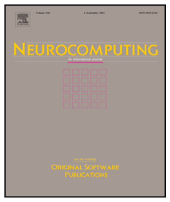
Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www elsevier com/locate/neucom



Explaining Graph Neural Networks with Mixed-Integer Programming

Blake B. Gaines [a], Chunjiang Zhu [b], Jinbo Bi [a],*

[a] *Department of Computer Science University of Connecticut Storrs 06268 CT USA*
[b] *University of North Carolina Greensboro Greensboro 27412 NC USA*

ARTICLE INFO

Communicated by J. Cao

Dataset link: https://github.com/blake-gaines/MIPExplainer

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 stable explanations on both synthetic and real-world datasets. Code is available at https://github.com/blake-gaines/MIPExplainer.

1. Introduction

Graph neural networks (GNNs), such as graph convolutional networks (GCN) [1], GraphSAGE networks [2], and graph attention networks (GAT) [3], provide a family of powerful tools for modeling graphs that learn from both the features contained in nodes and edges and the structure of the graph itself. However, the limited explainability of GNN prediction makes it impossible to justify the use of GNNs 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, such as ProtGNN [4], GIB [5], and GraphChef [6]. The disadvantage of this approach is that changing the GNN itself to enforce explainability restricts users' choice of GNN architectures and does not allow for the explanation of already-trained GNNs. 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 instance-level explanation, which aims to explain the reasoning behind individual predictions, has been extensively explored for GNNs (see surveys from [7–9]), but fewer methods exist to explain the overall patterns used by GNNs to differentiate classes.

1.1. Related work

GNN explanation

At least six categories of instance-level GNN explanations have been proposed so far, including those based on gradients/features [10,11], perturbations [12–16], surrogates [17], generation [13,18], decomposition [19–21], and counterfactuals [20]. 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 common subgraphs shared by a significant portion of the individual explanations. A more recent technique, GLGExplainer [22], finds smaller components of the extracted instance explanations that can be used to build logical expressions consistent with the overall GNN's predictions. However, these methods that combine instance-level explanations may be limited by the scope of the training instances, and can be influenced by bias in the dataset. Generating explanations directly from the GNN model is

* Corresponding author.
E-mail address: jinbo.bi@uconn.edu (J. Bi).

https://doi.org/10.1016/j.neucom.2025.130214
Received 12 July 2024; Received in revised form 10 March 2025; Accepted 5 April 2025
Available online 19 April 2025

more faithful, and provides deeper insights into the degree of bias in the model itself.

Our proposed method, like many methods for explaining GNNs at the model level, focuses on generating graphs that reflect the representative patterns of individual classes differentiated by a GNN classifier. PAGE [23], GCExplainer [24], MAGE [25], and the method described in [26] are global explanation methods that focus on explaining GNNs via concept generation, an approach that only considers graphs and subgraphs from the training data. For example, PAGE begins by clustering training graphs in each class based on the features extracted by the message-passing layers of the GNN (i.e. the training graph embeddings), and then finds common subgraphs within the graphs comprising each cluster. It then applies the GNN to these subgraphs and finds the ones that maximize their respective graph's class probability as the class-level explanations. The other methods listed above also follow a similar procedure, searching for subgraphs of the training graphs using criteria based on their embeddings from the GNN. Among these methods, we choose to compare with PAGE, because the criteria it uses to select representative subgraphs is the most comparable to ours. On the other hand, XGNN [27] is a well-established approach that explains the model by finding graphs that maximize the model output for a target class. It serves as the baseline in several recent papers that focus on similar objectives [22,23,28–30]. XGNN trains a second neural network by reinforcement learning to generate graphs that obey explicit, hand-crafted generation rules (e.g. a maximum node degree) while maximizing the original GNN's prediction for a specific class. Similarly, D4Explainer [30] trains a separate denoising model and generates explanation graphs through a diffusion process. GNNInterpreter [29], GDM [31], and GraphEx [32] avoid training a second neural network by assuming that the graphs in the dataset are sampled from 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 GNN's 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 follows a similar strategy to GNNInterpreter, but attempts to learn the conditional probabilities of graphs in the dataset given the classes predicted by the model. GDM minimizes the estimated maximum mean discrepancy between the distribution of embeddings of training graphs and the distribution of embeddings of the learned explanations, and also employs regularizers to keep the explanations in-distribution. KnowGNN [33] also learns the latent parameters of categorical distributions to extract graph features, but additionally trains a second neural network that learns to mask edges as part of generating explanations.

There are several common problems among existing approaches to model-level GNN explanation. First, they often have many hyperparameters (e.g. learning rate, regularizer weights) that can change the quality of generated explanations, and generating a high-quality explanation may require setting them within a specific range of values. Because there is no single metric to quantify global explanation quality, the performance of these methods is mostly evaluated through visualization and manual inspection of the results. Therefore, it is impossible to objectively compare the results from different hyperparameter settings and determine which explanation to use, and performing hyperparameter optimization qualitatively is similarly difficult without prior knowledge of a ground-truth explanation. Second, all of the methods rely on stochastic gradient descent algorithms to optimize parameter values used to generate explanations. A stochastic optimization algorithm converges to a critical point only in expectation, so in individual runs it may stop anywhere within a large neighborhood of the desired optima. When a maximum number of iterations is set, the final explanation's objective value might be far away from the globally optimal value. Thus, explanations generated in different runs with different starting values for the parameters vary significantly, even with the same choices of hyperparameters. Third, these methods do not have theoretical guarantees of achieving optimality, or even bound the gap in optimization objective between their solution and an optimum. Due to their lack of algorithmic stability and their inability to guarantee solution quality, existing methods cannot explain a GNN in a consistent manner across multiple runs. This variability is undesirable, because the distribution of generated explanations is unknown relative to the global optima for a given interpretation objective. As a result, it is questionable whether the generated explanations contain the information that the objective function was designed to extract. We argue that the degree of consistency among explanations generated in successive runs of an explanation method is an important measure of their performance, even though it does not directly measure the quality of the generated explanation graphs themselves.

Mixed-integer programming for deep neural networks

Mixed-integer programming (MIP) has been used to encode ReLU networks to solve verification problems [34–36], inverse design problems [37,38], and to generate instance-level explanations for ReLU networks [39]. MIP defines a constrained optimization problem where some of the decision variables must take integer values. MIP problems are commonly solved through branch-and-bound, where the original problem is decomposed into subproblems defined on partitions of the feasible solution set and solved recursively, creating a search tree. Note that for a maximization problem, removing any constraints and solving the relaxed problem will yield solutions that bound the original problem's optimal solution from above. Thus, an upper bound on any of the MIP subproblems can be found by relaxing the integrality constraints and solving the resulting system. Then, large subtrees can be pruned if the upper bound at the root node of the subtree is less than the objective value of a known solution, improving the tractability of the search.

The approach in [40] successfully encodes GCN and GraphSAGE layers into MIPs, but MIPs encoding larger GNNs often remain intractable to solve, primarily due to the issue of symmetry. GNN layers are equivariant with respect to the ordering of the nodes in the graph, and permutation invariant pooling layers are generally used to make graph-level predictions. As a result, the worst-case number of possible representations for an optimal graph defined in terms of a node feature matrix and adjacency matrix grows exponentially with respect to the number of nodes in the graph. This creates practical problems when solving a GNN's MIP using branch-and-bound, as any subtree containing an equivalent representation of the optimal solution cannot be pruned. This problem can be addressed by adding constraints to reduce the number of feasible solutions in the equivalence class of each graph, as described in [41]. Our proposed approach benefits from these works by combining existing formulations of GNN layers with symmetry breaking techniques, creating an explanation method that is both tractable and stable.

1.2. Main contributions

We propose a new explanation method based on MIP, which we call MIPExplainer, to find graph structures or subgraphs that explain GNN models from multiple perspectives. We design three novel objective functions to respectively discover subgraphs that the GNN model reports as the most representative of a class (reaching the highest class probability), the most difficult-to-classify (on the boundary between two classes), and contrastive class-specific explanations in a range of scenarios. We further propose a new quantitative metric for explanation methods to assess their stability by measuring the dissimilarity of the generated explanations across multiple runs. While any appropriate metric can be used to measure the (dis)similarity of these graphs [42], we employ graph edit distance [43], 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 node 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, as only the number of nodes of the explanation graph needs to be specified, and empirically generates recognizably in-distribution explanations without the need for additional regularization terms. This facilitates the application of our approach and mitigates the effects of bias when analyzing the results.
3. It can verify the optimality of results with respect to the explanation objective. In cases where this is intractable, MIPExplainer can place an upper bound on the optimal solution, guaranteeing the quality of the generated explanation.

2. Our approach MIPExplainer

Our model-level explanation seeks to optimize an input graph $G = \ X\ A)$ with respect to some explanation objective defined in terms of the GNN's output, 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. Like XGNN, which searches for a graph to maximize the predicted probability of a certain class, and GNNInterpreter, which maximizes the logit of a certain class, we generate unweighted graphs where A is binary adjacency matrix ($a_{ij} \in \{0\ 1\}$), so that $a_{ij} = 1$ indicates there is an edge between nodes i and j, and X is the node feature matrix where x_{ij} represents feature j of node i. Let a GNN realize a function $f_c\ G\ \theta)$ that maps G to the (possibly unnormalized) probabilities of several classes indexed by c, where θ contains all trainable parameters in the GNN. To train a GNN, a set of graphs is given and θ needs to be determined, whereas in the explanation setting, θ has been fixed and we optimize $G = \ X\ A)$ in terms of an explainability objective, for example to maximize $f_c\ G\ \theta)$.

To formulate the MIP, we examine the calculation performed by a multi-layer GNN. Each layer of the GNN imposes a set of constraints on X and A in the MIP. Consider how node features are updated in the first GNN layer of a GraphSAGE model. Using the features of node i in G, represented by the row vector x_i, and its neighbors, represented by A_i or the ith row of A, the node's updated representation x'_i is computed as $x'_i = \phi\ x_i W_1 + A_i X W_2 + b)$ where W_1, W_2, and b are part of the fixed parameters in θ, and ϕ is an activation function. The nonzero a_{ij} in A_i indicates that node j is a neighbor of node i, so the term $A_i X$ aggregates the features of all of node i's neighbors. By adding new decision variables $X' = \ x'_i)$, this equation forms a constraint on X and A. The second GNN layer is then encoded in exactly the same way as the first, but now to constrain X'. This constraining process propagates all the way to the final layer that calculates $f_c\ G\ \theta)$, at which point the decision variables representing the final output can be used to directly express an explainability-related objective function. All of the constraints formed at each layer eventually back-propagate to constrain X and A. Additional constraints on nodes and edges can be included to ensure that a connected graph is generated. In the subsequent sections, we provide a full description of our MIP formulation including the design of objective functions and the formation of constraints that encode the GNN.

Through a sequence of algebraic operations, we ensure our MIP encodings of GNNs have both linear objective functions and linear constraints in terms of the decision variables, so our MIP is actually a Mixed-Integer Linear Program (MILP). Linearity greatly reduces the complexity of the optimization, and well-studied methods exist to solve continuous relaxations of the MILP problem and establish upper bounds on solution quality with time complexity that is polynomial in terms of the number of decision variables and constraints [44].

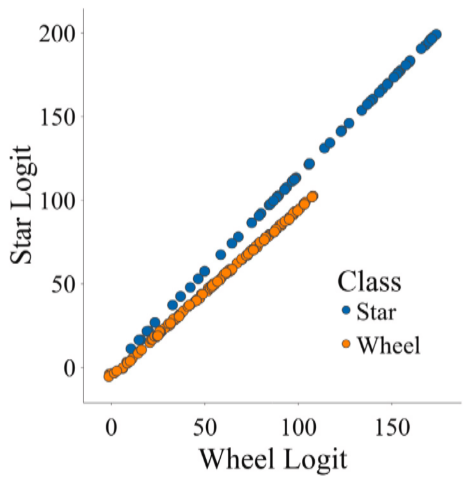


Fig. 1. Logits of Star and Wheel Graphs in the Shapes Dataset.

2.1. Objective functions

Existing model-level GNN explanation methods define objectives to represent the knowledge that a GNN has learned about each class in the dataset. To find a representative graph for a class label, some of the existing methods (e.g., GNNInterpreter) do not directly maximize the class probability (which is computed through the softmax of logits). Rather they maximize the class logit (while ignoring the logits of other classes in the denominator of the softmax). However, we argue that maximizing a single logit may lead to wrong explanations. For instance, after training a GNN to differentiate between two classes of graphs (stars and wheels, a task defined in [29], see Section 4 for details), each graph receives two logits, one for each class, and the graphs are classified to the class of the larger logit. We plot the two logits for the training graphs in Fig. 1. The points above the diagonal line are correctly classified as Stars whereas those below the line are Wheels. All graphs are correctly classified but the maximum logit in the wheel class is actually attained by a correctly classified star graph (the rightmost blue point). Thus, simply maximizing the logit for wheels will produce a representative graph for the star class.

Our objectives are designed to examine the various differences between logits. Although we focus on discussing how the proposed MIP finds class-representative graphs for explanations in order to compare against extant methods, we also demonstrate that MIP can be used to identify other sorts of explanations for the GNN with appropriately-designed objective functions. Eq. (Class-Representative Explanation) encourages class-representative features by maximizing the difference between the target class and the maximum of the other classes, a similar objective to other baseline methods. Eq. (Boundary Explanation) is a novel explanation objective, which aims to find features representative of graphs at the decision boundary between two classes. Eq. (Paired Explanation) is another novel objective based on instance-level counterfactual explanations, which jointly optimizes two graphs to be minimally different with a maximal change in the prediction of a certain class.

2.1.1. Finding class representatives

To accurately find class-discriminative information, we should maximize the difference between the logit of the target class and the logits of the other classes. Maximizing the normalized probability, as done by XGNN, is possible but can lead to numerical instability 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 for only the target class. However, it is possible that an optimal solution simply minimizes 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 maximize the difference between the logit of the target class and the maximum logit of the other classes. According to our observations, this approach is more effective, so we propose the following objective function:

$$\max_{G} \;\; O\; G) = \left(f_c\; G\; \theta) - \max_{i\; c} f_i\; G\; \theta)) \right) \tag{Class-Representative Explanation}$$

where f_i denotes the ith logit (or the ith output of the GNN before the application of the softmax function for classification).

2.1.2. Finding boundary explanations

For classification tasks, it can be insightful to illustrate what kind of graph data the GNN has difficulty to classify, i.e., those graphs that are located on the separation boundary (i.e., $f_{c_1}\; G\; \theta) = f_{c_2}\; G\; \theta))$ between two classes c_1, c_2. In particular, we identify the one that best represents a class under the condition of within a small distance d from the decision boundary between the class and another class. It corresponds to minimizing the following objective:

$$\max_{\{G : |f_{c_1}\; G\; \theta) - f_{c_2}\; G\; \theta)| < \delta\}} \;\; O\; G) = \left(f_{c_1}\; G\; \theta) - \max_{i \notin \{c_1\; c_2\}} f_i\; G\; \theta)) \right) \tag{Boundary Explanation}$$

where c_1 and c_2 can represent any of the classes. The motivation for this objective function is to isolate features that may be used by the model to distinguish between the two classes and separate them from the rest. By maximizing the prediction of the chosen classes while minimizing the prediction of the other classes, this objective function distills the features that positively indicate these two classes. Proving that this equation has no solutions (e.g. if the two selected classes do not share a boundary or no graphs lie within a specified distance of it) may also reveal information about the model's behavior. This would be done by replacing the MIP objective with the distance of the input graph from the decision boundary and minimizing. If the objective value to an optimal solution of this problem is greater than δ, the equation above has no solutions. Note that in the case of 2 classes, the inner maximum is taken over an empty set and can be discarded.

