Explaining Graph Neural Networks with Mixed-Integer Programming

Anonymous Authors[1]

Abstract

Graph Neural Networks (GNNs) provide state-of-the-art graph learning performance, but their lack of transparency hinders our ability to understand and trust them, ultimately limiting the areas where they can be applied. Many 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 existing 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 outperforms existing methods in finding accurate and consistent explanations on both synthetic and real-world datasets.

1. Introduction

Graph neural networks (GNNs), such as graph convolutional networks (GCN) (Kipf & Welling, 2016), GraphSAGE networks (Hamilton et al., 2017), and graph attention networks (GAT) (Veličković et al., 2017), 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. However, 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. These problems have motivated a significant amount of research into techniques for GNN explainability.

Research on explainable deep learning proceeds along two lines. One line is to develop intrinsically explainable methods, which modify standard neural networks or the training process so that final models naturally expose information about the importance and interaction of input features. Several proposed GNN architectures aim to achieve inherent explainability, for example, ProtGNN (Zhang et al., 2022) and GIB (Yu et al., 2020). The disadvantage of this approach is that changing the GNN itself to enforce explainability generally comes at the cost of performance. As a result, there is great interest in the second line of research, post-hoc explainability, which aims to interpret networks that have already been trained.

Post-hoc explanation for individual predictions has been extensively explored (see surveys from (Liu et al., 2021; Yuan et al., 2023; Kakkad et al., 2023)), but fewer methods exist to explain the overall patterns used by GNNs to differentiate classes (Yuan et al., 2020; Wang & Shen, 2022; Azzolin et al., 2022). There are several common problems among existing approaches for model-level GNN explanation, which focus on generating graphs to reflect knowledge learned by a model. For one, they often have many hyperparameters that can change the generated explanations, and generating a high-quality explanation may require setting them within a specific range of values. Without a single metric to quantify explanation quality, it is impossible to see if a certain choice of hyperparameters was effective, let alone compare the results across different hyperparameter settings. Furthermore, many methods randomly initialize parameters used to generate the explanations, and then rely on stochastic gradient optimization to assign them values. Due to the stochastic nature of the approach and the necessity of setting a maximum number of iterations, the final explanation's objective value might be far away from the global optimum. More importantly, it leads to significant variation in the explanations across different random initializations of learned parameters, even with the same choices of hyperparameters. Due to the lack of consistency and inability to guarantee solution quality, generating trustworthy global explanations in this way is almost impossible.

Because generating a graph is naturally a discrete process, we propose a new explanation method based on mixed-integer programs (MIPs), which we call MIPExplainer, for finding graph structures or subgraphs that maximally differentiate distinct classes as reported by the GNN. A MIP defines a constrained optimization problem where some of the

decision variables must take integer values. They are commonly solved through branch-and-bound, where the original problem is split into subproblems that partition the set of feasible solutions and solved recursively, creating a search tree. Because an upper bound on any of these subproblems can be found quickly by relaxing the integrality constraints, large subtrees can be pruned if the upper bound at an internal node is less than the objective value for a known solution, improving the tractability of the search. Differing from current methods that search for a graph maximizing the output probability of a single class, we also define a new explanation objective that measures the discriminative power of the GNN. We further propose a new quantitative metric to assess the consistency of the explanations from multiple runs of an explanation method, by measuring the dissimilarity of the generated graphs. While any appropriate graph distance metric can be used in conjunction with our framework (Gao et al., 2010), we employ graph edit distance (Sanfeliu & Fu, 1983b) which is commonly used in inexact graph matching.

MIPExplainer offers several benefits over existing approaches. (1) It directly optimizes over the discrete space of possible input graphs, without any restrictions on types of node and edge features. The only assumptions we make about the space of graphs are bounds on the number of nodes and the magnitude of their features, and we do not require any assumptions about the underlying distribution of the training data. (2) It has a minimal number of hyperparameters that influence the explanation (only the number of nodes of the explanation graph 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, MIPExplainer can place an upper bound on the optimal solution, guaranteeing the quality of the generated explanation.

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 (Yuan et al., 2023), at least six categories of instance-level GNN explanation methods have been proposed so far: gradient-based (Pope et al., 2019), perturbation-based (Yuan et al., 2021; Luo et al., 2020; Ying et al., 2019; Schlichtkrull et al., 2020), surrogate (Vu & Thai, 2020), generation-based (Lin et al., 2021), decomposition (Schnake et al., 2021), and counterfactual-based (Lucic et al., 2022) methods. These methods do not immediately provide insights into the overall patterns a GNN has identified, but it is possible to consolidate instance-level explanations to reveal model-level patterns. For example, we can employ purely statistical methods to determine whether there are nodes/edges shared by a significant portion of the individual explanations. A more recent technique, GLGExplainer (Azzolin et al., 2022), finds smaller components of the extracted explanations that can be used to build logical expressions consistent with the overall GNN's predictions. However, these methods are limited by the scope of the training data, and can be influenced by bias in the dataset. By allowing for explanations to be out of distribution, we are able to isolate graph structures that may not appear by themselves without noise in the actual data. A direct model-level explanation can offer more faithful explanations, and is more useful for determining the degree of bias in the model itself.

Relatively few methods exist to explain GNNs at the model level. XGNN (Yuan et al., 2020) is the most widely used, and serves as the only baseline in several recent papers that focus on similar objectives (Azzolin et al., 2022; Saha et al., 2023a; Shin et al., 2022; Wang & Shen, 2022). 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 trains a second neural network by reinforcement learning to generate graphs that obey explicit generation rules and maximize the original GNN's prediction for a specific class. GNNInterpreter (Wang & Shen, 2022) and GraphEx (Saha et al., 2023b) avoid training a second neural network by assuming that the graphs in the dataset are sampled from a set of underlying distributions parameterized by continuous latent parameters. In particular, 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. GCExplainer (Magister et al., 2021) and the work by (Xuanyuan et al., 2023) are also global explanation methods, but both focus on explaining GNNs via concept generation, which is a separate approach that relies on identifying patterns in activation maps of the training data.

In the past, discrete optimization techniques have been applied to deep neural networks, such as in (Cheon, 2022), (Botoeva et al., 2020), and (Ansari et al., 2022). However, these existing methods were only defined for layers consisting of a linear transformation with ReLU activations, and solve constraint satisfaction or optimization problems related to inverse design and verification. This work will reformulate the optimization problem for explainability and generalize the application of mixed-integer programming on standard neural networks to a range of GNN architectures.

