GNN architectures can be perfectly represented by linear and quadratic constraints, and many more can be closely approximated. For example, if we choose our aggregation function to be a feature-wise average instead of a feature-wise sum, we can simply modify constraint (4) as $\Phi^{(i)} = \mathbf{1}^T \Phi^{(i-1)} \frac{1}{N}$ for $i = L_c + 1$, . If mean aggregation is used in Eq.(2), we need another set of decision variables $D^{(i)}$ for each layer, where row j of $D^{(i)}$ will represent the feature-wise average of the neighbors of node j. To properly constrain $D^{(i)}$, we add the constraint $\mathbf{1}(\mathbf{1}^T A)D^{(i)} = AX$ to the model. The elements of D can be distributed in the multiplications for the left hand side, and then all the terms can be linearized as previously described. Now, constraint (3) can be changed to:

$$\Phi^{(i)} = X^{(i-1)} W_1^{(i)} + D^{(i)} W_2^{(i)} + \boldsymbol{b}^{(i)}$$

Consider a message passing layer from a Graph Isomorphism Network [29], where updated node representations are calculated as $X' = h((A + (1+\epsilon)I)X)$, h is a neural network, and ϵ is a constant. We can split this computation by constraining intermediate decision variables according to the inner piece, $AX + ((1+\epsilon)I)X$, and the application of the neural network to those intermediate variables, both of which we previously discussed how to express with linear constraints. Additionally, piecewise-linear approximations can be created for non-linear functions, allowing us to model different activation functions and the convolutional layers in GCNs or GATs.

2.5. Optimality

We now show that a globally optimal solution to the MIP problem described above always exists, and that it can be provably found by MIPExplainer.

Theorem 1. *Consider the MILP problem that maximizes the objective function $X_c^{(L)} - y$, with decision variables A, $X^{(i)}$, $\Phi^{(i)}$, $Z^{(i)}$, $B^{(i)}$, $E^{(i)}$, d_j, y subject to constraints (4-15). A global optimum exists for this MILP.*

Proof. Since all the constraints are linear equalities or inequalities (after linearizing the multiplication of binary and continuous variables in Eq.3), they define a polyhedral feasible region in the space parameterized by the decision variables if binary variables are relaxed to be $\in [0, 1]$. All decision variables in the MILP are bounded, either directly or in terms of bounds on the input variables A and X, so the feasible set is a closed polytope (a compact set). The objective function is linear in terms of the decision variables as calculated by $X_c^{(L)} - y$. Therefore, the linear relaxation of the MILP must have an optimum, over the compact set, that is located specifically on its boundary. In addition, there are a finite number of integer solutions within the compact feasible region, so at least one of them must have the maximum objective value. □

We can use typical methods such as branch and bound, cutting planes, and heuristics to find the optimal solution efficiently. Using branch and bound, we start by finding the optimal solution of the LP relaxation (i.e. the MIP with the integrality constraints removed) of the MIP problem, for example using the simplex method. This serves as an upper bound to the original problem with integer domains for some decision variables. If some integral decision variables take fractional values in the relaxation, we branch on one of them by partitioning the set of candidate solutions for the problem with integer constraints into 2 subproblems, one with the extra constraint that a chosen fractional variable is at most the floor of its value in the LP relaxation's optimum and another ensuring that the variable at least the ceiling of that value. The optimal solution to the integral problem will be the maximum optimal solution of these two subproblems, which can be solved recursively in the same way. On the other hand, any integer solution serves as a lower bound to the integral problem's optimum, and we can use heuristics to find increasingly better solutions as we descend in the search tree and fix more variables. If the linear relaxation solved at an internal node is infeasible or has a maximum objective value that is lower than our current lower bound, we have proven that the optimal solution to the integral problem cannot lie anywhere in the subtree rooted at that node, allowing us to skip all the nodes it contains in our search. The process stops when there are no more subproblems to explore, at which point we will have found the optimal solution to the integral problem. In our experiments, we

use Gurobi Optimizer [7] to find the optimal solution efficiently. This solver benefits from being fully deterministic, allowing us to generate explanations with a high degree of consistency.

2.6. Practical Considerations

In practice, it can be difficult to solve MILPs corresponding to large GNNs, and several techniques are needed to make the process tractable. Often, just finding an initial setting for all of the decision variables that satisfies all constraints is difficult. In our experiments, we found that this step can actually take longer than the subsequent optimization. This problem can be completely eliminated with a warm start. Starting from an arbitrary input graph (either from the dataset or not), we can compute a forward pass through the network to obtain a valid setting of initial values for almost all of the decision variables. In cases where additional constraints have been imposed on the graph, such as to ensure connectivity as described above, an input graph must be converted into the canonical form that also satisfies these constraints.