2.1.3. Finding similar graphs with maximal changes in prediction

Simultaneous optimization over multiple graphs can significantly expand the number of possible explainability-related objectives. We show the advantage of this new approach by examining the pattern learned by the GNN that maximally shifts its decision with respect to a certain class. To find this pattern, we can find a pair of graphs $G_1\; G_2$ that are similar but cause the biggest change in the model's prediction of a certain class, and then look at the patterns created and destroyed by the changes between them (e.g. cycles being created or broken, more edges between nodes with certain features, etc.). This may be more informative than a single representative explanation, as it is not always clear which of the patterns a single graph has (or possibly more importantly, *does not have*) that are causing the model to behave in a certain way. The magnitude δ of the change between the two graphs can be varied, with larger values allowing for more complex/comprehensive changes to the graph. If we quantify the change in model prediction by the difference in the target class's predicted probabilities between G_1 and G_2 (i.e. $f\; G_1\; \theta) - f\; G_2\; \theta))$, it might be difficult to see the existence or absence of a certain pattern, because changes in probabilities get exponentially smaller for negative logits. Therefore, we choose to quantify the prediction change by the difference between the logits of the target class and the maximum of the other classes. This way it is equally advantageous to add patterns indicative of a different class to G_2 (or subtract them from G_1) as it is to add patterns indicative of the target class to G_1 (or subtract them from G_2). The explanation objective is then realized as follows:

$$\max_{\{G_1\; G_2 :\;\; G_1 - G_2\;\; < \delta\}} \;\; O\; G_1\; G_2) = \left(\left(f_c\; G_1\; \theta) - \max_{i\; c} f_i\; G_1\; \theta) \right) - \left(f_c\; G_2\; \theta) - \max_{i\; c} f_i\; G_2\; \theta) \right) \right) \tag{Paired Explanation}$$

Here, can be any graph distance metric, e.g. graph edit distance. Note that this differs substantially from aforementioned work using MIP for neural network verification, which is concerned with the distance between specific data points and the decision boundary (i.e. the minimum perturbations required to change their predicted classes). As opposed to optimizing the perturbation for a fixed input, our approach searches for the original and perturbed inputs at the same time, which removes any influence from the data and maintains sole reliance on the model itself for generating explanations. The two graphs do not have to be predicted as the target class, and may not resemble the data in that class at all, only their difference is relevant to the explanation.

2.1.4. Regularization

Many approaches incorporate regularizers into their explanation objectives to encourage generated graphs to be within the distribution of the training data. For regularization in XGNN, the explanation generator is penalized during reinforcement learning 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 is penalized for being farther from 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 make any guarantees. Furthermore, 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 qualitatively judging a large number of generated graphs increases the likelihood of mistakenly accepting spurious explanations. Therefore, we do not apply any regularization in the objective function during our experiments. If desired, MIPExplainer is able to incorporate regularizers such as the one used by GNNInterpreter, although this would require quadratic terms in the constraints and objective. Constraints for keeping explanations in-distribution, such as those used to penalize XGNN's explanation generator, can be directly encoded as constraints.

2.2. Constraints

All of the above objective functions will be optimized subject to the same set of constraints described in this section, with some constraints being duplicated when optimizing over multiple graphs as in Eq. (Paired Explanation). We first make a simple assumption that the node features are bounded by a constant M in magnitude. We do not make any other assumptions about the node features or their distribution. We require the number of nodes in the explanation n to be fixed in advance, and this is MIPExplainer's only hyperparameter.

From the range of existing GNN layers, we 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

$$X' = \sigma\; XW_1 + \text{Aggregation}\; A\;\; X)W_2 + b). \tag{1}$$

The aggregation function is generally a permutation-invariant function which combines the sets of feature vectors of each node's neighbors into single row vectors. For example, it can be an element-wise sum, so that Aggregation $A\;\; X) = AX$. Our encoding of GraphSAGE layers will follow the work of [40].

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 layers with ReLU activations. In total, there are $\ell = \ell_c+1+\ell_f$ layers (indexed as ℓ_i $1 \le i \le \ell$). 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 a number of intermediate decision variables to our formulation. In every layer, $\Phi^{(i)}$ represents the output of layer i before the application of an activation function, and $X^{(i)}$ represents ReLU $\Phi^{(i)})$, the output of layer i. In the GNN layers $\Phi^{(i)}$ and $X^{(i)}$ are matrices with a row vector for each node's updated representation, and afterwards for layers $i > \ell_c$ they represent vectors containing the entire graph's pooled representations.

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

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + AX^{(i-1)}W_2^{(i)} + b^{(i)}. \tag{2}$$

Note that the second term includes the multiplication of the adjacency matrix A and the node feature matrix $X^{(i-1)}$ from the early layer. Thus, if we encode this equation directly, we will have quadratic terms in the relaxation subproblems. There are several ways to perform the linearization of quadratic terms consisting of a continuous variable and a binary variable, and we will describe one such method here using big-M constraints [45]. For a given binary variable $a \in A$ and a variable $x \in X^{(i)}$ bounded by $M^{(i)}$, let $e = a\ x$ be a new intermediate decision variable constrained as follows:

$$-M^{(i)}a \le e \le M^{(i)}a \tag{3}$$

$$x - M^{(i)}(1-a) \le e \le x + M^{(i)}(1-a). \tag{4}$$

If $a = 0$, then Eq. (3) will force e to be 0. On the other hand, if $a = 1$, then Eq. (4) will force e to be equal to x. Let $E^{(i)}$ be a matrix that encodes $AX^{(i)}$, where each entry is the sum of the corresponding e's Eq. (2) can now be rewritten:

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + E^{(i)}W_2^{(i)} + b^{(i)} \tag{5}$$

While this step adds extra decision variables and constraints, the resulting system is linear, so it becomes faster to optimize.

For the pooling layer $i = \ell_c + 1$ (with representing a vector of 1s):

$$\Phi^{(i)} = {}^T X^{(i-1)} \tag{6}$$

and for the fully connected layers ($\ell_c + 1 < i \le \ell$):

$$\Phi^{(i)} = X^{(i-1)}W_1^{(i)} + b^{(i)}. \tag{7}$$

For all layers except the pooling and output layers ($0 < i \le \ell - 1$ i $\ell_c + 1$), we must constrain $X^{(i)}$ based on the corresponding $\Phi^{(i)}$. We add the slack variable matrix $B^{(i)}$ in Eq. (8) in order to encode the ReLU operation, where $B^{(i)}$ represents $ReLU\ -\Phi^{(i)})$, the negative components of each element of $\Phi^{(i)}$ discarded by the ReLU. $Z^{(i)}$ are binary decision variables indicating the truth value of $\Phi^{(i)} > 0$ elementwise. Note that 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.

$$X^{(i)} - B^{(i)} = \Phi^{(i)} \tag{8}$$

$$X^{(i)} \le MZ^{(i)} \tag{9}$$

$$B^{(i)} \le M(1 - Z^{(i)}) \tag{10}$$

$$0 \le X^{(i)}, B^{(i)} \le M \tag{11}$$

$$Z^{(i)} \in \{0, 1\} \tag{12}$$

At the indices where $Z^{(i)} = 0$, $\Phi^{(i)}$ is negative, Eq. (9) ensures that the corresponding elements of the layer's activation in $X^{(i)}$ are 0. Where $Z^{(i)} = 1$, $\Phi^{(i)}$ is positive, so Eq. (10) ensures that the corresponding elements of the negative component $B^{(i)}$ are 0. In both cases, Eq. (8) ensures that $\Phi^{(i)}$ equals its positive component minus its negative component. For the pooling layer and output layer, we simply have that $X^{(i)} = \Phi^{(i)}$.

In order to encode the maximum output of the non-target classes from Eq. (Class-Representative Explanation), for the last layer we introduce the single decision variable y and constrain it so that $y = \max_{k \ne c} X_k^{(\ell)}$. Note that $X^{(\ell)}$ at the output layer is a vector containing the class logits. To ensure the constraints are linear, we also introduce a vector of decision variables d in which each element is an indicator representing whether the corresponding element of $X^{(\ell)}$ is the maximum element in the output of layer ℓ when disregarding the target class, i.e., dimension j of d is 1 if $j = \text{argmax}_{k \ne c} X_k^{(\ell)}$ and 0 otherwise. Then, y and d are constrained as follows:

$$y \ge X_{\setminus c}^{(\ell)} \tag{13}$$

$$y \le X_{\setminus c}^{(\ell)} + (\max(U_{X_{\setminus c}^{(\ell)}}) - L_{X_{\setminus c}^{(\ell)}})(1 - d) \tag{14}$$

$$\sum_j d_j = 1,\ d_j \in \{0, 1\} \tag{15}$$

where $L_{X_{\setminus c}^{(\ell)}}$ and $U_{X_{\setminus c}^{(\ell)}}$ represent matrices of element-wise lower and upper bounds for the decision variables in $X^{(\ell)}$ excluding the one for class c. Since X, A, and $Z^{(i)}$ are explicitly bounded, these bounds can be derived via constraint propagation (further discussed in Section 3). Eq. (13) ensures that $y \ge \max_{k \ne c} \Phi_k^{(\ell)}$. Eq. (14) ensures $y \le \max_{k \ne c} \Phi_k^{(\ell)}$, as one element in the right-hand side will be exactly equal to the value of $X_{\setminus c}^{(\ell)}$ at the index where $d = 1$, and the rest will be values guaranteed to be larger than it. We know that d *must* indicate the correct maximum, as otherwise there would be a lower bound for y in (13) (the correct maximum) greater than an upper bound for y in (14) (the element identified by d), making the system inconsistent. The constraints in (15) ensure that d is a one-hot vector.

2.3. The final MIP formulation

The overall MIP has decision variables A, $X^{(i)}$, $\Phi^{(i)}$, $Z^{(i)}$, $B^{(i)}$, y, d, $e_{rst} = a_{rs}x_{st}^{(i)}$ for $1 \le r, s \le N$ and $1 \le t \le d$. The feasible region of these variables is defined by the constraints in (3)–(15), which specify each intermediate decision variable in terms of A and X alone.

In order to maximize Eq. (Class-Representative Explanation), we can simply set our objective function to $X_c^{(\ell)} - y$. Since we have a linear objective function and all linear constraints when integrality is relaxed, this is a MILP. In the case of Eq. (Boundary Explanation), additional constraints can be added to ensure that solutions lie on the decision boundary. Specifically, to constrain the L1 distance, we add the following constraints:

$$X_{c_1}^{(\ell)} - X_{c_2}^{(\ell)} \le \delta \tag{16}$$

$$X_{c_2}^{(\ell)} - X_{c_1}^{(\ell)} \le \delta \tag{17}$$

When optimizing over two graphs in Eq. (Paired Explanation), several modifications to the MIP described in the previous section are necessary. A simple approach is to increase the size of the adjacency matrix and constrain the appropriate elements to ensure that it is block-diagonal. The GNN layers can then be encoded in the same way, but the pooling layer must be modified so that features from nodes in separate graphs (i.e. with indices corresponding to different blocks of the adjacency matrix) are aggregated into separate representations for the different input graphs. After this, separate decision variables are used to encode the outputs of the fully connected layers operating on the representations of the different graphs. Finally, the constraints in Eqs. (13)–(15) can be duplicated for each set of outputs to allow this objective function to be represented as a linear combination of decision variables.

2.4. Additional constraints on A and X

Additional constraints can be placed on A and X when generating explanations. We specifically employ the following constraints in our experiments. When the input space contains graphs with one-hot features, we constrain the sum of each row of $X^{\ 0)}$ to be equal to 1 to ensure proper encoding. When the input graph is undirected, we can add the constraints $a_{ij} = a_{ji}$ for all $i\ j$ with $0 \leq ij < n$ and $i < j$ to ensure symmetric connections. We prevent self-loops in the explanation by constraining the diagonal elements of A to be 0.

Because GNN layers are permutation invariant, reducing the number of equivalent representations for each graph can greatly improve the tractability of branch-and-bound in many situations by reducing the number of feasible representations of the optimal solution in different branches of the search tree. We incorporate the three types of *symmetry breaking* constraints proposed by [41], which all work to limit feasible permutations of the node ordering defined by A and X, which reduces number of equivalent representations of graphs in the search space. The first set of constraints imposes a partial ordering on the graph nodes by ensuring each node has an edge to at least one other node with a smaller index, which also ensures that the explanation graphs are connected. The second set of constraints imposes a lexicographic ordering on the adjacency matrix. The third group of constraints ensures that the node receiving the first index also has the highest number of neighbors. Their work shows that these three constraints are compatible and do not exclude any graphs from the search space.

2.5. Generalizing to other 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 (6) as $\boldsymbol{\Phi}^{\ i)} = \ ^{T}\boldsymbol{\Phi}^{\ i-1)}\frac{1}{N}$ for $i = \ell_c + 1$. If mean aggregation is used in Eq. (1), we could use 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 $D^{\ i)}$, the constraint of $\quad ^{T}A)D^{\ i)} = AX$ can be included in the model. The multiplication of D by elements of A on the left-hand side of this expression can be linearized as previously described. Now, constraint (2) can be changed to:

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

In passing layer from a Graph Isomorphism Network [46], updated node representations are calculated as $X' = h\ \ A + \ 1 + \epsilon)I)X)$ where 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, which can be encoded with constraints similar to those in Eqs. (3)–(12). The work in [40] also describes a way to encode GCN layers with linear constraints.

2.6. The optimization algorithm

We employ a standard branch and bound procedure [47], along with cutting planes and heuristics, to find a globally optimal solution efficiently. In Algorithm 1 we describe the most basic form of this approach for solving our MIP described in the previous section, which is represented by its set of constraints C and objective function o. We obtain an initial solution at the root of a search tree by choosing an initial graph $G_0 = \ \ X_0\ A_0)$ and applying the GNN to obtain initial values for all the intermediate variables.

We start by finding the optimal solution of the continuous relaxation (i.e. the MIP with the integrality constraints removed) of the MIP problem (line 7), which can be done quickly using the simplex method. The objective value for the resultant optimal solution z^* serves as an upper bound U to the original problem with the additional integrality constraints. If this solution z^* happens to also satisfy all of the integrality constraints of the original MIP, then it is an optimal solution to the original problem rather than just its continuous relaxation, and we can stop since the simplex algorithm guarantees that there are no other solutions with better objective values. If any integer variable takes a fractional value in the solution of the continuous relaxation z^*, for instance, $v := z^*_v$ where z^*_v is a fraction, we branch the MIP on v by splitting the original problem into two subproblems with the extra constraints of $v \leq \lfloor z \rfloor$ or $v \geq \lceil z \rceil$ respectively (lines 14–15), which partition the search space of the original problem. The optimal solution to the original problem will then 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 at the root. After branching and adding the new constraints, the two MIP subproblems together consider all the same integral solutions as the original MIP, but the regions considered by their linear relaxations no longer include the area where $\lfloor z \rfloor < v < \lceil z \rceil$. As a result, the maximum of the children's upper bounds is still an upper bound for the parent MIP, but smaller than or equal to the upper bound provided by the parent MIP's linear relaxation. As a result, our upper bound on the original MIP strictly decreases as we explore more of the search tree. After we branch enough times, the added constraints eventually ensure that the continuous relaxation of the system has an integral solution, at which point we have reached a leaf in the search tree. Each time we reach a leaf, the solution it contains meets all of the constraints of the original problem, so its objective value also serves as a lower bound for the optimal solution to the original problem. Our tightest lower bound L increases as we find leaves with better and better solutions (lines 10–12). The upper and lower bounds tighten as we search the tree, and eventually meet, at which point the solution z for which $O\ z) = L$ proven to be the optimal. Crucially, we do not need to explore the entire search tree for this to happen. If the continuous 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 do not have to search the branch rooted

Algorithm 1 MIP Branch and Bound Procedure

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

at that node because no new optimal solution can lie in that subtree (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 original MIP. In our experiments, we use Gurobi Optimizer [48], a fast and efficient solver, to find an optimal solution to the MIP proposed in Section 2.3. While the theoretical complexity of this algorithm is exponential, the average complexity is significantly lower in practice, making it possible to find and guarantee an optimal solution in many situations.

3. 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 initial 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 the ones used to break symmetry, any graph used as a warm start must be converted into the canonical form that also satisfies these constraints. When optimizing Eq. (Boundary Explanation), we may not have an initial graph within the specified distance of the decision boundary. In this case, we can start by omitting the associated constraints and minimizing the distance to the decision boundary until they are satisfied. This solution can then be used to warm-start the full MIP.

Although a single, large number M 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 element-wise lower bound vector x_L and upper bound vector x_U, we can get upper and lower bounds on the output of a linear layer $x' = Wx + b$:

$$\begin{aligned} x'_L &= \text{ReLU}\ W)x_L + \text{ReLU}\ -W)x_U + b \\ x'_U &= \text{ReLU}\ W)x_U + \text{ReLU}\ -W)x_L + b. \end{aligned} \tag{18}$$

Given the bounds on the explanation graph (i.e., X and A), we can propagate the bounds forward through the GNN to iteratively bound the set of decision variables for each hidden representation. Bounds for the outputs of ReLU activation layers are the same as those for 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. This strategy of constraint propagation has been explored and validated in [36].