2. MIPExplainer

Our model-level explanation seeks to optimize an input graph $G = (X, A)$ on which the GNN maximally differentiates one class from the rest, where X contains the d attributes for each of N graph nodes as row vectors 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 and add constraints to represent the computation in that 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 in this layer. 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

A typical objective function for explanations contains two parts: a term related to class prediction and a regularizer that enforces the generated explanations to be in-distribution. In this paper, we decide not to apply any regularization in the objective function in order to minimize the number of hyperparameters; please see our detailed discussions in the Appendix. 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 (Wang & Shen, 2022)), 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.

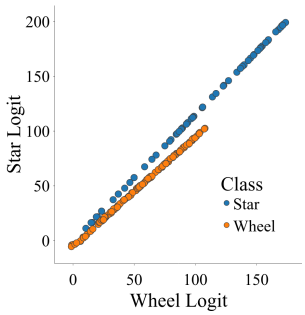

Figure 1. Logits of Star and Wheel Graphs in the Shapes Dataset

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 due to improvements getting 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 for only the target class, but an optimal solution may simply minimize one logit while leaving other logits close to or even greater than the logit of the target class, resulting in an incorrect explanation. 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, this approach is more effective, so we focus on discussing the following objective function:

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

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.

From the range of existing GNN layers, we choose to focus first on GraphSAGE convolution layers, where the updated node representations X' after a layer are calculated from existing node representations X with the formula

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

The aggregation on a node can be realized, for example, by summing its neighbors' feature vectors, 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 notations: the matrix of scalars, $W^{(i)}$, and the vector of scalars, $b^{(i)}$, denote the GNN's matrix of learned weights and learned bias vector in layer i. For 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 and discuss how they are constrained shortly: $\Phi^{(i)}$ 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]. The vector d (with some abuse of notation) is an indicator representing whether each element of $\Phi^{(L)}$ is the maximum element in the output of layer L except 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. Here, 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:

$$y \geq X_{\neq c}^{(L)}, \quad (10)$$

$$y \leq X_{\neq c}^{(L)} + (\max(U_{X_{\neq c}^{(L)}})\mathbf{1} - L_{X_{\neq c}^{(L)}})(1 - d), \quad (11)$$

$$\sum_j d_j = 1, d_j \in \{0, 1\}, \quad (12)$$

[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.

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, which makes the optimization significantly easier. 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 (Kalvelagen, 2008). 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 convex objective function and all linear constraints when integrality is relaxed, so it is a mixed-integer linear program (MILP).

2.3. Additional 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 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. The ith node found by DFS must have been found from one of the 1 through $i-1$ nodes, so this is always true.

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 (Xu et al., 2018), 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. Existence of Global Optima

We now show that a globally optimal solution to the MIP problem described above always exists.

Theorem 2.1. *Consider the MIP 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 MIP.*

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 MIP 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 MIP must have an optimum on the compact feasible set. 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. □

2.6. Optimization Algorithm

We can employ a branch and bound procedure, along with cutting planes and heuristics, to find a globally optimal solution efficiently. We present a branch and bound algorithm in Algorithm 1 to solve our MIP described in the previous section, which is represented by its set of constraints C and objective function f. We obtain an initial solution at the root of a search tree by choosing an initialization graph $G_0 = (X_0, A_0)$ and applying the GNN to obtain initial values for all the intermediate variables.

Algorithm 1 MIP Branch and Bound Procedure

1: **Input:** The constraint set C, the objective function f, and an initial graph $G_0 = (X_0, A_0)$
2: Initialize a queue Q containing only C
3: $L \leftarrow f(G_0)$
4: $z \leftarrow G_0$
5: **while** Q is not empty **do**
6: $N \leftarrow$ search node popped from Q
7: Solve the linear relaxation of N, denoted N_r, and store the result in z^*
8: $U \leftarrow f(z^*)$
9: **if** N_r was feasible and $U > L$ **then**
10: **if** z^* obeys all integrality constraints, defining a valid graph G^* **then**
11: $z \leftarrow z^*$
12: $L \leftarrow f(G^*)$
13: **else**
14: $v \leftarrow$ An integer variable with a non-integral value z_v^* in z^*
15: Add $N \cup \{v \leq \lfloor z_v^* \rfloor\}$ and $N \cup \{v \geq \lceil z_v^* \rceil\}$ to Q
16: **end if**
17: **else**
18: Prune the entire subtree rooted at N by continuing to the next iteration without adding any nodes to Q
19: **end if**
20: **end while**
21: **return** z

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 (line 7). This is an optimal solution to a problem with fewer

constraints, so it serves as an upper bound to the original problem with integer domains for some decision variables. If it happens to also be a solution to the original MIP, meaning all the variables are integral, then the correctness of the algorithm for solving the relaxation guarantees that this is an optimal solution to our MIP, and we can stop searching (lines 10-12).

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 (lines 14-15). 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, leading to a binary tree in which nodes represent further constrained versions of the original MIP. While an upper bound for an internal node (U in the pseudocode) only applies to the subtree rooted at that node, any integer solution serves as a lower bound (L in the pseudocode) to the integral problem's optimum anywhere in the search tree.

Integer solutions can be found in the leaves of our search tree, or using heuristics to complete partial solutions defined at internal nodes. If the linear relaxation solved at an internal node is infeasible or has a maximum objective value that is lower than or equal to our current lower bound, we have proven that a new optimal solution to the integral problem cannot lie anywhere in the subtree rooted at that node, allowing us to prune the branch and skip all the nodes it contains in our search (line 18). The process stops when there are no more subproblems to explore, at which point we will have found an optimal solution to the integral problem. In our experiments, we use Gurobi Optimizer (Gurobi Optimization, LLC, 2023) to find an optimal solution efficiently. This solver combines branch-and-bound with cutting plane methods, which makes the optimization process even faster. While the theoretical complexity of this algorithm is exponential, the average complexity is significantly lower in practice, making it tractable to apply in many situations.

A single, large number M can be used to bound all of the continuous decision variables, but 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 get upper and lower bounds on the output of a linear layer $x' = Wx + b$:

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

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

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.

Further discussion on the practical considerations of solving these MIPs can be found in Appendix Section C.

3. Experiments

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 (Debnath et al., 1991) 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 (Hsu et al., 2016), mutagenic molecules tend to have higher numbers of fused rings of carbon atoms. For this dataset, each node's features are a one-hot vector indicating atom type.