While a single, large number can be used to bound all of the continuous decision variables, tighter bounds greatly reduce the time needed to compute optimal solutions. While automated bound-tightening procedures exist, it is faster to use knowledge of the problem to bound manually. Each hidden representation computed by the model is encoded by a separate set of decision variables. Assuming we have bounded the decision variables for one, we can compute bounds for the outputs of a following transformation. For example, given a hidden representation vector x with lower bound x_L and upper bound x_U, we can bound the output of a linear layer $x' = Wx + b$ below and above by

$$x'_L = \text{ReLU}(W)x_L + \text{ReLU}(-W)x_U + b \quad (16)$$

$$x'_U = \text{ReLU}(W)x_U + \text{ReLU}(-W)x_L + b \quad (17)$$

Given bounds on the the decision variables representing the explanation graph, the input to the GNN, we can follow the propagation of values through the GNN to iteratively bound the set of decision variables for each hidden representation. Bounds for the outputs of ReLU activation layers will be the same as their inputs, but clipped below at 0. In the case of layers like GraphSAGE convolutions where the output is the sum of several matrix multiplications, bounds can be derived for each term in the sum and then added together.

Floating-point precision errors can lead to serious problems for MIQP solvers, and in cases where decision variables can take both small and large values, a significant amount of time may be needed to avoid numerical instability. This is relevant when the weights of GNNs become very small, an effect often produced by regularization. However, we found that weights below a certain threshold (we chose 10^{-5}) could be floored to zero without significantly affecting the behavior of the network. All performance metrics for the networks used in the experiments were computed after the networks were pruned in this way. We also found that smoothing networks with regularization improved solution times.

Despite these measures, runtime remains the most significant drawback of the proposed approach. In our largest experiments, we were not able to guarantee a global optimum within a single day. However, the largest reason for this runtime is the amount of symmetry in our formulation. A single graph corresponds to a number of adjacency matrices that, in the worst case, grows exponentially with the number of vertices it contains. This hinders our ability to tighten the upper bound while exploring the search tree, since an existing global optimum may be transformed into another as we traverse a branch. In the future, we plan to address this problem by introducing additional constraints to reduce the number of feasible adjacency matrices in each equivalence class defined by graph isomorphism. Despite the increased time needed to prove optimality, the proposed method often finds optimal solutions early in the search. Therefore, we impose a time limit during experiments to prove its practicality.

3. Empirical Evaluation

We use two synthetic datasets and one real-world dataset to evaluate our method: Is_Acyclic, Shapes, and MUTAG. The **Is_Acyclic** dataset comes from XGNN's experiments, and has two classes consisting of cyclic and acyclic graphs of various types. The cyclic graphs include graphs like grids, single cycles, and wheels, while the acyclic class includes graphs like paths and various types of trees. Every node is given the same feature, a single constant, in order to isolate the explanation methods' ability to capture structural information. For the **Shapes** dataset, which comes from GNNInterpreter's experiments, graphs are first generated from one of 5 base classes: lollipop graphs contain a fully connected component with one connection to a path graph's end node, grid graphs are lattices where each internal node has 4 neighbors, star graphs have multiple outer nodes connected to a single central node, and wheel graphs are star graphs with a single cycle connecting the outer nodes. For each of these graphs, a uniform proportion between 0 and 0.2 is chosen, and the number of edges in the graph is increased by that amount by adding in edges uniformly at random. The features of each node are the same as in Is_Acyclic. The **MUTAG** dataset [4] consists of graphs of chemical compounds, where nodes represent atoms and edges represent bonds between them. Each compound is classified as being either mutagenic or non-mutagenic. As described by the creators of this dataset and in [9], mutagenic molecules tend to have higher numbers of fused rings of carbon atoms.

	# of Graphs	# of Classes	Average # of Nodes	Average # of Edges	# of Node Features
Is_Acyclic	533	2	28.5	68.1	1
Shapes	8000	5	27.2	144.9	1
MUTAG	188	2	17.9	39.6	7

Table 1. Dataset Summary

For this dataset, each node's features are a one-hot vector indicating atom type.