Floating-point precision errors can lead to serious problems for MIP solvers. 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 problem emerges when the weights of GNNs become very small, an effect often produced by regularization. However, we found that weights below a certain threshold (e.g., 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 weight regularization during training improved MIP solution times.

4. Experimental evaluation

We use two synthetic datasets and four real-world datasets to evaluate our method: Is_Acyclic, Shapes, MUTAG, NCI1, IMDB-BINARY, and REDDIT-BINARY (see Table 1). These datasets have all been previously used to compare GNN explanation methods. The **Is_Acyclic** dataset comes from XGNN's experiments, and has two classes consisting of Cyclic and Acyclic graphs of various types. The Cyclic class includes graphs like grids, single cycles, and wheels, while the Acyclic class includes 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 five 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 [49] 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 [50], 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. NCI1 [51] is an additional molecule dataset that comes from a non-small cell lung human tumor cell line growth inhibition assay. IMDB-BINARY [52] contains networks of actors participating in movies, with edges linking costars. REDDIT-BINARY [52] is built from comment threads with nodes as users and edges between users where at least one has replied to the other, with graphs labeled according to whether the thread came from a question/answer subreddit or a discussion-based subreddit. As IMDB-BINARY and NCI1 do not have obvious ground-truth explanations to distinguish their classes, we will discuss quantitative but not qualitative results for these datasets.

To assess the stability of different explanation methods (i.e., to quantify the variation among generated explanations of each method), 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 [43], is the minimum number of graph edit operations (vertex/edge insertions/deletions/substitutions) needed to transform one graph into another. A lower average graph edit distance indicates a more stable explanation method.

4.1. Setup and implementation

Each dataset is randomly split into a training set (80%) for training a GNN model and a test set (20%) for measuring its accuracy. The performance of our GNN models trained on the six datasets is reported in Table 2, and is comparable to those previously used to test other GNN explanation methods. We compare MIPExplainer with the five most relevant approaches: XGNN, GNNInterpreter, PAGE, D4Explainer, and KnowGNN. To run experiments with these comparison methods, we use the implementations provided by their respective authors: XGNN in DIG[1] [7],GNNInterpreter[2] and PAGE,[3] D4Explainer,[4] and KnowGNN.[5] All methods are run with the same hardware including 32 Processors,

[1] https://github.com/divelab/DIG
[2] https://github.com/yolandalalala/GNNInterpreter
[3] https://github.com/jordan7186/PAGE
[4] https://github.com/Graph-and-Geometric-Learning/D4Explainer
[5] https://github.com/lxf770824530/KnowGNN

Table 1
Dataset summary and statistics.

	Graphs	Classes	Average of nodes	Average of edges	Node features
Shapes	8000	5	27.230	144.927	1
Is_Acyclic	533	2	28.463	68.079	1
MUTAG	188	2	17.931	39.585	7
NCI1	3847	2	29.946	65.053	8
IMDB-BINARY	1000	2	19.773	193.062	1
REDDIT-BINARY	2000	2	429.627	995.508	1

32G of RAM, and an NVidia Tesla A100 (unnecessary for MIPExplainer, which runs only on the CPU). As no clear strategy exists to tune all of their hyperparameters, we use the default hyperparameter settings provided in their papers as much as possible. An exception was made for XGNN because the default regularization weights provided by the authors cause the graph generator to quickly learn a policy that stopped after the first node in several instances. To fix this, we have increased the reward for creating additional valid edges to the point that it is favorable for the explanation model to generate reasonably-sized explanations. In all experiments, the GNNs use GraphSAGE-style convolutions with summation 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, Shapes, and IMDB-BINARY 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 REDDIT-BINARY dataset, we trained a deeper, less-wide network with 4 convolutional layers computing 8 features per node and two FC layers computing 8 features per graph before the final FC layer computing the 2 class logits. For the MUTAG and NCI1 datasets, the GNN uses 2 convolutional layers computing 64 and 32 features per node, two FC layers computing 16 and 8 features per graph, and the final FC layer computing the logits. All GNNs are implemented using PyTorch-Geometric [53] and trained for 200 epochs, optimizing with Adam [54] with a learning rate of 10^{-3} and L2 regularization with weight 10^{-4}.

XGNN's graph generator policy network penalizes the violation of valence constraints while generating molecules on the MUTAG dataset. 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, but no chemistry-specific constraints were used. Based on the reported time from all methods using the same hardware, we observed that two hours was enough for most methods to report reasonable performance. If MIPExplainer did not prove optimality after two hours, we report the best solution found. To ensure that any resemblance to target classes would not come from an initial solution, every explanation optimized by MIPExplainer was initialized with a graph generated by adding every possible edge to a line graph with probability 0.5.

We use stability of generated explanations, i.e., average graph edit distance (GED including node features) across multiple runs, as a performance metric for each explanation method. Note that stability does not guarantee the quality of the explanation graphs themselves, and thus is a necessary but not sufficient measure of performance. First, we generate explanations with 5, 6, 7, and 8 nodes (the baseline methods sometimes produce explanations that do not have the maximum number of nodes) using each method on each dataset 5 times. Then, we compute the average GED among the 5 explanations. Table 3 shows these metrics averaged over the different numbers of nodes when generating class representatives, and is discussed in Section 4.2.5. A full table containing individual results for each number of nodes can be found in Appendix.

Table 2
Performance summary of our trained GNNs.

	Train accuracy	Test accuracy	Number of model parameters
Shapes	0.991	0.993	757
Is_Acyclic	0.998	1.000	730
MUTAG	0.893	0.895	5770
NCI1	0.820	0.810	5898
IMDB-BINARY	0.718	0.715	730
REDDIT-BINARY	0.856	0.858	594

4.2. *Results: Class representative explanations*

We start by analyzing the class representatives found by optimizing over Eq. (Class-Representative Explanation) and comparing them to the representatives found by XGNN, GNNInterpreter, and PAGE. The main results from our experiments are discussed in Sections 4.2.1–4.2.3 as shown in Tables 4–6. These tables show 3 explanations randomly chosen among the results of 5 runs for each method and each dataset with each explanation size.

4.2.1. *Shapes*

For the Shapes dataset, we can consistently recognize class-specific features in each of the explanations generated by MIPExplainer, all of which are proven to be optimal. Despite the fact that a significant amount of noise has been added to the training data, the explanations are relatively clean. This may explain why the explanations for Star graphs are not perfect stars, because in the dataset, Star graphs often had noisy edges added between the outer nodes. Similarly, this may be the reason that explanations for the Lollipop class sometimes have cycles instead of tails. In reality, graphs from this class usually just consisted of a clique amid a less-densely connected group of nodes after the noisy edges were added (since no edges can be added to the clique in the original graph, they are all added to the stem). For reference, examples from the four classes in the Shapes dataset are shown in Fig. 2.

As shown in Table 4, MIPExplainer generates reasonable class representatives for all of the different shapes and across the different graph sizes. GNNInterpreter and XGNN perform comparably to MIPExplainer, but we do see that GNNInterpreter will sometimes generate path graphs across all shapes, whereas XGNN seems to generate shapes resembling lollipops for the classes of Star and Wheel. Although GNNInterpreter and XGNN can find suitable explanations, the same experimental settings with different parameter initializations lead to widely varying results (low stability). This is also the case in the rest of our experiments with different models and datasets (also seen in Table 3). PAGE produces reasonable explanations for most of the classes except for the Star class, for which it generated a structure found in graphs from all of the classes (despite attempts at parameter tuning for this class, the resulting explanation remained unchanged). This is an instance where our stability metric is not sufficient for measuring the performance of explanation methods.

Table 3
Stability comparison: average edit distance between 5 generated example graphs, averaged for numbers of nodes between 5 and 8 inclusive. Time limit is 2 h.

Dataset	Method:	Average edit distance					
	Class	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Shapes	Grid	0.0 ± 0.00	3.6 ± 0.89	3.0 ± 2.42	2.0 ± 0.46	2.6 ± 0.47	1.4 ± 1.00
	Lollipop	0.0 ± 0.00	3.5 ± 0.77	3.4 ± 2.03	3.4 ± 1.81	4.0 ± 1.85	1.4 ± 0.0
	Star	0.0 ± 0.00	3.9 ± 1.21	3.0 ± 2.60	0.0 ± 0.00	3.2 ± 1.59	1.4 ± 0.28
	Wheel	0.75 ± 1.0	3.3 ± 1.04	4.0 ± 3.32	4.0 ± 2.26	2.9 ± 1.37	1.4 ± 0.63
Is_Acyclic	Acyclic	0.15 ± 0.30	3.5 ± 0.77	2.8 ± 1.62	0.0 ± 0.00	3.0 ± 1.38	1.6 ± 1.18
	Cyclic	0.0 ± 0.00	3.1 ± 1.05	3.2 ± 1.82	3.4 ± 1.14	3.2 ± 1.40	1.6 ± 0.57
MUTAG	Mutagen	0.0 ± 0.00	8.1 ± 1.19	7.8 ± 2.17	0.8 ± 0.57	6.0 ± 1.14	3.3 ± 0.44
	Nonmutagen	3.1 ± 4.53	7.8 ± 1.56	7.4 ± 2.39	0.0 ± 0.00	5.8 ± 1.53	3.1 ± 1.35
NCI1	Active	8.2 ± 4.12	7.6 ± 1.69	7.4 ± 2.14	3.7 ± 0.76	7.0 ± 1.40	3.6 ± 1.33
	Non-Active	0.5 ± 0.60	9.4 ± 3.46	7.0 ± 2.17	2.7 ± 0.60	6.8 ± 1.72	3.3 ± 1.27
IMDB-BINARY	Action	0.0 ± 0.00	3.3 ± 1.73	4.0 ± 3.32	3.8 ± 4.17	4.9 ± 5.61	1.6 ± 0.75
	Romance	0.2 ± 0.40	4.4 ± 1.21	3.4 ± 2.14	0.0 ± 0.00	2.0 ± 2.56	1.4 ± 0.44
REDDIT-BINARY	Discussion	0.0 ± 0.00	5.0 ± 4.07	2.8 ± 2.36	OOM	OOM	OOM
	QA	0.0 ± 0.00	3.6 ± 1.26	2.9 ± 1.01	OOM	OOM	OOM

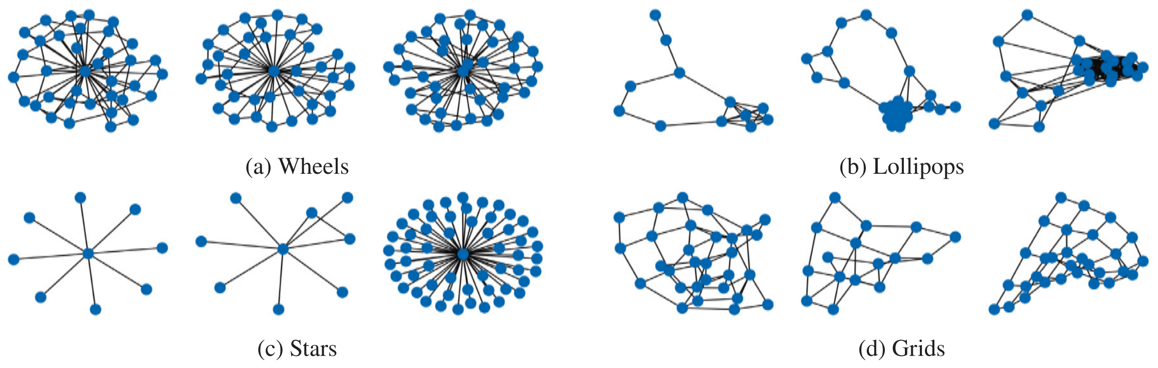


Fig. 2. Randomly selected graphs from the Shapes dataset.

4.2.2. *Is_Acyclic*

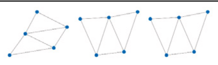
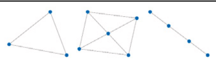
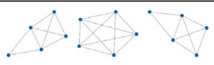
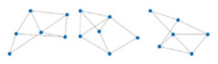
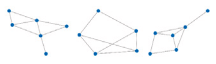
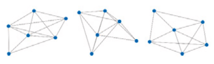
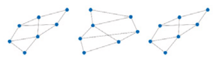
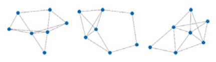
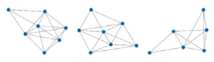
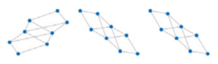
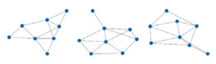
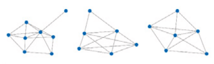
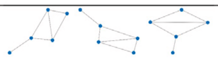
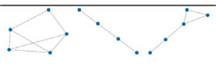
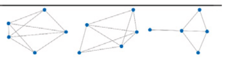
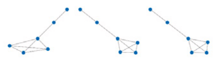
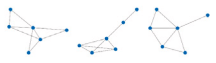
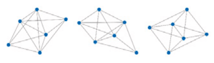
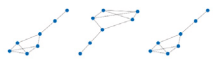
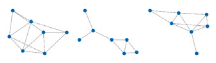
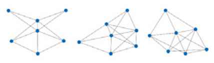
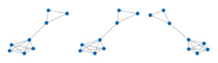
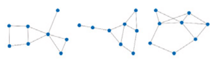
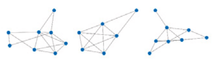
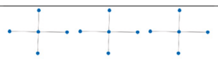
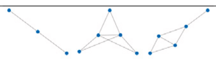
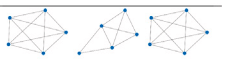
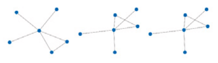
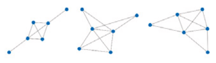
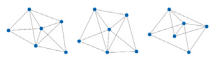
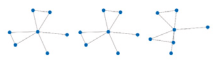
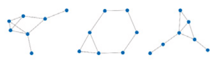
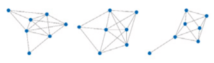
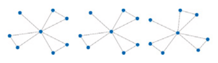
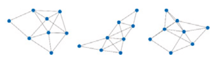
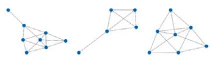
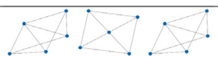
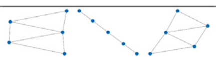
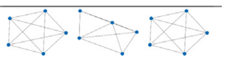
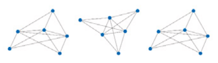
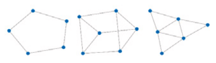
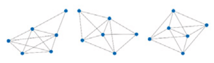
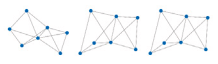
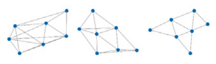
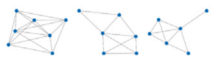
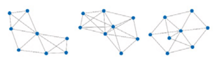
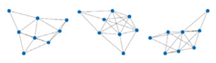
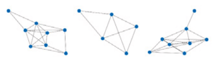
In the experiments with Is_Acyclic, MIPExplainer always explains the Cyclic class with a complete graph, which has the maximum possible number of cycles. It explains the Acyclic class with a Star graph, which is one of the most straightforward examples from the class. PAGE performs very well on the Acyclic class, but the results on the Cyclic class are less stable, as it generates path graphs in multiple trials. In contrast, the explanation graphs of XGNN and GNNInterpreter for the Cyclic class sometimes contain nodes with a single neighbor, and their explanations for the Acyclic class often include many cycles. D4Explainer always produced graphs with cycles, while KnowGNN always produced graphs without cycles. For MIPExplainer, while optimality was achieved for all results, its runtime was notably different between the Cyclic and Acyclic classes as shown in Table 5. Although the average time to generate the Cyclic explanation increased by less than a second between 5 nodes and 8 nodes, the average time to generate the Acyclic explanation increased from around 5 s to around 144. This is partially due to the number of graph representations in the equivalence class of each explanation. 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 before symmetry breaking constraints are added, 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, the solver may explore more of the search tree to prove the optimality of the Acyclic explanation. While the symmetry breaking techniques we employed do mitigate this problem, they do not alleviate it completely. Other causes could have to do with the structure of the GNN itself and numerical issues. Example plots of the objective bounds and the number of explored and unexplored search nodes during the search for an optimal solution can be found in Appendix.

To further visualize the behavior of MIPExplainer, in Fig. 3 we show all of the intermediate incumbent solutions discovered while searching for an explanation for our GNN's prediction of the Wheel dataset. These were obtained when new graphs were found during branch and bound that improved upon the previous incumbent solution's objective value.

4.2.3. *MUTAG*

For the mutagenic class of the MUTAG dataset, MIPExplainer produces a complete graph of carbon atoms. While the presence of carbon cycles is an important factor in the mutagenicity of organic molecules, they appear exclusively as rings of 5 or 6 carbon atoms. None of

Table 4
Explanations from each method for the Shapes GNN.