To quantify the variation between explanation graphs, we run repeated experiments with each explanation method and measure the average graph edit distance between all pairs of explanations. Graph edit distance, as described in (Sanfeliu & Fu, 1983a), is the minimum number of graph edit operations (vertex/edge insertions/deletions/substitutions)

	# 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

	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

needed to transform one graph into another. A lower average graph edit distance indicates a more consistent explanation method.

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. For the experiments with XGNN, we used the implementation provided by the authors in DIG[2] (Liu et al., 2021). For the experiments with GNNInterpreter, we also use the implementation provided by the authors[3]. We used default sets of hyperparameters provided in the papers and implementations of the baselines. An exception was made for XGNN because the default regularization weights sometimes 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. Additional details about the experiments can be found in Appendix B. To measure the consistency of the explanations, we generated explanations with 5,6,7, and 8 nodes using each method on each dataset 5 times. Then, we computed the average GED among the 5 explanations. Table 4 shows these metrics averaged over the different numbers of nodes. A full table containing separate results for each number of nodes can be found in Appendix D. For MIPExplainer, the initialization graph was created by starting with a path graph and adding all other possible edges with probability 0.5. Variation in the baselines comes from the random initialization of the explanation network for XGNN and the latent parameters in GNNInterpreter.

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 complete graph, which has the maximum possible number of cycles. It explains the acyclic class with a star graph, which is one of the most straightforward examples from the class. In contrast, the explanation graphs of XGNN and GNNInterpreter for the cyclic class contain some nodes with a single neighbor, and their explanations for the acyclic class even include multiple cycles. Our solver was able to prove the optimality of both classes, taking an average of 1.90 seconds for the cyclic class explanation and 119.14 seconds for the acyclic class explanation. The left plot in Figure 5 shows how the bounds converged over the course of the acyclic class experiment. This demonstrates how graph symmetries factor into MIPExplainer's runtime, we further discuss this problem in Appendix C. 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, despite the solution having the same number of nodes and fewer edges, more of the search tree must be explored to prove the optimality of the acyclic explanation. Figure 5 shows the convergence of the objective bounds and the number of explored and unexplored search nodes over the course of the search for the optimal solution. Similar figures for more experimental settings can be found in Appendix D.

For the mutagenic class of the MUTAG dataset, the MIPExplainer produces a complete graph of carbon atoms. While the presence of carbon cycles are an important factor in the mutagenicity of organic molecules, they appear exclusively as rings of 5 or 6 carbon atoms. Neither of the explanations generated by the two baseline methods contained a cycle of carbon atoms. The explanations 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 the presence of 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.

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 2. 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 clearer, optimality was not proven for these explanations, but they still appear reasonable.

Table 4 shows that MIPExplainer was significantly more

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

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

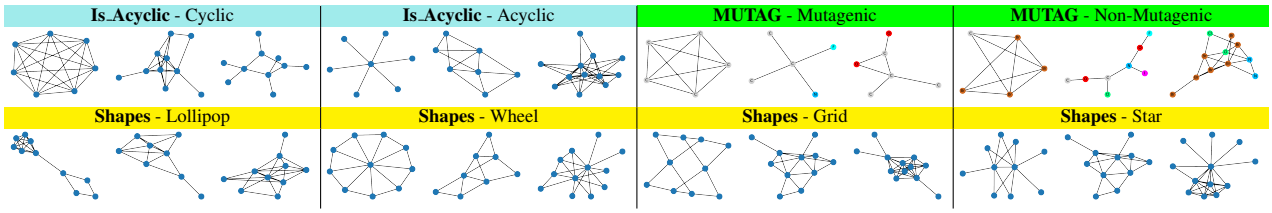

Table 3. Generated Explanations. The graphs from left to right are generated by MIPExplainer, XGNN, and GNNInterpreter, respectively

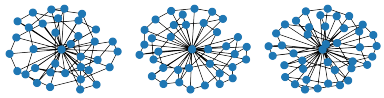

Figure 2. Three randomly selected wheel graphs from the Shapes dataset

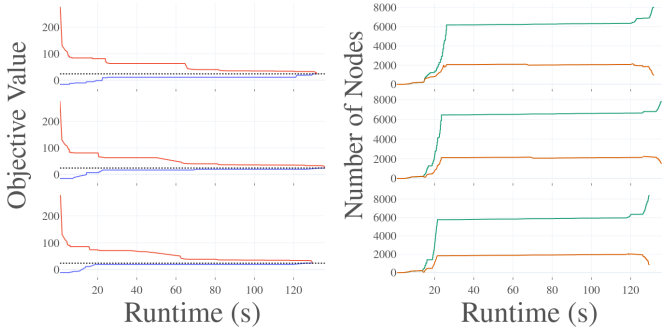

Figure 3. Solver metrics for 3 runs explaining the Acylcic class of Is_Acyclic with 7 nodes: On the left, the current best solution's objective (blue) and upper bound (red) converging to the same global optimum (the dotted black line). On the right, the number of explored (green)/unexplored (orange) nodes during the search.

consistent than the baseline methods in all experimental settings. In some cases, small variations in explanations are due to the existence of multiple explanation graphs with the exact same objective value, which tended to be extremely similar. In other cases, it was due to the algorithm running out of time before finding the optimal solution. However, this also rarely caused deviations, as the best solution was generally found much earlier than it was proven to be optimal. For example, proving optimality for the MUTAG dataset for an explanation graph with 8 nodes always took longer than 30 minutes, but across all 5 runs with random initializations, the algorithm consistently produced the same explanation after 30 minutes. An extended table showing separate consistency metrics for differently sized explanations, as well as tables showing the output logits for the explanation graphs and runtimes for each experiment, can be found in Appendix D.

4. Conclusion and Discussion

Despite the ability of GNNs to model complex patterns in graph-structured data, their lack of transparency remains one

		Averaged Edit Distance		
		MIPExplainer	GNNInterpreter	XGNN
Dataset	Class			
Is_Acyclic	Acyclic	**0.2 ± 0.40**	3.5 ± 0.77	2.8 ± 1.62
	Cyclic	**0.0 ± 0.00**	3.1 ± 1.05	3.2 ± 1.82
MUTAG	Mutagen	**0.7 ± 1.40**	8.1 ± 1.19	7.8 ± 2.17
	Nonmutagen	**0.4 ± 0.80**	7.8 ± 1.56	7.4 ± 2.39
Shapes	Grid	**0.0 ± 0.00**	3.6 ± 0.89	3.0 ± 2.42
	Lollipop	**0.2 ± 0.40**	3.5 ± 0.77	3.4 ± 2.03
	Star	**0.0 ± 0.00**	3.9 ± 1.21	3.0 ± 2.60
	Wheel	**0.75 ± 0.90**	3.3 ± 1.04	4.0 ± 3.32

Table 4. Average edit distance between 5 generated example graphs, averaged for numbers of nodes between 5 and 8 inclusive. Time limit is one half hour.