3.1. Experimental Setup

Every dataset was randomly split into a training set (80%) and a test set (20%). GNNs were trained on each, and performance metrics are reported in Table 2. In all experiments, the GNNs use GraphSAGE-style convolutions with sum being used as the aggregation operator, followed by a global mean pooling layer, and finally several fully-connected (FC) layers. ReLU activations are placed between each hidden layer. For the Is_Acyclic and Shapes datasets, the GNN uses 2 convolutional layers computing 16 features per node, a FC layer computing 8 features, and a final FC layer to compute the class logits. For the MUTAG dataset, the GNN uses 2 convolutional layers computing 64 and 32 features per node, two FC layers computing 16 and 8 features per graph, and a final FC layer to compute the logits. We implemented these GNNs using PyTorch-Geometric [6]. Models were trained for 200 epochs, optimizing with Adam [11] with a learning rate of 10^{-3} and L2 regularization with weight 10^{-4}.

For the experiments with XGNN, we used the implementation provided by the authors in DIG[2] [16]. For the experiments with GNNInterpreter, we also use the implementation provided by the authors[3]. For the MUTAG dataset, XGNN's graph generator policy network was penalized when it violated valence constraints while generating molecules, and no penalties were used on the other datasets. Both methods optimize objective functions with weighted regularizers, but there is no single quantitative metric for model-level explanation quality, so it is not possible to objectively tune the hyperparameters associated with these methods. Instead, we used default sets of hyperparameters provided in the implementations. An exception was made for XGNN, the default regularization weights caused the graph generator to quickly learn a policy that stopped after the first node in several instances. To fix this, we increased the reward for creating additional valid edges until it became favorable for the model to generate reasonably-sized explanations.

In the experiments with MIPExplainer, adjacency matrices were constrained to be symmetric to represent undirected connected graphs without self-loops. For the MUTAG dataset, node features were constrained to one-hot vectors by ensuring the sum of the elements in each row added up to 1. Any experiments lasting longer than 6 hours were automatically terminated, and we report the best solution found. To ensure that any resemblance to target classes would not come from an initial solution, all runs for our method were initialized with a path graph of n nodes.

[2] https://github.com/divelab/DIG

[3] https://github.com/yolandalalala/GNNInterpreter

	Train Accuracy	Test Accuracy	# of Model Parameters
Is_Acyclic	0.998	1.000	730
Shapes	0.991	0.993	757
MUTAG	0.893	0.895	5770

Table 2. Performance Metrics of Trained GNNs

3.2. Results

The main results from our experiments are shown in Table 3. Note that when depicting molecular graphs, the node colors are assigned as follows: gray=C, blue=N, red=O, cyan=F, purple=I, green=Cl, and brown=Br. In the experiments with Is_Acyclic, MIPExplainer explains the cyclic class with a fully-connected graph, which has the maximum possible number of cycles, and the acyclic class with a star graph, which is one of the most straightforward examples from the class. XGNN does explain the acyclic class with a graph that has fewer cycles than the explanation for the cyclic class, but both graphs are connected somewhat randomly and both contain cycles. On the other hand, GNNInterpreter produces a more densely-connected graph to explain the acyclic class than the cyclic class. For MIPExplainer, the solver was able to prove the optimality of both classes, taking 1.52 seconds for the cyclic class explanation and 227.68 seconds for the acyclic class explanation. The left plot in Figure 4 shows how the bounds converged over the course of the latter experiment. This demonstrates how graph symmetries factor into MIPExplainer's runtime. A fully connected graph with equal node features only has a single adjacency matrix and feature matrix representation, while a star graph with n nodes has n representations, as there are n options for the position of the central node in the node ordering. As a result, more of the search tree must be explored to prove optimality. The right plot in Figure 4 shows the number of explored nodes and unexplored leaf nodes over the course of the search for the optimal solution. A total of 183543 nodes were explored to verify the optimal solution.