Class	Method: # Nodes	MIPExplainer	GNNInterpreter	XGNN
Grid	5			
	6			
	7			
	8			
Lollipop	5			
	6			
	7			
	8			
Star	5			
	6			
	7			
	8			
Wheel	5			
	6			
	7			
	8			

(*continued on next page*)

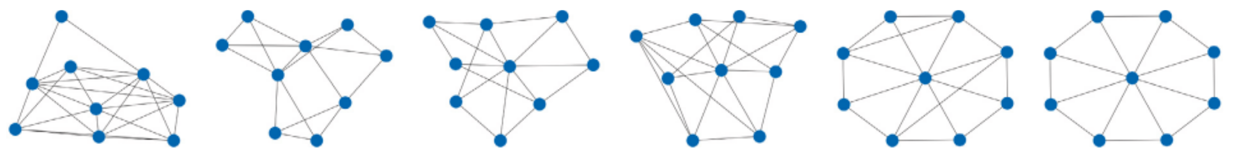
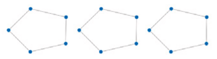
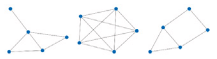
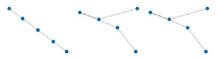
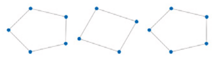
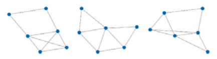
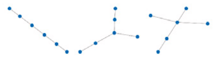
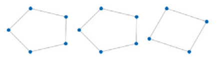
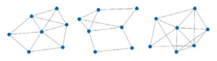
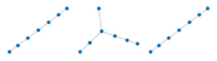
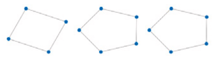
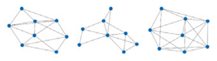
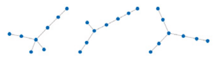
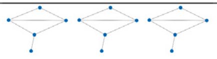
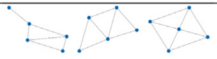
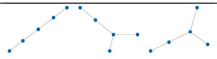
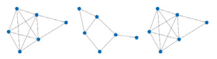
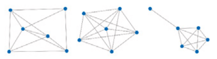
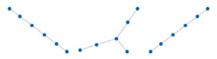
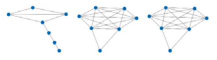
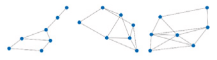
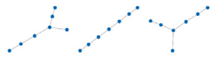
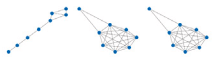
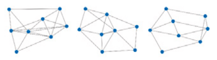
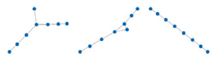
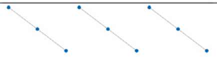
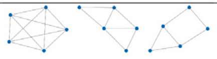
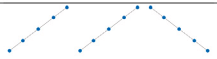
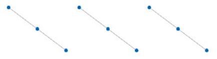
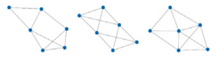
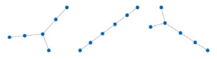
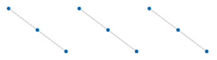
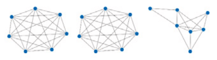
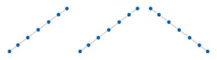
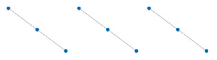
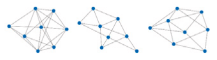
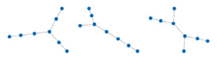
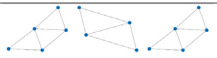
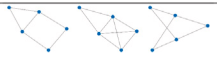
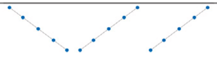
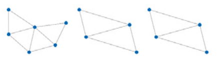
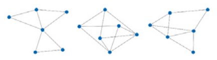
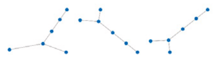
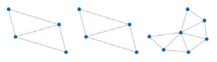
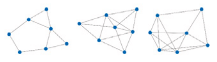
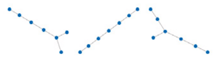
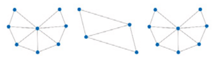
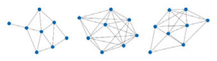
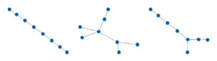


Fig. 3. Algorithmic behavior: improved solutions over the branch and bound tree search starting from a randomly initialized graph from the left all the way to the final optimal solution on the right.

the explanations generated by the baseline methods contained a cycle of carbon atoms. For nonmutagens, MIPExplainer produces molecules with carbon atoms and bromine atoms, the latter of which appear in 2 of them. Notably, any two carbon atoms in these explanations have a bromine atom in between that ensures there are no carbon cycles, but we cannot clearly state that this is the pattern that the GNN is using to differentiate the classes. The explanations of the non-mutagenic class are actually less 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. PAGE struggled the most on this dataset, it only produced path graphs of carbon atoms for both classes. On the other hand, all baselines produced completely inconsistent results with seemingly random structure.

Due to the larger size of the GNN model and the introduction of node features in MUTAG, almost none of MIPExplainer's explanation graphs were able to be verified as optimal. However, even starting from random initialization graphs every time, the outputs were very stable

Table 4 (*continued*).

	Method:	PAGE	D4Explainer	KnowGNN
Class	# Nodes			
Grid	5			
	6			
	7			
	8			
Lollipop	5			
	6			
	7			
	8			
Star	5			
	6			
	7			
	8			
Wheel	5			
	6			
	7			
	8			

and often extremely close to the upper bound established by the solver. While MIPExplainer's full runtime may be longer than that of other methods in several settings, our experimental results demonstrate that the method is practically useful even with runtime limits that stop the optimization procedure early.

4.2.4. *REDDIT-BINARY*

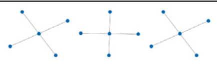
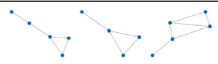
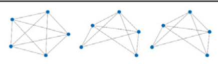
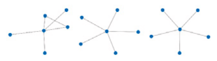
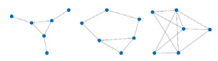
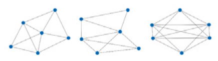
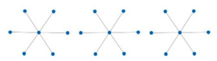
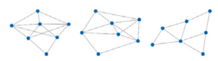
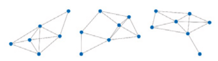
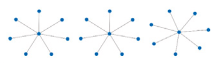
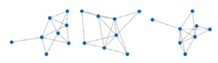
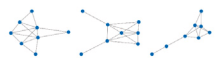
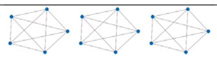
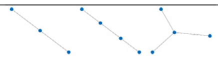
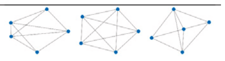
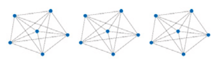
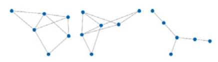
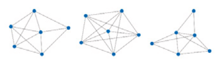
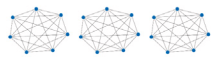
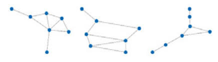
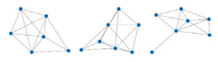
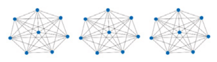
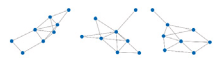
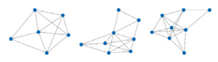
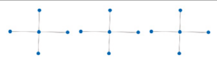
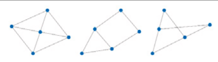
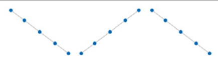
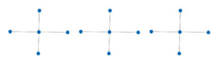
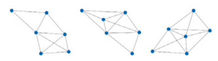
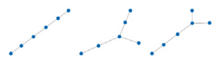
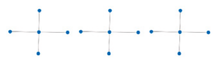
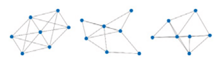
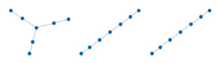
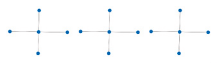
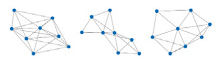
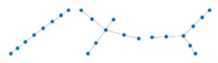
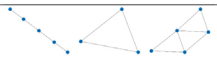
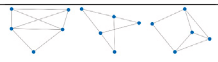
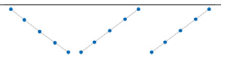
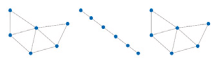
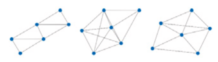
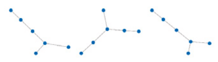
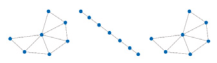
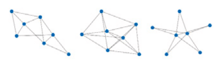
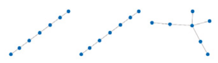
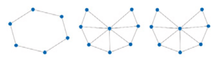
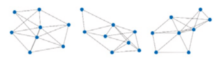
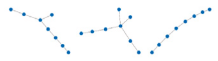
The REDDIT-BINARY dataset contains graphs representing the structure of comments sections on Reddit, with nodes representing users and edges between pairs of users if one replied to the other. The two classes contain posts from discussion-based forums and Question/Answer (QA)-based forums. MIPExplainer produces explanations that are aligned with the fact that discussion-based forums would have a denser reply structure, while QA-based forums would tend to be more tree-like with the original user as the root. This is reflected in some of GNNInterpreter's results, but not most, especially with higher numbers of nodes in the explanation graph. XGNN also tended to produce more densely connected graphs for the discussion class, but this would be hard to see without prior knowledge. Because the graphs from this dataset are significantly larger in size, the remaining three baselines ran out of memory and were not able to produce any results.

4.2.5. *Stability and runtime*

Table 3 clearly shows the high stability of MIPExplainer with substantially lower graph edit distances on average over the generated explanations than GNNInterpreter and XGNN which both rely on stochastic optimization. In our MIPExplainer, small variations in explanations are due to the existence of multiple explanation graphs with the exact same objective value, which tend to be extremely similar (as observed in our experiments on Shapes and Is_Acyclic data). On MUTAG, the algorithm could run out of time before finding an optimal solution, which, however, rarely caused deviations, as the best solution was generally found much earlier than it was proven to be optimal. The only case in which MIPExplainer does not achieve the highest stability was with the NCI1 dataset, which could have a number of causes. Not only was this GNN the largest, not allowing us to prove optimality within the time limit, but this task was also the most complex and without a clear ground-truth. KnowGNN and PAGE are also relatively stable in certain experimental settings, but by cross-referencing Tables 4–7 we observe that in these cases the explanatory graphs are not related to the ground truth of the dataset.

The runtimes for all of our experiments are shown in Table 8. Note that only the time recorded for our method is the convergence time whereas for all other methods, the runtime depends on the value of a hyperparameter which is the number of maximum iterations. The runtime in Table 8 for these methods was obtained by using their default

Table 5
Explanations from each method for the Is_Acyclic GNN.

Class	Method: # Nodes	MIPExplainer	GNNInterpreter	XGNN
Acyclic	5			
	6			
	7			
	8			
Cyclic	5			
	6			
	7			
	8			
Class	**Method: # Nodes**	**PAGE**	**D4Explainer**	**KnowGNN**
Acyclic	5			
	6			
	7			
	8			
Cyclic	5			
	6			
	7			
	8			

hyperparameter values in their original implementations. These methods can run arbitrarily long with different hyperparameter choices, but without any convergence guarantee. Most importantly, there is no guidance on how to choose proper values for this hyperparameter, so we report their default runtimes. While the computational cost of convergence is significant, it is also necessary when generating explanations without any ground truth. Furthermore, the computational complexity of MIPExplainer does not depend on the size of the dataset or the graphs it contains, allowing it to be applied to datasets like REDDIT-BINARY where baseline methods such as PAGE, D4Explainer, and KnowGNN, run out of memory with the same hardware configuration. Various methods for improving MIP encodings of ReLU neural networks [55–57] can also be used to reduce MIPExplainer's runtime.

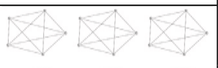
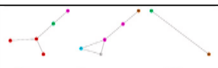
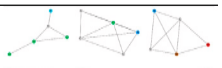
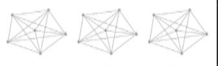
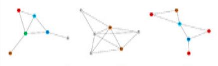
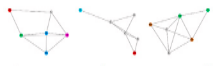
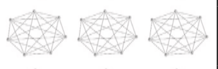
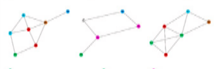
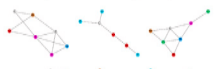
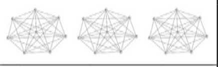
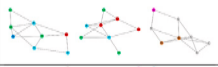
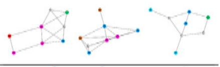
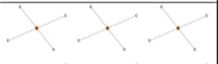
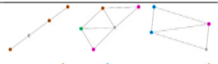
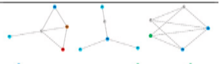
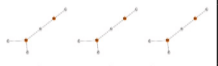
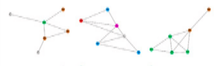
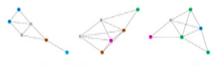
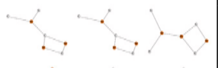
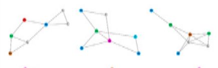
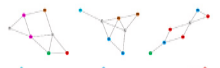
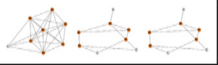
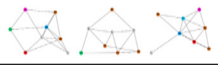
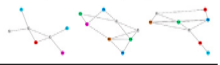
4.3. Results: Alternative objectives

We now discuss the results we obtained when optimizing over the alternate objective functions that we proposed in Sections 2.1.2–2.1.3.

4.3.1. Boundary explanations

Fig. 4 shows the result of encoding GNNs with MIPs and optimizing the objective defined in Eq. (Boundary Explanation) with various pairs of classes. Between Cyclic and Acyclic classes, the identified graph is half completely acyclic and half densely connected, so it is understandable why this might be a hard-to-classify case. Applying this same explanation objective to a multi-class problem on the Shapes dataset provided more diverse results. Part of the reason for this is that not every pair of classes must share a decision boundary. Nevertheless, each explanation does exhibit features of the prototypes used to construct the corresponding classes in the datasets. For instance, the Wheel/Grid explanation contains both lattice-like structures and an outer cycle. The Star/Lollipop graph does have a sparsely connected subgraph and a more densely connected subgraph, but the densely-connected subgraph seems to be composed of two Star subgraphs. Fig. 4(d) shows a pattern that GNN model exhibits as difficult to distinguish between Lollipop and Wheel, but is not easily connected to ground-truth knowledge about the distribution of the data in the two classes.

Table 6
Explanations from each method for the MUTAG GNN. Atom types are assigned to node colors as follows: gray=Carbon, blue=Nitrogen, red=Oxygen, cyan=Fluorine, purple=Iodine, green=Chlorine, and brown=Bromine.

Class	Method: # Nodes	MIPExplainer	GNNInterpreter	XGNN
Mutagen	5			
	6			
	7			
	8			
Nonmutagen	5			
	6			
	7			
	8			

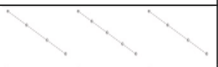
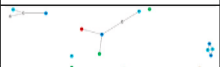
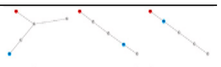
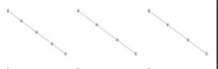
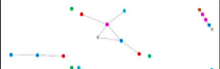
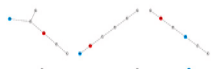
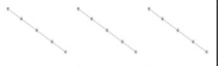
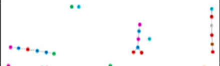
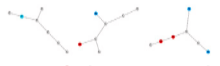
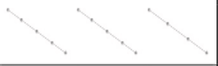
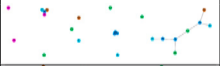
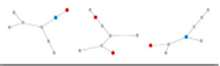
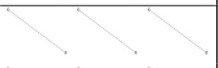
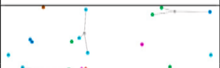
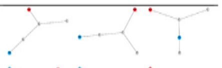
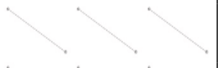
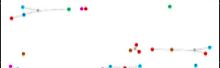
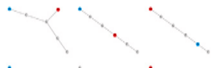
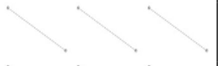
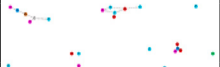
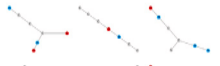
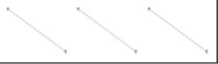
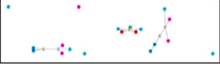
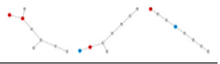
Class	Method: # Nodes	PAGE	D4Explainer	KnowGNN
Mutagen	5			
	6			
	7			
	8			
Nonmutagen	5			
	6			
	7			
	8			

Table 7
Explanations for a GNN trained on REDDIT-BINARY. PAGE, D4Explainer, and KnowGNN all ran out of memory before generating any explanations.