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 drawback. 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. Despite these limitations, even before proving optimality, we observe that the proposed method is able to find reasonable explanations.

Impact Statement

Techniques for improving the explainability of deep neural networks have significant implications for safety and transparency wherever this technology is applied. While the method proposed in this work may be useful for elucidating implicitly learned patterns and biases, it is not able to make any guarantees about fairness or robustness, and should not be included in the verification process necessary for the safe deployment of deep learning models.

References

Ansari, N., Seidel, H.-P., and Babaei, V. Mixed integer neural inverse design. *ACM Transactions on Graphics*, 41(4):151:1–151:14, July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530083. URL https://dl.acm.org/doi/10.1145/3528223.3530083.

Azzolin, S., Longa, A., Barbiero, P., Liò, P., and Passerini, A. Global Explainability of GNNs via Logic Combination of Learned Concepts. 2022. doi: 10.48550/ARXIV.2210.07147. URL https://arxiv.org/abs/2210.07147.

Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3291–3299, April 2020. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v34i04.5729. URL https://ojs.aaai.org/index.php/AAAI/article/view/5729.

Cheon, M.-S. An outer-approximation guided optimization approach for constrained neural network inverse problems. *Mathematical Programming: Series A and B*, 196(1-2):173–202, November 2022. ISSN 0025-5610. doi: 10.1007/s10107-021-01653-y. URL https://doi.org/10.1007/s10107-021-01653-y.

Debnath, A. K., Lopez De Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. 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, February 1991. ISSN 0022-2623, 1520-4804. doi: 10.1021/jm00106a046. URL https://pubs.acs.org/doi/abs/10.1021/jm00106a046.

Fey, M. and Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gao, X., Xiao, B., Tao, D., and Li, X. A survey of graph edit distance. *Pattern Analysis and applications*, 13:113–129, 2010.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive Representation Learning on Large Graphs. June 2017. doi: 10.48550/arXiv.1706.02216. URL https://arxiv.org/abs/1706.02216v4.

Hsu, K.-H., Su, B.-H., Tu, Y.-S., Lin, O. A., and Tseng, Y. J. Mutagenicity in a Molecule: Identification of Core Structural Features of Mutagenicity Using a Scaffold Analysis. *PLOS ONE*, 11(2):e0148900, February 2016. ISSN 1932-6203. doi: 10.1371/journal.pone.0148900. URL https://dx.plos.org/10.1371/journal.pone.0148900.

Kakkad, J., Jannu, J., Sharma, K., Aggarwal, C., and Medya, S. A Survey on Explainability of Graph Neural Networks, June 2023. URL http://arxiv.org/abs/2306.01958. arXiv:2306.01958 [cs].

Kalvelagen, E. Multiplication of a continuous and a binary variable, May 2008. URL http://yetanothermathprogrammingconsultant.blogspot.com/2008/05/multiplication-of-continuous-and-binary.html.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. November 2016. URL https://openreview.net/forum?id=SJU4ayYgl.

Lin, W., Lan, H., and Li, B. Generative causal explanations for graph neural networks. In *International Conference on Machine Learning*, pp. 6666–6679. PMLR, 2021.

Liu, M., Luo, Y., Wang, L., Xie, Y., Yuan, H., Gui, S., Xu, Z., Yu, H., Zhang, J., Liu, Y., Yan, K., Oztekin, B., Liu, H., Zhang, X., Fu, C., and Ji, S. DIG: A Turnkey Library for Diving into Graph Deep Learning Research. *arXiv preprint arXiv:2103.12608*, 2021.

Lucic, A., Ter Hoeve, M. A., Tolomei, G., De Rijke, M., and Silvestri, F. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 4499–4511. PMLR, 2022.

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.

Magister, L. C., Kazhdan, D., Singh, V., and Liò, P. GCExplainer: Human-in-the-Loop Concept-based Explanations for Graph Neural Networks. In *Workshop on Human in the Loop Learning*, volume 3. ICML, 2021. doi: 10.48550/ARXIV.2107.11889. URL https://arxiv.org/abs/2107.11889.

Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10772–10781, 2019.

Saha, S., Das, M., and Bandyopadhyay, S. GraphEx: A User-Centric Model-Level Explainer for Graph Neural Networks. In Maughan, K., Liu, R., and Burns, T. F. (eds.), *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. OpenReview.net, 2023a. URL https://openreview.net/pdf?id=CuE1F1M0_yR.

Saha, S., Das, M., and Bandyopadhyay, S. GraphEx: A User-Centric Model-Level Explainer for Graph Neural Networks. March 2023b. URL https://openreview.net/forum?id=CuE1F1M0_yR.

Sanfeliu, A. and Fu, K.-S. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13 (3):353–362, 1983a. doi: 10.1109/TSMC.1983.6313167.

Sanfeliu, A. and Fu, K.-S. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13 (3):353–362, May 1983b. ISSN 0018-9472, 2168-2909. doi: 10.1109/TSMC.1983.6313167. URL http://ieeexplore.ieee.org/document/6313167/.

Schlichtkrull, M. S., De Cao, N., and Titov, I. Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577, Proceedings of International Conference on Learning Representations*, 2020.

Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., and Montavon, G. Higher-order explanations of graph neural networks via relevant walks. *IEEE transactions on pattern analysis and machine intelligence*, 44(11):7581–7596, 2021.

Shin, Y.-M., Kim, S.-W., and Shin, W.-Y. PAGE: Prototype-Based Model-Level Explanations for Graph Neural Networks. 2022. doi: 10.48550/ARXIV.2210.17159. URL https://arxiv.org/abs/2210.17159.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. 2017. doi: 10.48550/ARXIV.1710.10903. URL https://arxiv.org/abs/1710.10903.