For the Shapes dataset, we can easily recognize the classes of each of the explanations generated by MIPExplainer. Despite the fact that a significant amount of noise was added to the training data, the explanations are relatively clean. For reference, three examples from the wheel class are shown in Figure 3. In the cases of lollipops and stars, we see important features of the graph duplicated, a lollipop with two ends and star with two centers. Because we chose a higher number of nodes to make the patterns

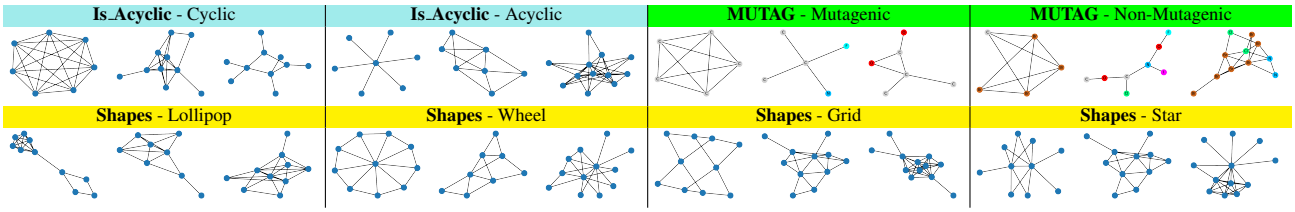

Table 3. Generated Explanations. The graphs from left to right are generated by MIPExplainer, XGNN, and GNNInterpreter, respectively

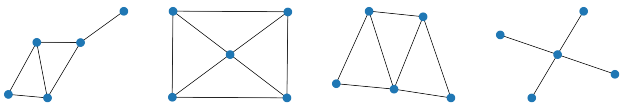

Figure 2. Smaller explanations for lollipops, wheels, grids, and stars respectively

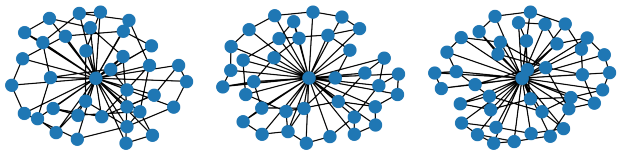

Figure 3. Three randomly selected wheel graphs from the Shapes dataset

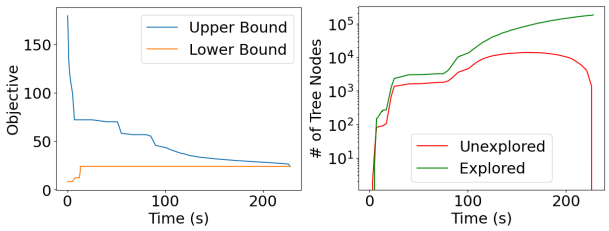

Figure 4. Solver metrics while explaining the acylcic class of Is_Acyclic: Bounds converging to the globally optimum (left), and the number of explored/unexplored nodes in the search tree (right)

clearer, optimality was not proven for these explanations, but they still appear to be reasonable. To further demonstrate that MIPExplainer performs well at varying choices of the number of nodes in the explanation, we provide 5-node explanations for the same classes in Figure 2, which were proven to be optimal.

For the MUTAG dataset, our results show that the model's true behavior may not accurately describe chemistry-related patterns in the data. For the mutagenic class, the produced explanation is simply a complete graph of carbon atoms. Comparing this with the result from the cyclic class of the cyclicity dataset, it seems that the model may simply be looking for arbitrary cycles of carbon atoms, not just ones that occur in molecules. Neither of the explanations generated by the two baseline methods contained a cycle of carbon atoms. The explanation of the non-mutagenic class are not as reasonable across all methods, which is expected since non-mutagens are more accurately described by the absence of mutagenic features than by non-mutagenic features. The generated explanation mostly consists of bromine atoms, which only actually appear in 2 of the graphs in the dataset. Despite the larger network architecture, both these solutions were able to be verified as optimal within the time limit.

4. Conclusion and Discussion

Despite the ability of GNNs to model complex patterns in graph-structured data, their lack of transparency remains one of the key factors hindering their application in a wide range of domains. Model-level explanations of these networks are key to understanding the information they learn and improving their trust and reliability. In order to address key shortcomings that limit the use of existing methods in most real-world situations, this work proposes MIPExplainer for generating such explanations. Without a way to objectively evaluate their quality, it is essential that generated explanations are truly high-quality solutions of optimization problems that are not sensitive to user-defined hyperparameters. MIPExplainer achieves this by avoiding the use of both weighted regularizers and stochastic optimization, instead focusing on maximizing a simpler objective with deterministic methods that are able to prove the global optimality of the generated solutions. Minimal assumptions are made about the distributions of graphs and their features, and no secondary models are trained in the process.

The proposed method also has several shortcomings, which we hope to address in future work. While it is more general than previous methods in some ways, it also requires different GNN layers to be individually encoded with constraints, and may require piecewise-linear approximations for highly nonlinear components. From a practical perspective, the runtime of MIPExplainer as described here is the most significant. Reducing symmetries in the encoding can greatly improve runtime, but this is a hard problem in general, and more work is required to understand which symmetries are the most costly when optimizing over sets of graphs in this way. Despite these limitations, even before proving optimality, we observe that the proposed method is generally able to find reasonable explanations.