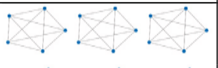
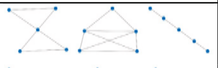
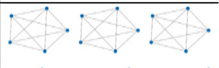
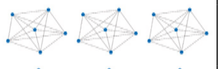
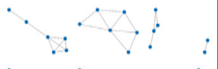
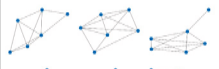
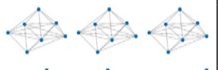
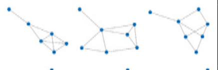
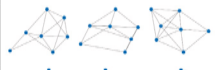
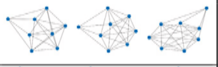
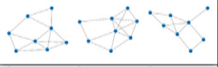
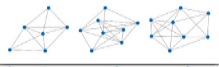
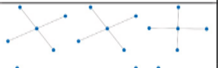
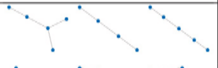
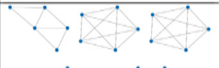
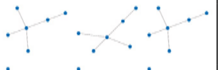
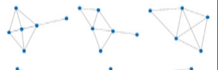
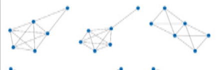
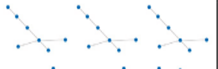
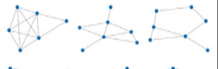
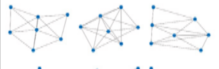
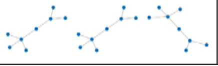
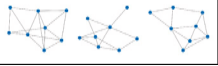
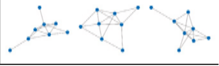
Class	Method: # Nodes	MIPExplainer	GNNInterpreter	XGNN
Discussion	5			
	6			
	7			
	8			
QA	5			
	6			
	7			
	8			

Table 8
Runtime of explanation methods in seconds given a time limit of 2 h, averaged over 5 runs. Note that baseline methods all require a hyperparameter to specify the maximum number of iterations, so the runtime with the default values for this hyperparameter are shown here for these methods, which does not correspond to convergence time.

Dataset	Class	Method:	Runtime (s)					
		Nodes	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Shapes	Grid	5	2.833 ± 0.138	7.563 ± 0.018	11.644 ± 0.196	169.229 ± 35.599	6636.139 ± 13.134	34.769 ± 11.610
		6	5.787 ± 1.258	7.614 ± 0.040	14.964 ± 0.151	168.219 ± 27.860	6624.593 ± 5.153	41.541 ± 9.323
		7	31.601 ± 9.317	7.648 ± 0.022	17.817 ± 0.149	196.568 ± 56.987	6632.882 ± 13.970	81.888 ± 62.356
		8	129.581 ± 44.449	7.667 ± 0.007	20.497 ± 0.735	186.375 ± 39.010	6647.856 ± 36.116	79.794 ± 32.195
	Lollipop	5	3.347 ± 0.310	7.631 ± 0.016	11.543 ± 0.102	1722.388 ± 1645.125	6638.538 ± 6.854	44.444 ± 6.163
		6	9.284 ± 2.155	7.634 ± 0.013	15.048 ± 0.410	2446.997 ± 1744.256	6655.414 ± 22.061	92.424 ± 83.872
		7	84.935 ± 57.100	7.661 ± 0.019	17.934 ± 0.370	4519.390 ± 2483.927	6642.972 ± 15.999	88.168 ± 84.727
		8	395.661 ± 35.234	7.705 ± 0.027	19.852 ± 0.355	4544.374 ± 2391.289	6637.966 ± 13.236	96.729 ± 69.865
	Star	5	2.679 ± 0.211	7.560 ± 0.014	11.512 ± 0.173	1092.232 ± 24.380	6640.302 ± 11.576	44.053 ± 10.285
		6	8.565 ± 0.593	7.592 ± 0.007	14.955 ± 0.279	981.970 ± 201.647	6641.986 ± 15.288	1555.602 ± 2019.522
		7	18.440 ± 3.583	7.624 ± 0.014	17.942 ± 0.440	960.851 ± 177.717	6633.952 ± 11.672	330.358 ± 575.275
		8	201.184 ± 10.097	7.659 ± 0.016	19.931 ± 0.673	1052.084 ± 156.791	6641.880 ± 6.828	67.176 ± 13.427
	Wheel	5	2.574 ± 0.295	7.716 ± 0.255	11.550 ± 0.043	339.397 ± 68.815	6638.569 ± 9.102	57.327 ± 24.986
		6	4.852 ± 0.811	7.641 ± 0.010	15.507 ± 0.763	287.603 ± 89.035	6642.933 ± 8.283	42.357 ± 3.097
		7	30.555 ± 27.573	7.652 ± 0.010	18.047 ± 0.395	267.701 ± 62.479	6640.200 ± 9.726	79.994 ± 49.452
		8	103.380 ± 6.843	7.706 ± 0.051	20.095 ± 0.387	344.541 ± 81.041	6639.850 ± 10.452	101.561 ± 82.853
Is_Acyclic	Acyclic	5	5.031 ± 1.237	7.610 ± 0.015	10.040 ± 0.135	70.247 ± 8.764	2182.708 ± 238.532	80.434 ± 27.150
		6	8.660 ± 1.731	7.626 ± 0.022	13.482 ± 0.528	68.175 ± 9.639	2191.858 ± 276.752	71.319 ± 48.391
		7	20.655 ± 3.781	7.679 ± 0.054	15.419 ± 0.561	72.303 ± 7.872	2064.532 ± 221.328	218.001 ± 264.619
		8	144.003 ± 18.049	7.684 ± 0.013	17.265 ± 0.479	74.056 ± 10.106	2110.262 ± 218.104	67.598 ± 12.993
	Cyclic	5	2.721 ± 0.356	0.027 ± 0.008	9.842 ± 0.057	45.882 ± 53.643	2126.876 ± 89.113	61.424 ± 29.783
		6	2.389 ± 0.310	0.024 ± 0.000	12.877 ± 0.482	81.296 ± 60.228	2204.120 ± 324.208	110.652 ± 122.933
		7	2.497 ± 0.637	0.030 ± 0.003	15.257 ± 0.377	94.117 ± 63.071	2033.200 ± 354.808	112.535 ± 58.912
		8	3.160 ± 0.359	0.115 ± 0.051	18.216 ± 1.273	82.400 ± 46.520	1929.667 ± 152.450	107.532 ± 36.047
MUTAG	Mutagen	5	643.690 ± 40.088	0.024 ± 0.001	8.701 ± 0.455	77.669 ± 8.570	660.824 ± 61.413	41.135 ± 7.637
		6	2344.545 ± 305.734	0.025 ± 0.004	10.305 ± 0.297	77.002 ± 13.707	690.062 ± 76.165	58.004 ± 15.924
		7	2838.178 ± 677.224	0.060 ± 0.060	12.069 ± 0.257	82.721 ± 12.034	678.170 ± 48.154	71.139 ± 21.163
		8	6685.271 ± 1154.943	0.114 ± 0.007	14.727 ± 0.619	85.198 ± 11.644	670.695 ± 45.891	393.042 ± 395.584
	Nonmutagen	5	4092.657 ± 513.121	5.633 ± 5.109	8.747 ± 0.530	78.059 ± 9.494	646.606 ± 55.436	31.863 ± 7.178
		6	7202.002 ± 0.264	7.599 ± 4.213	10.403 ± 0.342	84.976 ± 11.062	707.473 ± 46.319	51.356 ± 4.744
		7	7201.925 ± 0.312	9.659 ± 0.049	12.595 ± 0.731	79.095 ± 14.449	689.110 ± 65.032	88.163 ± 48.180
		8	7201.931 ± 0.170	7.857 ± 4.304	13.946 ± 0.429	71.706 ± 13.730	652.691 ± 52.439	98.607 ± 25.698
NCI1	Active	5	7200.976 ± 0.335	18.695 ± 3.624	36.145 ± 27.658	28.105 ± 6.455	6628.768 ± 7.199	293.771 ± 139.735
		6	7201.089 ± 0.158	42.992 ± 40.033	73.601 ± 43.805	38.930 ± 3.414	6635.254 ± 16.367	437.854 ± 340.840
		7	7201.499 ± 0.837	12.780 ± 28.418	133.788 ± 124.983	25.549 ± 5.204	6622.113 ± 4.287	182.442 ± 110.223
		8	7201.659 ± 0.637	17.832 ± 5.407	371.441 ± 209.454	37.124 ± 12.530	6628.570 ± 5.847	98.045 ± 66.289
	Non-Active	5	7200.941 ± 0.261	0.037 ± 0.020	13.650 ± 2.605	19.088 ± 7.303	6627.870 ± 7.406	381.068 ± 149.799
		6	7200.610 ± 0.015	0.028 ± 0.001	18.425 ± 6.625	14.147 ± 2.998	6642.870 ± 27.445	360.059 ± 346.804
		7	7201.064 ± 0.241	0.040 ± 0.007	64.451 ± 31.440	17.477 ± 7.124	6633.142 ± 3.875	318.452 ± 220.787
		8	7200.968 ± 0.093	0.153 ± 0.061	160.083 ± 90.902	15.855 ± 3.352	6624.526 ± 4.914	294.216 ± 220.045
IMDB-BINARY	Action	5	2.499 ± 0.304	10.702 ± 0.038	15.002 ± 1.436	45.802 ± 33.538	4039.482 ± 460.117	91.798 ± 22.387
		6	2.913 ± 0.185	10.912 ± 0.107	82.564 ± 50.800	70.076 ± 29.706	4002.983 ± 324.241	498.072 ± 293.310
		7	6.771 ± 0.914	10.974 ± 0.108	144.109 ± 29.075	97.650 ± 35.220	3774.532 ± 452.741	479.132 ± 358.399
		8	132.072 ± 51.895	19.732 ± 13.523	151.082 ± 86.939	81.356 ± 40.307	3599.583 ± 263.459	527.060 ± 645.516
	Romance	5	2.579 ± 0.627	0.691 ± 0.193	15.156 ± 3.558	1171.962 ± 1050.303	3966.199 ± 258.170	181.780 ± 213.022
		6	2.493 ± 0.337	129.888 ± 81.306	29.177 ± 15.858	824.337 ± 1097.327	4062.581 ± 484.982	117.554 ± 48.547
		7	6.489 ± 1.069	146.370 ± 83.668	65.645 ± 66.173	775.247 ± 1035.251	3956.617 ± 461.542	490.110 ± 495.073
		8	7.682 ± 1.540	23.087 ± 17.984	25.137 ± 4.034	1233.731 ± 1104.846	3905.860 ± 669.663	175.041 ± 32.839
REDDIT-BINARY	Discussion	5	5.810 ± 0.812	0.028 ± 0.008	14.428 ± 1.579	OOM	OOM	OOM
		6	26.655 ± 9.815	0.105 ± 0.072	18.422 ± 3.013	OOM	OOM	OOM
		7	140.175 ± 96.354	0.658 ± 0.165	20.056 ± 1.840	OOM	OOM	OOM
		8	1021.709 ± 122.451	0.964 ± 1.695	32.622 ± 13.243	OOM	OOM	OOM
	QA	5	17.026 ± 2.842	15.222 ± 6.558	12.253 ± 0.280	OOM	OOM	OOM
		6	130.413 ± 32.799	16.263 ± 9.934	15.880 ± 0.901	OOM	OOM	OOM
		7	568.535 ± 99.300	40.655 ± 17.155	19.099 ± 0.664	OOM	OOM	OOM
		8	4867.402 ± 885.642	48.459 ± 55.990	21.044 ± 0.709	OOM	OOM	OOM

Runtime for MIPExplainer reported to convergence unless the time limit was reached.

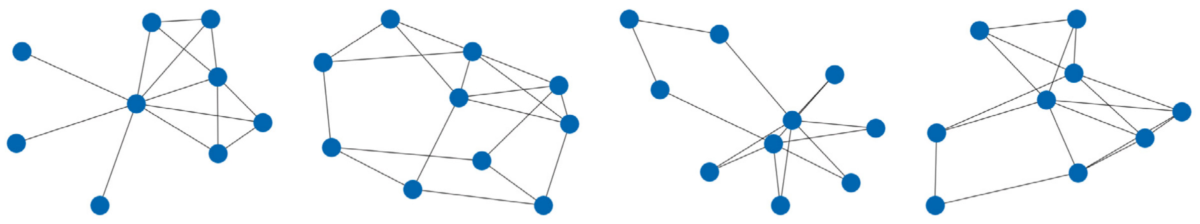

(a) Cyclic/Acyclic (b) Wheel/Grid (c) Star/Lollipop (d) Lollipop/Wheel

Fig. 4. Finding separation boundary exemplar graphs by solving Eq. (Boundary Explanation) via MIPExplainer on Is_Acyclic (a) and Shapes (b, c, and d) data.

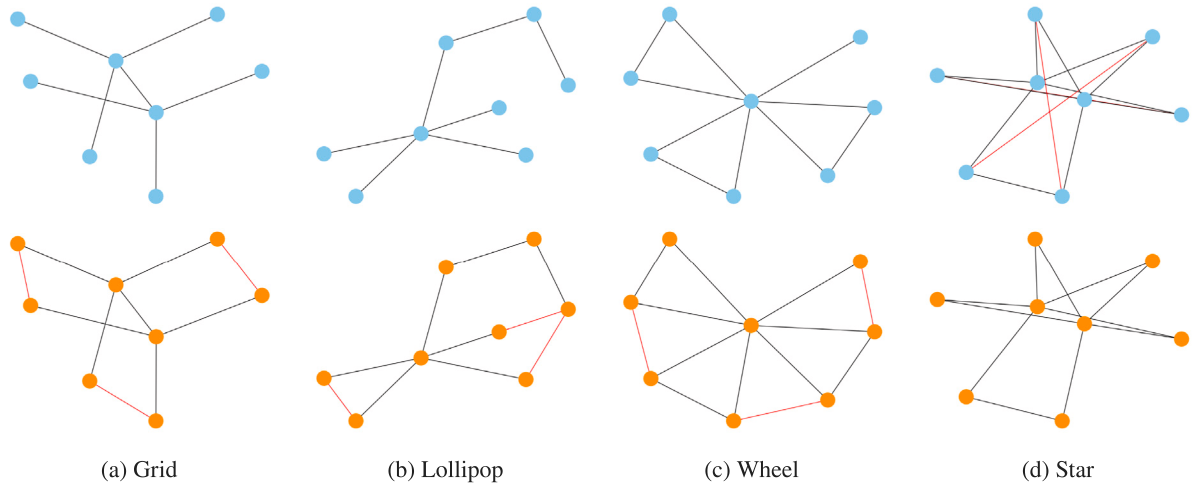


Fig. 5. Paired explanations for several classes from the Shapes Dataset, given a budget of 3 changes to the adjacency matrix. We aim to maximize the difference in predicted probability for each class between the blue graphs on top (lower probability) and the orange graphs on bottom (higher probability). Differing edges within each pair are highlighted in red.

4.3.2. *Paired explanations*

Fig. 5 shows explanations generated by optimizing Eq. (Paired Explanation) for each class in the Shapes dataset with a budget of $\delta = 3$ edge insertions/deletions. The orange graphs on bottom correspond to G_1 in Eq. (Paired Explanation), and are predicted with higher probability than the blue graphs on top corresponding to G_2. The explanation for the Grid class shows edges being added to ensure that the graph is composed exclusively of 4-cycles, the defining characteristic of this class's prototype. The Wheel explanation starts as a graph that looks more like the Star graphs from the dataset after noise is added, but then additional edges are added between the outer nodes to create a rim. In the Star class explanation, we see that edges are actually taken away from the higher graph to get to the lower graph. The edges are only removed from the nodes with the lowest degrees, creating Star class's defining characteristic. For the Lollipop class, the explanation does not seem to create any distinctive features. However, looking at the sets of logits for both graphs, we see that the terms involving the lower-probability (blue) graph dominated the objective, meaning that the relevant change for this explanation is actually the removal of features indicative of another class, in this case Stars. Fig. 6 shows the result of adding the constraint that G_1 actually be predicted to the correct class by the GNN (i.e. that its logit is at least as large as the maximum of the other logits). Interestingly, although the changes between the graphs can mostly be interpreted in the same way as in Fig. 5, the bottom graphs are less immediately recognizable as instances of the corresponding class, which is opposite to the result we expected from adding the new constraint.