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

Wang, X. and Shen, H.-W. GNNInterpreter: A Probabilistic Generative Model-Level Explanation for Graph Neural Networks. 2022. doi: 10.48550/ARXIV.2209.07924. URL https://arxiv.org/abs/2209.07924.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How Powerful are Graph Neural Networks? 2018. doi: 10.48550/ARXIV.1810.00826. URL https://arxiv.org/abs/1810.00826.

Xuanyuan, H., Barbiero, P., Georgiev, D., Magister, L. C., and Liò, P. Global concept-based interpretability for graph neural networks via neuron analysis. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, volume 37 of *AAAI'23/IAAI'23/EAAI'23*, pp. 10675–10683. AAAI Press, February 2023. ISBN 9781577358800. doi: 10.1609/aaai.v37i9.26267. URL https://doi.org/10.1609/aaai.v37i9.26267.

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

Yu, J., Xu, T., Rong, Y., Bian, Y., Huang, J., and He, R. Graph Information Bottleneck for Subgraph Recognition. October 2020. URL https://openreview.net/forum?id=bM4Iqfg8M2k.

Yuan, H., Tang, J., Hu, X., and Ji, S. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 430–438, Virtual Event CA USA, August 2020. ACM. ISBN 9781450379984. doi: 10.1145/3394486.3403085. URL https://dl.acm.org/doi/10.1145/3394486.3403085.

Yuan, H., Yu, H., Wang, J., Li, K., and Ji, S. On explainability of graph neural networks via subgraph explorations. In *International conference on machine learning*, pp. 12241–12252. PMLR, 2021.

Yuan, H., Yu, H., Gui, S., and Ji, S. Explainability in Graph Neural Networks: A Taxonomic Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5): 5782–5799, May 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3204236. URL https://ieeexplore.ieee.org/abstract/document/9875989.

Zhang, Z., Liu, Q., Wang, H., Lu, C., and Lee, C. ProtGNN: Towards Self-Explaining Graph Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):9127–9135, June 2022. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v36i8.20898. URL https://ojs.aaai.org/index.php/AAAI/article/view/20898.

A. Regularization Terms

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.

B. Experimental Setup

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 (Fey & Lenssen, 2019). Models were trained for 200 epochs, optimizing with Adam (Kingma & Ba, 2015) with a learning rate of 10^{-3} and L2 regularization with weight 10^{-4}.

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. 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 in 3 were initialized with a path graph of n nodes. On the other hand, to test consistency, runs for MIPExplainer were initialized with a graph generated by adding every possible edge to a line graph with probability 0.5.

C. Practical Considerations

In practice, it can be difficult to solve MIPs 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.

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.

D. Extended Results

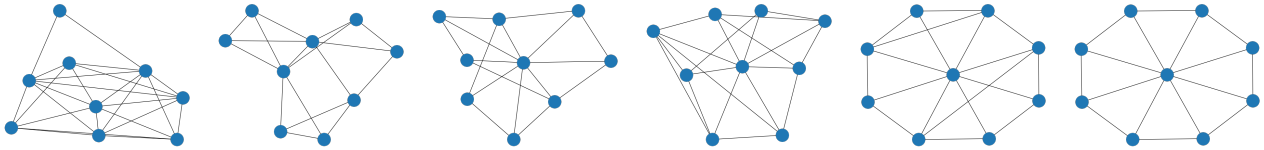

Figure 4. Improving solutions found while optimizing the explanation for the Wheel class from the Shapes dataset

			Average Edit Distance		
			MIPExplainer	GNNInterpreter	XGNN
Dataset	Class	# Nodes			
Is_Acyclic	Acyclic	5	0.000 ± 0.000	3.000 ± 1.549	0.600 ± 0.490
		6	0.000 ± 0.000	3.200 ± 1.077	2.400 ± 1.200
		7	0.800 ± 0.980	3.000 ± 1.000	4.000 ± 1.414
		8	0.000 ± 0.000	4.600 ± 1.020	4.000 ± 0.775
	Cyclic	5	0.000 ± 0.000	2.600 ± 1.428	1.200 ± 0.600
		6	0.000 ± 0.000	2.200 ± 1.249	2.800 ± 1.077
		7	0.000 ± 0.000	3.000 ± 1.000	3.400 ± 1.744
		8	0.000 ± 0.000	4.600 ± 1.020	5.600 ± 1.497
MUTAG	Mutagen	5	0.000 ± 0.000	7.000 ± 1.483	5.200 ± 1.327
		6	0.000 ± 0.000	7.300 ± 1.616	7.000 ± 1.414
		7	2.800 ± 3.429	8.500 ± 1.285	8.500 ± 1.500
		8	0.000 ± 0.000	9.600 ± 2.458	10.300 ± 1.345
	Nonmutagen	5	0.000 ± 0.000	5.800 ± 1.327	4.600 ± 1.428
		6	0.000 ± 0.000	7.500 ± 1.204	7.000 ± 1.549
		7	1.600 ± 1.960	8.400 ± 1.428	7.700 ± 0.781
		8	0.000 ± 0.000	9.500 ± 1.025	10.400 ± 2.107
Shapes	Grid	5	0.000 ± 0.000	4.400 ± 2.107	1.200 ± 0.748
		6	0.000 ± 0.000	2.600 ± 0.917	1.000 ± 0.632
		7	0.000 ± 0.000	3.000 ± 0.894	3.400 ± 1.855
		8	0.000 ± 0.000	4.200 ± 0.872	6.200 ± 2.441
	Lollipop	5	0.000 ± 0.000	2.800 ± 1.249	1.600 ± 1.428
		6	0.800 ± 0.980	3.000 ± 0.894	2.000 ± 0.894
		7	0.000 ± 0.000	4.000 ± 1.000	4.000 ± 1.000
		8	0.000 ± 0.000	4.400 ± 1.356	6.000 ± 2.145
	Star	5	0.000 ± 0.000	5.200 ± 2.272	1.200 ± 0.980
		6	0.000 ± 0.000	2.600 ± 0.663	1.000 ± 0.632
		7	0.000 ± 0.000	3.200 ± 0.748	3.400 ± 1.200
		8	0.000 ± 0.000	4.600 ± 1.356	6.600 ± 3.611
	Wheel	5	0.000 ± 0.000	1.800 ± 0.980	0.600 ± 0.490
		6	0.000 ± 0.000	4.000 ± 1.183	2.000 ± 0.775
		7	1.200 ± 0.600	3.400 ± 1.428	5.200 ± 2.750
		8	1.800 ± 0.600	4.000 ± 0.894	8.000 ± 3.924

Table 5. Average edit distance between 5 generated example graphs. The time limit was one half hour for all experiments. MIPExplainer had the lowest average edit distance in all experiments.