References

[1] Navid Ansari, Hans-Peter Seidel, and Vahid Babaei. Mixed integer neural inverse design. *ACM Transactions on Graphics*, 41(4):151:1–151:14, 2022. 2

[2] Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global Explainability of GNNs via Logic Combination of Learned Concepts. 2022. 1, 2

[3] Myun-Seok Cheon. An outer-approximation guided optimization approach for constrained neural network inverse problems. *Mathematical Programming: Series A and B*, 196 (1-2):173–202, 2022. 2

[4] Asim Kumar Debnath, Rosa L. Lopez De Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. 6

[5] Alexandre Duval and Fragkiskos D. Malliaros. GraphSVX: Shapley Value Explanations for Graph Neural Networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 302–318, Cham, 2021. Springer International Publishing. 1

[6] Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 7

[7] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. 6

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. 2017. 1

[9] Kuo-Hsiang Hsu, Bo-Han Su, Yi-Shu Tu, Olivia A. Lin, and Yufeng J. Tseng. Mutagenicity in a Molecule: Identification of Core Structural Features of Mutagenicity Using a Scaffold Analysis. *PLOS ONE*, 11(2):e0148900, 2016. 6

[10] Erwin Kalvelagen. Multiplication of a continuous and a binary variable, 2008. 4

[11] Diederik P. Kingma and Jimmy Ba. 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. 7

[12] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. 2016. 1

[13] Wanyu Lin, Hao Lan, and Baochun Li. Generative causal explanations for graph neural networks. In *International Conference on Machine Learning*, pages 6666–6679. PMLR, 2021. 1

[14] Wanyu Lin, Hao Lan, and Baochun Li. Generative Causal Explanations for Graph Neural Networks. In *International Conference on Machine Learning*, 2021. 2

[15] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. OrphicX: A Causality-Inspired Latent Variable Model for Interpreting Graph Neural Networks. 2022. 2

[16] Meng Liu, Youzhi Luo, Limei Wang, Yaochen Xie, Hao Yuan, Shurui Gui, Zhao Xu, Haiyang Yu, Jingtun Zhang, Yi Liu, Keqiang Yan, Bora Oztekin, Haoran Liu, Xuan Zhang, Cong Fu, and Shuiwang Ji. DIG: A Turnkey Library for Diving into Graph Deep Learning Research. *arXiv preprint arXiv:2103.12608*, 2021. 7

[17] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 4499–4511. PMLR, 2022. 1

[18] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020. 1

[19] Alan Perotti, Paolo Bajardi, Francesco Bonchi, and André Panisson. Explaining Identity-aware Graph Classifiers through the Language of Motifs. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2023. ISSN: 2161-4407. 1

[20] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10772–10781, 2019. 1

[21] Sayan Saha, Monidipa Das, and Sanghamitra Bandyopadhyay. GraphEx: A User-Centric Model-Level Explainer for Graph Neural Networks. In *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. OpenReview.net, 2023. 2

[22] Sayan Saha, Monidipa Das, and Sanghamitra Bandyopadhyay. GraphEx: A User-Centric Model-Level Explainer for Graph Neural Networks. 2023. 2

[23] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577, Proceedings of International Conference on Learning Representations*, 2020. 1

[24] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. *IEEE transactions on pattern analysis and machine intelligence*, 44(11):7581–7596, 2021. 1

[25] Yong-Min Shin, Sun-Woo Kim, and Won-Yong Shin. PAGE: Prototype-Based Model-Level Explanations for Graph Neural Networks. 2022. 2

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. 2017. 1

[27] Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33:12225–12235, 2020. 1

[28] Xiaoqi Wang and Han-Wei Shen. GNNInterpreter: A Probabilistic Generative Model-Level Explanation for Graph Neural Networks. 2022. 2, 3

[29] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? 2018. 5

[30] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019. 1

[31] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph Information Bottleneck for Subgraph Recognition. 2020. 1

[32] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 430–438, Virtual Event CA USA, 2020. ACM. 2

[33] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*, pages 12241–12252. PMLR, 2021. 1

[34] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in Graph Neural Networks: A Taxonomic Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5782–5799, 2023. 1

[35] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. ProtGNN: Towards Self-Explaining Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):9127–9135, 2022. 1