5. Conclusions and discussion

Despite the ability of GNNs to model complex patterns in graph-structured data, their lack of transparency remains one of the major 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 shortcomings that limit the use of existing methods in most real-world situations, this work proposes MIPExplainer for generating post-hoc model-level 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 has limitations which we hope to address in future work. While it is more general than previous methods without specific data assumptions, 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 challenging in general, and more work is required to understand which symmetries are the most costly when optimizing over sets of graphs. Despite these limitations, we observe that the proposed method is able to find reasonable explanations.

CRediT authorship contribution statement

Blake B. Gaines: Conceptualization, Methodology, Software. **Chun jiang Zhu:** Writing – original draft, Visualization. **Jinbo Bi:** Conceptualization, Methodology, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Dr. Laurent Michel for lending his expertise in the field of discrete optimization to make this approach possible. This work was partially supported by NIH, United States grants R01-DA051922 and U19-AI171421 to J. Bi. We thank the funding agencies for their sponsorship, although this work does not necessarily represent their opinions.

Appendix. Extended experimental results

See Fig. 7, Tables 9–12.

Data availability

Our data and code is publicly available on Github at https://github.com/blake-gaines/MIPExplainer.

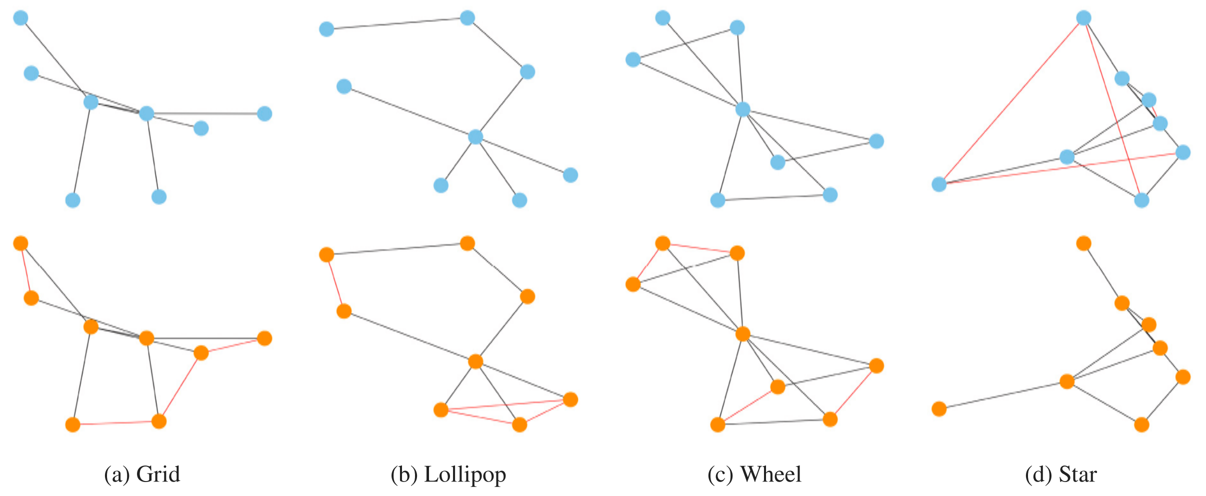


Fig. 6. Figures generated in the same way as in Fig. 5, except with a budget of 4 and the added constraint that the GNN must classify the orange graphs to the corresponding classes.

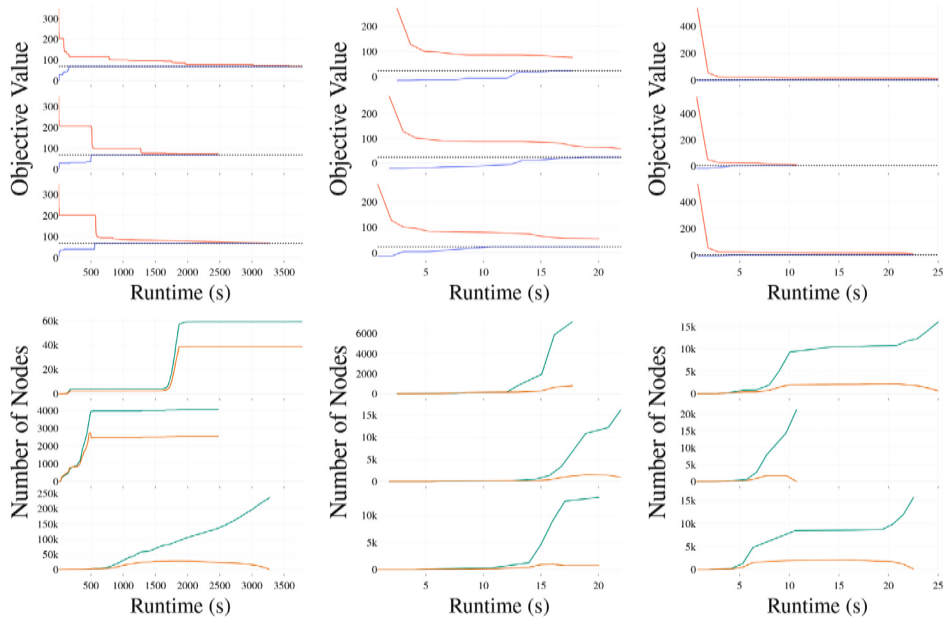


Fig. 7. Solver metrics for several runs explaining the mutagen class of MUTAG (left), the acyclic class of Is_Acyclic (middle), and the wheel class of Shapes (right) with 7 nodes: On the top, the current best solution's objective (blue) and upper bound (red) converging to the same global optimum (the dotted black line). On the bottom, the number of explored (green)/unexplored (orange) nodes during the search.

Table 9
Average edit distance between 5 generated example graphs. The time limit was two hours for all experiments. MIPExplainer had the lowest average edit distance in almost all experiments.

Dataset	Class	Method:	Average edit distance					
		Nodes	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Shapes	Grid	5	0.000 ± 0.000	4.400 ± 2.107	1.200 ± 0.748	1.600 ± 1.960	2.200 ± 1.661	0.000 ± 0.000
		6	0.000 ± 0.000	2.600 ± 0.917	1.000 ± 0.632	2.400 ± 1.960	2.200 ± 0.600	2.333 ± 1.374
		7	0.000 ± 0.000	3.000 ± 0.894	3.400 ± 1.855	1.600 ± 1.960	2.600 ± 0.800	1.400 ± 0.917
		8	0.000 ± 0.000	4.200 ± 0.872	6.200 ± 2.441	2.400 ± 1.960	3.200 ± 0.600	1.800 ± 0.600
	Lollipop	5	0.000 ± 0.000	2.800 ± 1.249	1.600 ± 1.428	1.200 ± 0.980	1.600 ± 0.663	1.200 ± 0.980
		6	0.000 ± 0.000	3.000 ± 0.894	2.000 ± 0.894	3.000 ± 2.449	3.600 ± 1.428	1.200 ± 0.980
		7	0.000 ± 0.000	4.000 ± 1.000	4.000 ± 1.000	3.600 ± 4.409	6.000 ± 3.098	1.200 ± 0.980
		8	0.000 ± 0.000	4.400 ± 1.356	6.000 ± 2.145	5.600 ± 6.859	4.600 ± 0.490	2.200 ± 1.077
	Star	5	0.000 ± 0.000	5.200 ± 2.272	1.200 ± 0.980	0.000 ± 0.000	1.600 ± 1.428	1.200 ± 0.980
		6	0.000 ± 0.000	2.600 ± 0.663	1.000 ± 0.632	0.000 ± 0.000	2.200 ± 1.077	1.400 ± 0.917
		7	0.000 ± 0.000	3.200 ± 0.748	3.400 ± 1.200	0.000 ± 0.000	4.800 ± 2.088	1.200 ± 0.980
		8	0.000 ± 0.000	4.600 ± 1.356	6.600 ± 3.611	0.000 ± 0.000	4.400 ± 1.685	1.800 ± 0.600
	Wheel	5	0.000 ± 0.000	1.800 ± 0.980	0.600 ± 0.490	1.800 ± 1.470	1.600 ± 0.800	1.000 ± 1.000
		6	0.000 ± 0.000	4.000 ± 1.183	2.000 ± 0.775	3.600 ± 2.939	2.000 ± 1.342	0.800 ± 0.980
		7	0.000 ± 0.000	3.400 ± 1.428	5.200 ± 2.750	3.600 ± 4.409	3.400 ± 1.356	1.600 ± 0.800
		8	3.000 ± 2.049	4.000 ± 0.894	8.000 ± 3.924	7.200 ± 5.879	4.600 ± 2.059	2.200 ± 1.077
Is_Acyclic	Acyclic	5	0.000 ± 0.000	3.000 ± 1.549	0.600 ± 0.490	0.000 ± 0.000	1.600 ± 0.663	0.000 ± 0.000
		6	0.600 ± 0.490	3.200 ± 1.077	2.400 ± 1.200	0.000 ± 0.000	2.400 ± 1.114	1.600 ± 0.800
		7	0.000 ± 0.000	3.000 ± 1.000	4.000 ± 1.414	0.000 ± 0.000	3.400 ± 0.917	2.400 ± 1.200
		8	0.000 ± 0.000	4.600 ± 1.020	4.000 ± 0.775	0.000 ± 0.000	4.800 ± 1.536	2.600 ± 0.917
	Cyclic	5	0.000 ± 0.000	2.600 ± 1.428	1.200 ± 0.600	3.000 ± 2.191	1.600 ± 0.490	0.800 ± 0.980
		6	0.000 ± 0.000	2.200 ± 1.249	2.800 ± 1.077	2.400 ± 1.960	2.600 ± 1.356	1.400 ± 0.917
		7	0.000 ± 0.000	3.000 ± 1.000	3.400 ± 1.744	3.000 ± 2.449	4.200 ± 1.778	2.000 ± 1.549
		8	0.000 ± 0.000	4.600 ± 1.020	5.600 ± 1.497	5.000 ± 3.550	4.600 ± 2.154	2.000 ± 1.265
MUTAG	Mutagen	5	0.000 ± 0.000	7.000 ± 1.483	5.200 ± 1.327	0.800 ± 0.980	4.800 ± 1.661	2.800 ± 0.872
		6	0.000 ± 0.000	7.300 ± 1.616	7.000 ± 1.414	1.200 ± 0.980	5.400 ± 1.497	3.000 ± 1.000
		7	0.000 ± 0.000	8.500 ± 1.285	8.500 ± 1.500	0.000 ± 0.000	6.400 ± 1.356	3.700 ± 1.269
		8	0.000 ± 0.000	9.600 ± 2.458	10.300 ± 1.345	1.200 ± 0.980	7.400 ± 0.800	3.600 ± 0.800
	Nonmutagen	5	0.000 ± 0.000	5.800 ± 1.327	4.600 ± 1.428	0.000 ± 0.000	4.400 ± 1.497	1.600 ± 0.800
		6	0.000 ± 0.000	7.500 ± 1.204	7.000 ± 1.549	0.000 ± 0.000	4.600 ± 0.917	2.500 ± 0.806
		7	2.800 ± 1.327	8.400 ± 1.428	7.700 ± 0.781	0.000 ± 0.000	7.100 ± 1.446	3.900 ± 0.700
		8	9.600 ± 7.838	9.500 ± 1.025	10.400 ± 2.107	0.000 ± 0.000	7.200 ± 2.272	4.600 ± 1.200
NCI1	Active	5	2.000 ± 2.449	5.900 ± 1.375	4.800 ± 1.536	3.600 ± 2.939	5.500 ± 1.025	1.833 ± 0.373
		6	10.100 ± 6.188	6.800 ± 0.872	6.900 ± 1.136	3.200 ± 3.919	6.400 ± 0.800	3.400 ± 1.020
		7	10.400 ± 4.079	8.000 ± 1.000	8.100 ± 0.539	3.200 ± 3.919	7.200 ± 1.600	4.000 ± 1.612
		8	10.200 ± 2.600	9.800 ± 0.748	9.900 ± 1.921	4.800 ± 3.919	8.800 ± 1.720	5.000 ± 1.183
	Non-Active	5	0.000 ± 0.000	6.200 ± 0.980	4.600 ± 1.020	2.400 ± 2.939	4.600 ± 1.114	3.167 ± 0.687
		6	0.800 ± 0.980	8.400 ± 1.281	5.800 ± 1.077	2.400 ± 2.939	6.200 ± 1.249	1.500 ± 0.806
		7	0.000 ± 0.000	8.600 ± 1.356	8.100 ± 0.539	2.400 ± 2.939	7.700 ± 0.900	4.200 ± 0.872
		8	1.200 ± 1.470	14.300 ± 4.406	9.400 ± 0.917	3.600 ± 2.939	8.500 ± 1.118	4.200 ± 1.470
IMDB-BINARY	Action	5	0.000 ± 0.000	3.000 ± 1.342	0.800 ± 0.600	1.800 ± 1.470	0.000 ± 0.000	1.000 ± 1.000
		6	0.000 ± 0.000	1.200 ± 0.600	1.400 ± 1.020	0.000 ± 0.000	1.600 ± 1.960	0.800 ± 0.980
		7	0.000 ± 0.000	3.600 ± 1.281	7.200 ± 3.789	3.600 ± 4.409	12.600 ± 11.629	2.200 ± 1.077
		8	0.000 ± 0.000	5.400 ± 1.020	6.400 ± 2.107	9.600 ± 5.463	5.400 ± 3.382	2.200 ± 0.872
	Romance	5	0.000 ± 0.000	3.600 ± 1.685	0.600 ± 0.490	0.000 ± 0.000	0.000 ± 0.000	1.000 ± 1.000
		6	0.000 ± 0.000	3.800 ± 1.327	3.200 ± 1.400	0.000 ± 0.000	2.400 ± 1.960	1.600 ± 1.200
		7	0.800 ± 0.980	6.200 ± 2.993	3.800 ± 1.778	0.000 ± 0.000	0.000 ± 0.000	2.000 ± 0.894
		8	0.000 ± 0.000	4.000 ± 0.894	5.800 ± 2.182	0.000 ± 0.000	5.400 ± 4.409	1.200 ± 0.980
REDDIT-BINARY	Discussion	5	0.000 ± 0.000	2.000 ± 0.894	0.400 ± 0.490	OOM	OOM	OOM
		6	0.000 ± 0.000	3.200 ± 0.872	2.000 ± 1.000	OOM	OOM	OOM
		7	0.000 ± 0.000	3.800 ± 1.249	2.800 ± 1.327	OOM	OOM	OOM
		8	0.000 ± 0.000	11.000 ± 8.649	6.000 ± 2.408	OOM	OOM	OOM
	QA	5	0.000 ± 0.000	2.000 ± 0.894	2.200 ± 1.327	OOM	OOM	OOM
		6	0.000 ± 0.000	3.200 ± 1.166	2.400 ± 1.281	OOM	OOM	OOM
		7	0.000 ± 0.000	4.800 ± 1.327	2.600 ± 0.800	OOM	OOM	OOM
		8	0.000 ± 0.000	4.400 ± 1.744	4.400 ± 1.356	OOM	OOM	OOM

Table 10
Logits for Is_Acyclic explanation graphs, averaged over 5 runs with random initial solutions.

Class	Nodes	Method	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Acyclic	5	Cyclic Logit	−8.981 ± 0.000	−1.343 ± 4.048	6.417 ± 0.709	−8.981 ± 0.000	3.007 ± 1.337	−4.913 ± 0.000
		Acyclic Logit	11.896 ± 0.000	1.835 ± 5.022	−7.456 ± 0.800	11.896 ± 0.000	−3.581 ± 1.553	6.327 ± 0.000
	6	Cyclic Logit	−8.635 ± 1.489	1.452 ± 4.292	7.333 ± 1.891	−8.981 ± 0.000	5.044 ± 1.616	−5.479 ± 0.827
		Acyclic Logit	11.460 ± 2.016	−1.577 ± 5.222	−8.490 ± 2.134	11.896 ± 0.000	−5.908 ± 1.823	7.105 ± 1.121
	7	Cyclic Logit	−10.291 ± 0.000	4.607 ± 1.646	6.520 ± 1.742	−8.981 ± 0.000	6.954 ± 1.615	−4.769 ± 1.472
		Acyclic Logit	13.742 ± 0.000	−5.415 ± 1.857	−7.573 ± 1.965	11.896 ± 0.000	−8.063 ± 1.822	6.150 ± 2.001
	8	Cyclic Logit	−10.754 ± 0.000	4.924 ± 1.633	6.372 ± 1.884	−8.981 ± 0.000	8.708 ± 1.872	−4.675 ± 1.033
		Acyclic Logit	14.409 ± 0.000	−5.771 ± 1.843	−7.400 ± 2.131	11.896 ± 0.000	−10.042 ± 2.112	6.029 ± 1.402

(*continued on next page*)

Table 10 (*continued*).