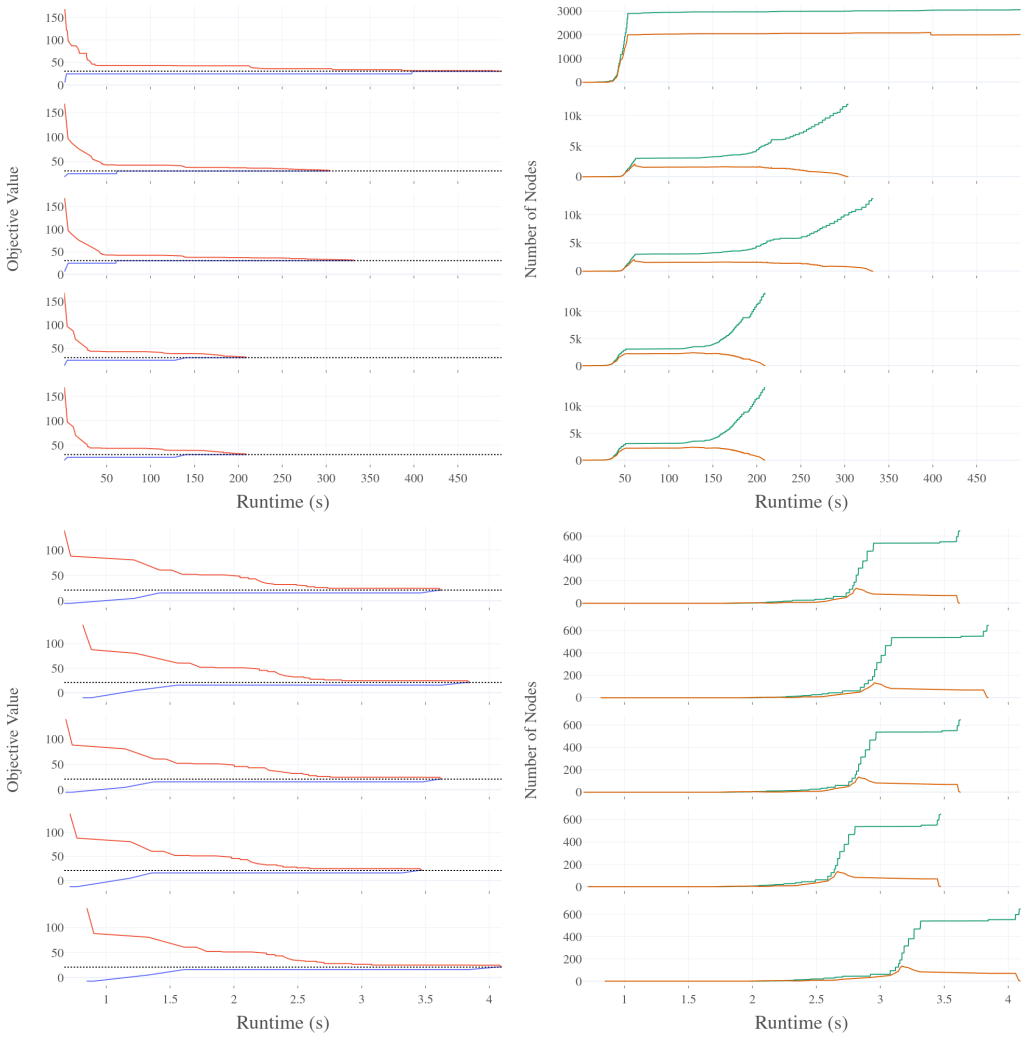

Figure 5. Solver metrics for 5 runs explaining the Mutagen class of MUTAG (Top) and the Acyclic class of Is_Acyclic (bottom) with 5 nodes: On the left, the current best solution's objective (blue) and upper bound (red) converging to the same global optimum (the dotted black line). On the right, the number of explored (green)/unexplored (orange) nodes during the search.

Dataset	Class	# Nodes	Runtime MIPExplainer	GNNInterpreter	XGNN
Is_Acyclic	Acyclic	5	3.965 ± 0.313	7.610 ± 0.015	10.040 ± 0.135
		6	13.752 ± 0.767	7.626 ± 0.022	13.482 ± 0.528
		7	119.144 ± 19.283	7.679 ± 0.054	15.419 ± 0.561
		8	919.366 ± 387.422	7.684 ± 0.013	17.265 ± 0.479
	Cyclic	5	1.701 ± 0.356	0.027 ± 0.008	9.842 ± 0.057
		6	1.781 ± 0.521	0.024 ± 0.000	12.877 ± 0.482
		7	1.902 ± 0.102	0.030 ± 0.003	15.257 ± 0.377
		8	1.945 ± 0.098	0.115 ± 0.051	18.216 ± 1.273
MUTAG	Mutagen	5	312.586 ± 120.643	0.024 ± 0.001	8.701 ± 0.455
		6	901.010 ± 82.795	0.025 ± 0.004	10.305 ± 0.297
		7	1157.387 ± 386.822	0.060 ± 0.060	12.069 ± 0.257
		8	3031.742 ± 288.508	0.114 ± 0.007	14.727 ± 0.619
	Nonmutagen	5	1916.024 ± 753.053	5.633 ± 5.109	8.747 ± 0.530
		6	7201.688 ± 0.890	7.599 ± 4.213	10.403 ± 0.342
		7	7201.532 ± 0.087	9.659 ± 0.049	12.595 ± 0.731
		8	7201.842 ± 0.122	7.857 ± 4.304	13.946 ± 0.429
Shapes	Grid	5	4.964 ± 1.060	7.563 ± 0.018	11.644 ± 0.196
		6	17.508 ± 3.246	7.614 ± 0.040	14.964 ± 0.151
		7	92.649 ± 19.726	7.648 ± 0.022	17.817 ± 0.149
		8	1886.365 ± 537.301	7.667 ± 0.007	20.497 ± 0.735
	Lollipop	5	7.560 ± 1.296	7.631 ± 0.016	11.543 ± 0.102
		6	60.700 ± 4.313	7.634 ± 0.013	15.048 ± 0.410
		7	172.511 ± 44.716	7.661 ± 0.019	17.934 ± 0.370
		8	4261.094 ± 2267.636	7.705 ± 0.027	19.852 ± 0.355
	Star	5	4.050 ± 0.711	7.560 ± 0.014	11.512 ± 0.173
		6	18.200 ± 2.062	7.592 ± 0.007	14.955 ± 0.279
		7	176.005 ± 113.007	7.624 ± 0.014	17.942 ± 0.440
		8	260.384 ± 40.326	7.659 ± 0.016	19.931 ± 0.673
	Wheel	5	4.256 ± 1.225	7.716 ± 0.255	11.550 ± 0.043
		6	107.301 ± 134.613	7.641 ± 0.010	15.507 ± 0.763
		7	103.210 ± 8.210	7.652 ± 0.010	18.047 ± 0.395
		8	5576.961 ± 1071.344	7.706 ± 0.051	20.095 ± 0.387

Table 6. Runtime of explanation methods given a time limit of 2 hours, averaged over 5 runs

			Is_Acyclic Output Logits		
			MIPExplainer	GNNInterpreter	XGNN
Class	# Nodes				
Acyclic	5	Cyclic Logit	-8.981 ± 0.000	-1.343 ± 4.048	6.417 ± 0.709
		Acyclic Logit	11.896 ± 0.000	1.835 ± 5.022	-7.456 ± 0.800
	6	Cyclic Logit	-9.722 ± 0.000	1.452 ± 4.292	7.333 ± 1.891
		Acyclic Logit	12.932 ± 0.000	-1.577 ± 5.222	-8.490 ± 2.134
	7	Cyclic Logit	-9.184 ± 2.476	4.607 ± 1.646	6.520 ± 1.742
		Acyclic Logit	12.247 ± 3.341	-5.415 ± 1.857	-7.573 ± 1.965
	8	Cyclic Logit	-10.754 ± 0.000	4.924 ± 1.633	6.372 ± 1.884
		Acyclic Logit	14.409 ± 0.000	-5.771 ± 1.843	-7.400 ± 2.131
Cyclic	5	Cyclic Logit	7.194 ± 0.000	-5.769 ± 2.112	5.665 ± 1.055
		Acyclic Logit	-8.333 ± 0.000	7.516 ± 2.844	-6.609 ± 1.191
	6	Cyclic Logit	10.260 ± 0.000	0.200 ± 4.304	6.997 ± 2.119
		Acyclic Logit	-11.793 ± 0.000	-0.080 ± 5.363	-8.111 ± 2.391
	7	Cyclic Logit	13.488 ± 0.000	-0.293 ± 1.360	7.080 ± 1.031
		Acyclic Logit	-15.436 ± 0.000	0.524 ± 1.625	-8.205 ± 1.164
	8	Cyclic Logit	16.870 ± 0.000	4.585 ± 1.148	7.587 ± 1.127
		Acyclic Logit	-19.184 ± 0.000	-5.328 ± 1.362	-8.777 ± 1.271

Table 7. Logits for Is_Acyclic explanation graphs generated by MIPExplainer, averaged over 5 runs with random initial solutions

			MUTAG Output Logits		
			MIPExplainer	GNNInterpreter	XGNN
Class	# Nodes				
Mutagen	5	Nonmutagen Output Logit	-19.249 ± 0.000	-0.422 ± 5.318	3.825 ± 5.060
		Mutagen Output Logit	11.269 ± 0.000	4.048 ± 7.581	-0.041 ± 5.502
	6	Nonmutagen Output Logit	-30.439 ± 0.000	-7.649 ± 7.386	-1.738 ± 8.144
		Mutagen Output Logit	17.673 ± 0.000	18.074 ± 13.474	7.321 ± 8.283
	7	Nonmutagen Output Logit	-43.654 ± 0.000	-9.854 ± 13.851	-1.755 ± 3.123
		Mutagen Output Logit	25.229 ± 0.000	20.275 ± 23.446	4.828 ± 6.009
	8	Nonmutagen Output Logit	-59.041 ± 0.000	-21.218 ± 12.229	-1.859 ± 5.728
		Mutagen Output Logit	34.029 ± 0.000	41.498 ± 19.722	5.446 ± 7.365
Nonmutagen	5	Nonmutagen Output Logit	6.691 ± 0.000	1.887 ± 4.245	-3.481 ± 1.849
		Mutagen Output Logit	-7.612 ± 0.000	-0.107 ± 5.117	8.834 ± 2.699
	6	Nonmutagen Output Logit	6.830 ± 0.000	7.146 ± 8.787	2.359 ± 3.898
		Mutagen Output Logit	-7.779 ± 0.000	-0.851 ± 6.609	1.204 ± 6.986
	7	Nonmutagen Output Logit	7.185 ± 0.098	-3.194 ± 7.405	-0.841 ± 6.120
		Mutagen Output Logit	-8.174 ± 0.102	10.138 ± 9.418	3.811 ± 8.576
	8	Nonmutagen Output Logit	7.236 ± 0.000	-5.436 ± 10.869	-9.350 ± 15.218
		Mutagen Output Logit	-8.251 ± 0.000	13.410 ± 16.078	19.037 ± 22.654

Table 8. Logits for MUTAG explanation graphs generated by MIPExplainer, averaged over 5 runs with random initial solutions