Class	Nodes	Method	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Cyclic	5	Cyclic Logit	7.194 ± 0.000	−5.769 ± 2.112	5.665 ± 1.055	−2.178 ± 3.873	3.257 ± 1.516	−5.306 ± 0.879
		Acyclic Logit	−8.333 ± 0.000	7.516 ± 2.844	−6.609 ± 1.191	2.853 ± 4.890	−3.863 ± 1.755	6.860 ± 1.192
	6	Cyclic Logit	10.260 ± 0.000	0.200 ± 4.304	6.997 ± 2.119	0.674 ± 4.435	5.378 ± 2.708	−5.640 ± 0.884
		Acyclic Logit	−11.793 ± 0.000	−0.080 ± 5.363	−8.111 ± 2.391	−0.639 ± 5.466	−6.232 ± 3.152	7.321 ± 1.199
	7	Cyclic Logit	13.488 ± 0.000	−0.293 ± 1.360	7.080 ± 1.031	1.128 ± 4.375	5.335 ± 2.873	−4.442 ± 1.139
		Acyclic Logit	−15.436 ± 0.000	0.524 ± 1.625	−8.205 ± 1.164	−1.197 ± 5.336	−6.136 ± 3.436	5.706 ± 1.549
	8	Cyclic Logit	16.870 ± 0.000	4.585 ± 1.148	7.587 ± 1.127	2.239 ± 3.562	6.821 ± 1.762	−4.824 ± 1.138
		Acyclic Logit	−19.184 ± 0.000	−5.328 ± 1.362	−8.777 ± 1.271	−2.602 ± 4.278	−7.912 ± 1.988	6.231 ± 1.548

Table 11
Logits for MUTAG explanation graphs, averaged over 5 runs with random initial solutions.

Class	Nodes	Method	MIPExplainer	GNNInterpreter	XGNN	PAGE	D4Explainer	KnowGNN
Mutagen	5	Nonmutagen Logit	−19.249 ± 0.000	−0.422 ± 5.318	3.825 ± 5.060	6.220 ± 0.209	1.450 ± 2.023	4.409 ± 0.430
		Mutagen Logit	11.269 ± 0.000	4.048 ± 7.581	−0.041 ± 5.502	−7.044 ± 0.251	−1.568 ± 2.015	−4.862 ± 0.521
	6	Nonmutagen Logit	−30.439 ± 0.000	−7.649 ± 7.386	−1.738 ± 8.144	6.126 ± 0.256	2.120 ± 1.779	4.946 ± 0.556
		Mutagen Logit	17.673 ± 0.000	18.074 ± 13.474	7.321 ± 8.283	−6.932 ± 0.307	−2.265 ± 1.779	−5.511 ± 0.674
	7	Nonmutagen Logit	−43.654 ± 0.000	−9.854 ± 13.851	−1.755 ± 3.123	5.846 ± 0.000	2.416 ± 1.185	4.303 ± 0.729
		Mutagen Logit	25.229 ± 0.000	20.275 ± 23.446	4.828 ± 6.009	−6.595 ± 0.000	−2.456 ± 1.418	−4.735 ± 0.883
	8	Nonmutagen Logit	−59.041 ± 0.000	−21.218 ± 12.229	−1.859 ± 5.728	6.033 ± 0.256	3.268 ± 1.320	4.579 ± 0.429
		Mutagen Logit	34.029 ± 0.000	41.498 ± 19.722	5.446 ± 7.365	−6.820 ± 0.307	−3.480 ± 1.596	−5.071 ± 0.520
Nonmutagen	5	Nonmutagen Logit	8.614 ± 0.000	1.887 ± 4.245	−3.481 ± 1.849	7.492 ± 0.000	2.270 ± 2.323	3.709 ± 1.586
		Mutagen Logit	−9.948 ± 0.000	−0.107 ± 5.117	8.834 ± 2.699	−8.568 ± 0.000	−2.472 ± 2.443	−4.124 ± 1.677
	6	Nonmutagen Logit	8.294 ± 0.000	7.146 ± 8.787	2.359 ± 3.898	7.492 ± 0.000	3.582 ± 0.897	4.751 ± 0.493
		Mutagen Logit	−9.555 ± 0.000	−0.851 ± 6.609	1.204 ± 6.986	−8.568 ± 0.000	−3.859 ± 1.083	−5.274 ± 0.596
	7	Nonmutagen Logit	8.086 ± 0.188	−3.194 ± 7.405	−0.841 ± 6.120	7.492 ± 0.000	0.906 ± 2.465	4.028 ± 0.997
		Mutagen Logit	−9.304 ± 0.230	10.138 ± 9.418	3.811 ± 8.576	−8.568 ± 0.000	−1.147 ± 2.280	−4.402 ± 1.208
	8	Nonmutagen Logit	7.267 ± 0.112	−5.436 ± 10.869	−9.350 ± 15.218	7.492 ± 0.000	1.803 ± 1.308	4.636 ± 0.600
		Mutagen Logit	−8.306 ± 0.141	13.410 ± 16.078	19.037 ± 22.654	−8.568 ± 0.000	−1.811 ± 1.494	−5.139 ± 0.728

Table 12
Logits for Shapes explanation graphs, averaged over 5 runs with random initial solutions.

Class	Nodes	Method	Lollipop logit	Wheel logit	Grid logit	Star logit
Grid	5	MIPExplainer	−1.343 ± 0.000	−19.762 ± 0.000	9.577 ± 0.000	−20.261 ± 0.000
		GNNInterpreter	−10.019 ± 6.822	−38.818 ± 40.875	−8.103 ± 23.501	−14.756 ± 18.554
		XGNN	−5.121 ± 1.118	3.259 ± 6.784	8.262 ± 0.688	−31.471 ± 4.109
		PAGE	−11.737 ± 0.000	−69.500 ± 0.000	2.554 ± 0.000	−5.368 ± 0.000
		D4Explainer	−1.074 ± 3.791	−29.853 ± 20.463	3.409 ± 9.946	−17.471 ± 9.216
		KnowGNN	−19.235 ± 0.000	−76.560 ± 0.000	−35.927 ± 0.000	3.033 ± 0.000
	6	MIPExplainer	−0.846 ± 0.000	−1.949 ± 0.000	9.154 ± 0.000	−31.297 ± 0.000
		GNNInterpreter	−3.386 ± 3.512	−21.545 ± 11.607	−6.520 ± 11.184	−17.602 ± 8.887
		XGNN	−4.907 ± 1.180	5.398 ± 3.124	4.816 ± 2.687	−30.366 ± 3.257
		PAGE	−11.737 ± 0.000	−69.500 ± 0.000	2.554 ± 0.000	−5.368 ± 0.000
		D4Explainer	−2.050 ± 2.131	−21.696 ± 12.831	5.361 ± 5.933	−19.014 ± 3.347
		KnowGNN	−16.871 ± 1.435	−71.344 ± 4.756	−39.940 ± 14.219	3.606 ± 4.123
	7	MIPExplainer	−1.269 ± 0.000	−11.014 ± 0.000	9.972 ± 0.000	−25.884 ± 0.000
		GNNInterpreter	−2.233 ± 2.319	−5.299 ± 5.715	3.110 ± 3.284	−23.110 ± 2.283
		XGNN	−1.976 ± 0.763	−4.893 ± 2.458	−1.793 ± 7.365	−21.878 ± 3.027
		PAGE	−11.737 ± 0.000	−69.500 ± 0.000	2.554 ± 0.000	−5.368 ± 0.000
		D4Explainer	−4.218 ± 1.879	4.195 ± 4.249	4.767 ± 2.175	−29.559 ± 1.943
		KnowGNN	−15.973 ± 1.540	−71.334 ± 4.394	−29.122 ± 5.736	0.596 ± 0.118
	8	MIPExplainer	−1.502 ± 0.000	−8.745 ± 0.000	9.270 ± 0.000	−26.275 ± 0.000
		GNNInterpreter	−2.884 ± 3.272	−11.408 ± 6.947	1.346 ± 5.852	−19.579 ± 3.193
		XGNN	−2.444 ± 1.782	−5.672 ± 8.371	−7.951 ± 13.444	−19.534 ± 7.940
		PAGE	−11.737 ± 0.000	−69.500 ± 0.000	2.554 ± 0.000	−5.368 ± 0.000
		D4Explainer	−2.016 ± 0.977	−2.345 ± 3.262	1.738 ± 0.764	−24.737 ± 1.756
		KnowGNN	−14.107 ± 1.599	−65.919 ± 3.523	−35.266 ± 6.608	0.809 ± 2.823

(*continued on next page*)

Table 12 (*continued*).

Class	Nodes	Method	Lollipop logit	Wheel logit	Grid logit	Star logit
Lollipop	5	MIPExplainer	−2.791 ± 0.000	−25.569 ± 0.000	−5.128 ± 0.000	−19.403 ± 0.000
		GNNInterpreter	−8.329 ± 8.233	−47.927 ± 27.593	−15.098 ± 19.693	−9.227 ± 13.375
		XGNN	−5.837 ± 0.697	−0.149 ± 22.146	3.951 ± 10.263	−30.313 ± 12.808
		PAGE	−3.241 ± 0.616	−16.925 ± 11.837	0.406 ± 7.577	−22.603 ± 4.382
		D4Explainer	−2.714 ± 2.925	−18.326 ± 25.050	5.298 ± 6.730	−23.279 ± 13.321
		KnowGNN	−16.735 ± 2.282	−69.864 ± 6.113	−44.351 ± 7.689	3.099 ± 0.060
	6	MIPExplainer	1.580 ± 0.000	−21.554 ± 0.000	−11.776 ± 0.000	−16.268 ± 0.000
		GNNInterpreter	−3.452 ± 3.391	−22.403 ± 11.090	−3.795 ± 8.878	−15.728 ± 7.512
		XGNN	−2.709 ± 1.291	−1.503 ± 3.706	2.021 ± 0.575	−25.136 ± 2.219
		PAGE	−2.110 ± 0.411	−16.864 ± 14.432	−3.108 ± 2.388	−18.305 ± 1.999
		D4Explainer	−3.640 ± 2.227	−2.081 ± 11.177	2.017 ± 10.966	−26.423 ± 9.002
		KnowGNN	−16.783 ± 1.646	−71.698 ± 5.047	−34.148 ± 6.345	1.528 ± 0.143
	7	MIPExplainer	2.700 ± 0.000	−18.050 ± 0.000	−5.612 ± 0.000	−21.298 ± 0.000
		GNNInterpreter	−3.375 ± 2.803	−11.131 ± 14.287	−4.683 ± 8.943	−22.799 ± 8.634
		XGNN	−3.732 ± 0.426	−1.052 ± 5.294	1.356 ± 2.642	−24.479 ± 4.387
		PAGE	0.988 ± 1.323	−21.316 ± 10.528	−1.649 ± 2.181	−13.994 ± 0.448
		D4Explainer	−2.061 ± 1.600	−11.771 ± 17.292	−0.007 ± 2.981	−21.281 ± 5.158
		KnowGNN	−15.534 ± 1.423	−69.752 ± 4.374	−31.187 ± 5.709	0.488 ± 0.132
	8	MIPExplainer	7.224 ± 0.000	−15.080 ± 0.000	−3.129 ± 0.000	−19.795 ± 0.000
		GNNInterpreter	−2.637 ± 1.622	−19.483 ± 17.827	1.137 ± 6.363	−18.262 ± 9.027
		XGNN	−2.184 ± 3.039	−13.259 ± 5.576	−6.440 ± 5.380	−14.606 ± 4.264
		PAGE	3.204 ± 2.252	−30.502 ± 7.832	−3.333 ± 1.101	−9.187 ± 1.162
		D4Explainer	−2.632 ± 2.129	−1.737 ± 7.246	0.536 ± 8.536	−24.909 ± 6.567
		KnowGNN	−13.654 ± 3.770	−66.377 ± 6.921	−29.037 ± 10.104	−0.250 ± 4.152
Star	5	MIPExplainer	−16.772 ± 0.000	−67.715 ± 0.000	−72.108 ± 0.000	12.894 ± 0.000
		GNNInterpreter	−12.575 ± 12.292	−42.824 ± 40.005	−31.701 ± 46.376	−8.555 ± 20.510
		XGNN	−5.020 ± 1.008	2.464 ± 5.862	7.990 ± 0.654	−30.609 ± 2.926
		PAGE	−23.863 ± 0.000	−80.654 ± 0.000	−62.974 ± 0.000	8.271 ± 0.000
		D4Explainer	−1.621 ± 2.657	−18.266 ± 16.228	8.843 ± 1.018	−21.837 ± 6.416
		KnowGNN	−17.568 ± 2.282	−72.096 ± 6.113	−41.543 ± 7.689	3.077 ± 0.060
	6	MIPExplainer	−13.064 ± 0.000	−64.317 ± 0.000	−48.784 ± 0.000	12.183 ± 0.000
		GNNInterpreter	−3.980 ± 2.651	−21.691 ± 9.404	−3.585 ± 12.975	−15.661 ± 2.185
		XGNN	−4.822 ± 1.125	5.352 ± 3.101	4.037 ± 2.655	−29.810 ± 3.144
		PAGE	−23.863 ± 0.000	−80.654 ± 0.000	−62.974 ± 0.000	8.271 ± 0.000
		D4Explainer	−4.689 ± 2.122	5.163 ± 7.173	7.551 ± 3.550	−32.577 ± 6.240
		KnowGNN	−16.685 ± 1.790	−71.660 ± 5.099	−34.286 ± 6.539	1.595 ± 0.128
	7	MIPExplainer	−11.148 ± 0.000	−61.412 ± 0.000	−33.850 ± 0.000	13.072 ± 0.000
		GNNInterpreter	−2.507 ± 2.511	−22.296 ± 21.034	−3.210 ± 13.723	−18.741 ± 8.891
		XGNN	−1.469 ± 2.153	−9.213 ± 5.345	−6.549 ± 8.455	−18.000 ± 4.776
		PAGE	−23.863 ± 0.000	−80.654 ± 0.000	−62.974 ± 0.000	8.271 ± 0.000
		D4Explainer	−3.464 ± 2.565	−0.203 ± 9.555	2.798 ± 4.728	−26.928 ± 7.178
		KnowGNN	−16.053 ± 1.423	−71.349 ± 4.374	−29.102 ± 5.709	0.536 ± 0.132
	8	MIPExplainer	−10.722 ± 0.000	−58.630 ± 0.000	−23.479 ± 0.000	14.536 ± 0.000
		GNNInterpreter	−3.702 ± 1.427	−7.587 ± 15.924	−1.624 ± 8.923	−21.715 ± 8.531
		XGNN	−1.924 ± 1.894	−8.287 ± 9.256	−11.054 ± 12.549	−18.099 ± 8.888
		PAGE	−23.863 ± 0.000	−80.654 ± 0.000	−62.974 ± 0.000	8.271 ± 0.000
		D4Explainer	−3.208 ± 2.232	−0.567 ± 6.732	2.633 ± 3.657	−25.972 ± 5.453
		KnowGNN	−14.111 ± 1.439	−65.935 ± 3.457	−35.179 ± 6.498	0.813 ± 2.820

(*continued on next page*)

Table 12 (*continued*).