			Shapes Output Logits (1)		
			MIPExplainer	GNNInterpreter	XGNN
Class	# Nodes				
Grid	5	Lollipop Logit	-1.343 ± 0.000	-10.019 ± 6.822	-5.121 ± 1.118
		Wheel Logit	-19.762 ± 0.000	-38.818 ± 40.875	3.259 ± 6.784
		Grid Logit	9.577 ± 0.000	-8.103 ± 23.501	8.262 ± 0.688
		Star Logit	-20.261 ± 0.000	-14.756 ± 18.554	-31.471 ± 4.109
	6	Lollipop Logit	-0.846 ± 0.000	-3.386 ± 3.512	-4.907 ± 1.180
		Wheel Logit	-1.949 ± 0.000	-21.545 ± 11.607	5.398 ± 3.124
		Grid Logit	9.154 ± 0.000	-6.520 ± 11.184	4.816 ± 2.687
		Star Logit	-31.297 ± 0.000	-17.602 ± 8.887	-30.366 ± 3.257
	7	Lollipop Logit	-1.269 ± 0.000	-2.233 ± 2.319	-1.976 ± 0.763
		Wheel Logit	-11.014 ± 0.000	-5.299 ± 5.715	-4.893 ± 2.458
		Grid Logit	9.972 ± 0.000	3.110 ± 3.284	-1.793 ± 7.365
		Star Logit	-25.884 ± 0.000	-23.110 ± 2.283	-21.878 ± 3.027
	8	Lollipop Logit	-1.502 ± 0.000	-2.884 ± 3.272	-2.444 ± 1.782
		Wheel Logit	-8.745 ± 0.000	-11.408 ± 6.947	-5.672 ± 8.371
		Grid Logit	9.270 ± 0.000	1.346 ± 5.852	-7.951 ± 13.444
		Star Logit	-26.275 ± 0.000	-19.579 ± 3.193	-19.534 ± 7.940
Lollipop	5	Lollipop Logit	-2.791 ± 0.000	-8.329 ± 8.233	-5.837 ± 0.697
		Wheel Logit	-25.569 ± 0.000	-47.927 ± 27.593	-0.149 ± 22.146
		Grid Logit	-5.128 ± 0.000	-15.098 ± 19.693	3.951 ± 10.263
		Star Logit	-19.403 ± 0.000	-9.227 ± 13.375	-30.313 ± 12.808
	6	Lollipop Logit	1.103 ± 1.065	-3.452 ± 3.391	-2.709 ± 1.291
		Wheel Logit	-17.920 ± 8.127	-22.403 ± 11.090	-1.503 ± 3.706
		Grid Logit	-7.984 ± 8.479	-3.795 ± 8.878	2.021 ± 0.575
		Star Logit	-18.629 ± 5.279	-15.728 ± 7.512	-25.136 ± 2.219
	7	Lollipop Logit	2.700 ± 0.000	-3.375 ± 2.803	-3.732 ± 0.426
		Wheel Logit	-18.050 ± 0.000	-11.131 ± 14.287	-1.052 ± 5.294
		Grid Logit	-5.612 ± 0.000	-4.683 ± 8.943	1.356 ± 2.642
		Star Logit	-21.298 ± 0.000	-22.799 ± 8.634	-24.479 ± 4.387
	8	Lollipop Logit	7.224 ± 0.000	-2.637 ± 1.622	-2.184 ± 3.039
		Wheel Logit	-15.080 ± 0.000	-19.483 ± 17.827	-13.259 ± 5.576
		Grid Logit	-3.129 ± 0.000	1.137 ± 6.363	-6.440 ± 5.380
		Star Logit	-19.795 ± 0.000	-18.262 ± 9.027	-14.606 ± 4.264

Table 9. Logits for explanation graphs generated by MIPExplainer for the ”Grid” and ”Lollipop” classes of the shapes dataset, averaged over 5 runs with random initial solutions

			Shapes Output Logits (2)		
			MIPExplainer	GNNInterpreter	XGNN
Class	# Nodes				
Star	5	Lollipop Logit	-16.772 ± 0.000	-12.575 ± 12.292	-5.020 ± 1.008
		Wheel Logit	-67.715 ± 0.000	-42.824 ± 40.005	2.464 ± 5.862
		Grid Logit	-72.108 ± 0.000	-31.701 ± 46.376	7.990 ± 0.654
		Star Logit	12.894 ± 0.000	-8.555 ± 20.510	-30.609 ± 2.926
	6	Lollipop Logit	-13.064 ± 0.000	-3.980 ± 2.651	-4.822 ± 1.125
		Wheel Logit	-64.317 ± 0.000	-21.691 ± 9.404	5.352 ± 3.101
		Grid Logit	-48.784 ± 0.000	-3.585 ± 12.975	4.037 ± 2.655
		Star Logit	12.183 ± 0.000	-15.661 ± 2.185	-29.810 ± 3.144
	7	Lollipop Logit	-11.148 ± 0.000	-2.507 ± 2.511	-1.469 ± 2.153
		Wheel Logit	-61.412 ± 0.000	-22.296 ± 21.034	-9.213 ± 5.345
		Grid Logit	-33.850 ± 0.000	-3.210 ± 13.723	-6.549 ± 8.455
		Star Logit	13.072 ± 0.000	-18.741 ± 8.891	-18.000 ± 4.776
	8	Lollipop Logit	-10.722 ± 0.000	-3.702 ± 1.427	-1.924 ± 1.894
		Wheel Logit	-58.630 ± 0.000	-7.587 ± 15.924	-8.287 ± 9.256
		Grid Logit	-23.479 ± 0.000	-1.624 ± 8.923	-11.054 ± 12.549
		Star Logit	14.536 ± 0.000	-21.715 ± 8.531	-18.099 ± 8.888
Wheel	5	Lollipop Logit	-6.180 ± 0.000	-7.290 ± 7.960	-5.957 ± 0.275
		Wheel Logit	13.062 ± 0.000	-42.230 ± 28.326	8.335 ± 2.176
		Grid Logit	10.323 ± 0.000	-3.870 ± 18.922	8.057 ± 0.745
		Star Logit	-40.319 ± 0.000	-12.452 ± 10.727	-34.470 ± 2.362
	6	Lollipop Logit	-5.616 ± 0.000	-4.918 ± 4.503	-3.865 ± 1.889
		Wheel Logit	6.921 ± 0.000	-18.368 ± 31.653	1.860 ± 6.420
		Grid Logit	2.541 ± 0.000	3.615 ± 6.467	3.038 ± 3.642
		Star Logit	-29.458 ± 0.000	-22.223 ± 14.202	-27.369 ± 5.667
	7	Lollipop Logit	-5.328 ± 0.402	-4.489 ± 2.121	-1.477 ± 4.376
		Wheel Logit	7.040 ± 1.338	-5.192 ± 15.185	-8.111 ± 10.727
		Grid Logit	2.960 ± 1.279	5.168 ± 3.163	-4.639 ± 12.694
		Star Logit	-29.519 ± 1.233	-24.700 ± 9.701	-20.903 ± 8.152
	8	Lollipop Logit	-4.147 ± 1.128	-2.193 ± 0.699	-1.513 ± 2.686
		Wheel Logit	4.468 ± 3.188	-5.022 ± 6.210	-11.867 ± 4.893
		Grid Logit	1.505 ± 0.752	-5.701 ± 13.464	-3.223 ± 12.412
		Star Logit	-27.293 ± 1.755	-20.266 ± 6.735	-17.747 ± 6.596

Table 10. Logits for explanation graphs generated by MIPExplainer for the ”Star” and ”Wheel” classes of the shapes dataset, averaged over 5 runs with random initial solutions