Class	Nodes	Method	Lollipop logit	Wheel logit	Grid logit	Star logit
Wheel	5	MIPExplainer	−6.180 ± 0.000	13.062 ± 0.000	10.323 ± 0.000	−40.319 ± 0.000
		GNNInterpreter	−7.290 ± 7.960	−42.230 ± 28.326	−3.870 ± 18.922	−12.452 ± 10.727
		XGNN	−5.957 ± 0.275	8.335 ± 2.176	8.057 ± 0.745	−34.470 ± 2.362
		PAGE	0.012 ± 1.856	−23.416 ± 5.004	9.200 ± 0.516	−20.101 ± 0.218
		D4Explainer	−0.106 ± 3.509	−24.328 ± 18.604	6.844 ± 2.234	−20.185 ± 6.592
		KnowGNN	−18.193 ± 2.084	−73.770 ± 5.580	−39.437 ± 7.019	3.060 ± 0.055
	6	MIPExplainer	−5.616 ± 0.000	6.921 ± 0.000	2.541 ± 0.000	−29.458 ± 0.000
		GNNInterpreter	−4.918 ± 4.503	−18.368 ± 31.653	3.615 ± 6.467	−22.223 ± 14.202
		XGNN	−3.865 ± 1.889	1.860 ± 6.420	3.038 ± 3.642	−27.369 ± 5.667
		PAGE	−0.664 ± 3.710	−22.743 ± 8.427	8.564 ± 0.097	−20.198 ± 0.460
		D4Explainer	−4.891 ± 1.793	−4.316 ± 23.429	7.055 ± 5.290	−27.932 ± 14.974
		KnowGNN	−15.187 ± 1.565	−67.861 ± 4.205	−39.333 ± 5.489	1.689 ± 0.032
	7	MIPExplainer	−5.723 ± 0.000	8.478 ± 0.000	4.220 ± 0.000	−30.816 ± 0.000
		GNNInterpreter	−4.489 ± 2.121	−5.192 ± 15.185	5.168 ± 3.163	−24.700 ± 9.701
		XGNN	−1.477 ± 4.376	−8.111 ± 10.727	−4.639 ± 12.694	−20.903 ± 8.152
		PAGE	0.436 ± 3.599	−24.944 ± 8.838	7.936 ± 1.563	−19.917 ± 0.122
		D4Explainer	−4.297 ± 2.530	−0.655 ± 19.448	6.685 ± 2.795	−29.687 ± 9.721
		KnowGNN	−15.454 ± 1.505	−69.737 ± 4.388	−31.206 ± 5.727	0.548 ± 0.145
	8	MIPExplainer	−4.686 ± 1.538	5.669 ± 5.114	1.911 ± 0.883	−28.019 ± 2.646
		GNNInterpreter	−2.193 ± 0.699	−5.022 ± 6.210	−5.701 ± 13.464	−20.266 ± 6.735
		XGNN	−1.513 ± 2.686	−11.867 ± 4.893	−3.223 ± 12.412	−17.747 ± 6.596
		PAGE	−1.752 ± 3.467	−15.373 ± 12.346	3.354 ± 4.821	−19.611 ± 0.229
		D4Explainer	−2.942 ± 2.183	−2.672 ± 8.559	0.716 ± 5.848	−24.973 ± 5.856
		KnowGNN	−14.515 ± 1.147	−67.331 ± 4.279	−33.354 ± 11.934	0.913 ± 2.812

References

[1] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, URL https://openreview.net/forum?id=SJU4ayYgl.

[2] W.L. Hamilton, R. Ying, J. Leskovec, Inductiverepresentation learning on large graphs, 2017, http://dx.doi.org/10.48550/arXiv.1706.02216, URL https://arxiv.org/abs/1706.02216v4.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, 2017, http://dx.doi.org/10.48550/ARXIV.1710.10903, URL https://arxiv.org/abs/1710.10903.

[4] Z. Zhang, Q. Liu, H. Wang, C. Lu, C. Lee, ProtGNN: Towards self-explaining graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, (no. 8) 2022, pp. 9127–9135, http://dx.doi.org/10.1609/aaai.v36i8.20898, URL https://ojs.aaai.org/index.php/AAAI/article/view/20898.

[5] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, R. He, Graph information bottleneck for subgraph recognition, in: International Conference on Learning Representations, 2020, URL https://openreview.net/forum?id=bM4Iqfg8M2k.

[6] P. Müller, L. Faber, K. Martinkus, R. Wattenhofer, GraphChef: Learning the recipe of your dataset, 2023, URL https://openreview.net/forum?id=ZgYZH5PFEg all.

[7] M. Liu, Y. Luo, L. Wang, Y. Xie, H. Yuan, S. Gui, Z. Xu, H. Yu, J. Zhang, Y. Liu, K. Yan, B. Oztekin, H. Liu, X. Zhang, C. Fu, S. Ji, DIG: A turnkey library for diving into graph deep learning research, 2021, arXiv preprint arXiv:2103.12608.

[8] H. Yuan, H. Yu, S. Gui, S. Ji, Explainability in graph neural networks: A taxonomic survey, IEEE Trans. Pattern Anal. Mach. Intell. 45 (5) (2023) 5782–5799, http://dx.doi.org/10.1109/TPAMI.2022.3204236, URL https://ieeexplore.ieee.org/abstract/document/9875989.

[9] J. Kakkad, J. Jannu, K. Sharma, C.C. Aggarwal, S. Medya, A survey on explainability of graph neural networks, 2023, http://dx.doi.org/10.48550/ARXIV.2306.01958, CoRR abs/2306.01958. arXiv:2306.01958.

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

[11] S. Lu, K.G. Mills, J. He, B. Liu, D. Niu, GOAt: Explaining graph neural networks via graph output attribution, in: The Twelfth International Conference on Learning Representations, 2024, URL https://openreview.net/forum?id=2Q8TZWAHv4.

[12] H. Yuan, H. Yu, J. Wang, K. Li, S. Ji, On explainability of graph neural networks via subgraph explorations, in: International Conference on Machine Learning, PMLR, 2021, pp. 12241–12252.

[13] D. Luo, T. Zhao, W. Cheng, D. Xu, F. Han, W. Yu, X. Liu, H. Chen, X. Zhang, Towards inductive and efficient explanations for graph neural networks, IEEE Trans. Pattern Anal. Mach. Intell. 46 (8) (2024) 5245–5259, http://dx.doi.org/10.1109/TPAMI.2024.3362584.

[14] Z. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, Gnnexplainer: Generating explanations for graph neural networks, Adv. Neural Inf. Process. Syst. 32 (2019).

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

[16] S. Lu, B. Liu, K.G. Mills, J. He, D. Niu, EiG-search: Generating edge-induced subgraphs for GNN explanation in linear time, 2024, URL http://arxiv.org/abs/2405.01762, arXiv:2405.01762 [cs].

[17] M. Vu, M.T. Thai, Pgm-explainer: Probabilistic graphical model explanations for graph neural networks, Adv. Neural Inf. Process. Syst. 33 (2020) 12225–12235.

[18] W. Lin, H. Lan, B. Li, Generative causal explanations for graph neural networks, in: International Conference on Machine Learning, PMLR, 2021, pp. 6666–6679, URL https://arxiv.org/pdf/2104.06643.pdf.

[19] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K.T. Schütt, K.R. Müller, G. Montavon, Higher-order explanations of graph neural networks via relevant walks, IEEE Trans. Pattern Anal. Mach. Intell. 44 (11) (2021) 7581–7596, Publisher: IEEE.

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

[21] W. Zhang, X. Li, W. Nejdl, Adversarial mask explainer for graph neural networks, in: Proceedings of the ACM on Web Conference 2024, ACM, Singapore Singapore, 2024, pp. 861–869, http://dx.doi.org/10.1145/3589334.3645608, URL https://dl.acm.org/doi/10.1145/3589334.3645608.

[22] S. Azzolin, A. Longa, P. Barbiero, P. Lio, A. Passerini, Global explainability of GNNs via logic combination of learned concepts, in: The Eleventh International Conference on Learning Representations, 2023, URL https://openreview.net/forum?id=OTbRTIY4YS.

[23] Y.M. Shin, S.W. Kim, W.Y. Shin, PAGE: Prototype-based model-level explanations for graph neural networks, 2022, http://dx.doi.org/10.48550/ARXIV.2210.17159, URL https://arxiv.org/abs/2210.17159.

[24] L.C. Magister, D. Kazhdan, V. Singh, P. Liò, GCExplainer: Human-in-the-loop concept-based explanations for graph neural networks, in: Workshop on Human in the Loop Learning, vol. 3, ICML, 2021, http://dx.doi.org/10.48550/ARXIV.2107.11889, URL https://arxiv.org/abs/2107.11889.

[25] Z. Yu, H. Gao, MAGE: Model-level graph neural networks explanations via motif-based graph generation, 2024, arXiv:2405.12519 [cs, q-bio]. URL http://arxiv.org/abs/2405.12519.

[26] H. Xuanyuan, P. Barbiero, D. Georgiev, L.C. Magister, P. Liò, 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, in: AAAI'23/IAAI'23/EAAI'23, vol. 37, AAAI Press, 2023, pp. 10675–10683, http://dx.doi.org/10.1609/aaai.v37i9.26267.

[27] H. Yuan, J. Tang, X. Hu, S. Ji, XGNN: Towards model-level explanations of graph neural networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, Virtual Event CA USA, 2020, pp. 430–438, http://dx.doi.org/10.1145/3394486.3403085, URL https://dl.acm.org/doi/10.1145/3394486.3403085.

[28] S. Saha, M. Das, S. Bandyopadhyay, GraphEx: A user-centric model-level explainer for graph neural networks, in: K. Maughan, R. Liu, T.F. Burns (Eds.), The First Tiny Papers Track At ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023, OpenReview.net, 2023, URL https://openreview.net/pdf?id=CuE1F1M0_yR.

[29] X. Wang, H.W. Shen, GNNInterpreter: A probabilistic generative model-level explanation for graph neural networks, 2022, http://dx.doi.org/10.48550/ARXIV.2209.07924, URL https://arxiv.org/abs/2209.07924.

[30] J. Chen, S. Wu, A. Gupta, R. Ying, D4Explainer: In-distribution explanations of graph neural network via discrete denoising diffusion, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), in: Advances in Neural Information Processing Systems, vol. 36, Curran Associates, Inc., 2023, pp. 78964–78986, URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f978c8f3b5f399cae464e85f72e28503-Paper-Conference.pdf.

[31] Y. Nian, Y. Chang, W. Jin, L. Lin, Globally interpretable graph learning via distribution matching, in: Proceedings of the ACM on Web Conference 2024, WWW '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 992–1002, http://dx.doi.org/10.1145/3589334.3645674, URL https://dl.acm.org/doi/10.1145/3589334.3645674.

[32] S. Saha, M. Das, S. Bandyopadhyay, GraphEx: A user-centric model-level explainer for graph neural networks, in: K. Maughan, R. Liu, T.F. Burns (Eds.), The First Tiny Papers Track At ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023, OpenReview.net, 2023, URL https://openreview.net/pdf?id=CuE1F1M0_yR.

[33] Y. Ma, X. Liu, C. Guo, B. Jin, H. Liu, KnowGNN: a knowledge-aware and structure-sensitive model-level explainer for graph neural networks, Appl. Intell. 55 (2) (2024) 126, http://dx.doi.org/10.1007/s10489-024-06034-4.

[34] C.H. Cheng, G. Nührenberg, H. Ruess, Maximum resilience of artificial neural networks, in: D. D'Souza, K. Narayan Kumar (Eds.), Automated Technology for Verification and Analysis, Springer International Publishing, Cham, 2017, pp. 251–268, http://dx.doi.org/10.1007/978-3-319-68167-2_18, GSCC: 0000334 92 citations (Crossref) [2024-05-30] 255 citations (Semantic Scholar/DOI) [2024-05-30].

[35] R.R. Bunel, I. Turkaslan, P. Torr, P. Kohli, P.K. Mudigonda, A unified view of piecewise linear neural network verification, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), in: Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018, GSCC: 0000413. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/be53d253d6bc3258a8160556dda3e9b2-Paper.pdf.

[36] E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, R. Misener, Efficient verification of ReLU-based neural networks via dependency analysis, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, (no. 04) 2020, pp. 3291–3299, http://dx.doi.org/10.1609/aaai.v34i04.5729, GSCC: 0000170 102 citations (Semantic Scholar/DOI) [2024-05-30] 60 citations (Crossref) [2024-05-30]. URL https://ojs.aaai.org/index.php/AAAI/article/view/5729.

[37] M.S. Cheon, An outer-approximation guided optimization approach for constrained neural network inverse problems, Math. Program.: Ser. A B 196 (1–2) (2022) 173–202, http://dx.doi.org/10.1007/s10107-021-01653-y.

[38] N. Ansari, H.P. Seidel, V. Babaei, Mixed integer neural inverse design, ACM Trans. Graph. 41 (4) (2022) 151:1–151:14, http://dx.doi.org/10.1145/3528223.3530083, GSCC: 0000268 4 citations (Semantic Scholar/DOI) [2024-05-30] 0 citations (Crossref) [2024-05-30]. URL https://dl.acm.org/doi/10.1145/3528223.3530083.

[39] A. Ignatiev, N. Narodytska, J. Marques-Silva, Abduction-based explanations for machine learning models, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, (no. 01) 2019, pp. 1511–1519, http://dx.doi.org/10.1609/aaai.v33i01.33011511, GSCC: 0000250 179 citations (Semantic Scholar/DOI) [2024-05-30] 57 citations (Crossref) [2024-05-30]. URL https://ojs.aaai.org/index.php/AAAI/article/view/3964.

[40] T. McDonald, C. Tsay, A.M. Schweidtmann, N. Yorke-Smith, Mixed-integer optimisation of graph neural networks for computer-aided molecular design, Comput. Chem. Eng. 185 (2024) 108660, http://dx.doi.org/10.1016/j.compchemeng.2024.108660, URL https://linkinghub.elsevier.com/retrieve/pii/S0098135424000784.

[41] S. Zhang, J. Campos, C. Feldmann, D. Walz, F. Sandfort, M. Mathea, C. Tsay, R. Misener, Optimizing over trained GNNs via symmetry breaking, Adv. Neural Inf. Process. Syst. 36 (2023) 44898–44924, URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c8cd1b78cdae751265c88efc136e5bd-Abstract-Conference.html.

[42] X. Gao, B. Xiao, D. Tao, X. Li, A survey of graph edit distance, Pattern Anal. Appl. 13 (2010) 113–129, Publisher: Springer.

[43] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. Syst. Man Cybern. SMC-13 (3) (1983) 353–362, http://dx.doi.org/10.1109/TSMC.1983.6313167, URL http://ieeexplore.ieee.org/document/6313167/.

[44] L.A. Wolsey, Integer Programming, second ed., Wiley, Hoboken, NJ Chichester, West Sussex, 2021.

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

[46] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? 2018, http://dx.doi.org/10.48550/ARXIV.1810.00826, URL https://arxiv.org/abs/1810.00826.

[47] A.H. Land, A.G. Doig, An automatic method of solving discrete programming problems, Econometrica 28 (3) (1960) 497, http://dx.doi.org/10.2307/1910129, GSCC: 0000399 2372 citations (Semantic Scholar/DOI) [2024-05-30] 1465 citations (Crossref) [2024-05-30] QID: Q55934470. URL https://www.jstor.org/stable/1910129?origin=crossref.

[48] Gurobi Optimization, LLC, Gurobi optimizer reference manual, 2023, GSCC: 0000006. URL https://www.gurobi.com.

[49] A.K. Debnath, R.L. Lopez De Compadre, G. Debnath, A.J. Shusterman, C. Hansch, Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity, J. Med. Chem. 34 (2) (1991) 786–797, http://dx.doi.org/10.1021/jm00106a046, URL https://pubs.acs.org/doi/abs/10.1021/jm00106a046.

[50] K.H. Hsu, B.H. Su, Y.S. Tu, O.A. Lin, Y.J. Tseng, Mutagenicity in a molecule: Identification of core structural features of mutagenicity using a scaffold analysis, PLoS One 11 (2) (2016) e0148900, http://dx.doi.org/10.1371/journal.pone.0148900, URL https://dx.plos.org/10.1371/journal.pone.0148900.

[51] N. Wale, G. Karypis, Comparison of descriptor spaces for chemical compound retrieval and classification, in: Sixth International Conference on Data Mining, ICDM'06, 2006, pp. 678–689, http://dx.doi.org/10.1109/ICDM.2006.39.

[52] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 1365–1374, http://dx.doi.org/10.1145/2783258.2783417, event-place: Sydney, NSW, Australia.

[53] M. Fey, J.E. Lenssen, Fast graph representation learning with PyTorch geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[54] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (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.

[55] J. Huchette, J.P. Vielma, Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools, Oper. Res. 71 (5) (2023) 1835–1856, http://dx.doi.org/10.1287/opre.2019.1973, URL https://pubsonline.informs.org/doi/10.1287/opre.2019.1973.

[56] C. Tsay, J. Kronqvist, A. Thebelt, R. Misener, Partition-based formulations for mixed-integer optimization of trained ReLU neural networks, in: A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021, URL https://openreview.net/forum?id=jhd62iKzRuj.

[57] R. Anderson, J. Huchette, C. Tjandraatmadja, J.P. Vielma, Strong mixed-integer programming formulations for trained neural networks, in: A. Lodi, V. Nagarajan (Eds.), Integer Programming and Combinatorial Optimization, Springer International Publishing, Cham, 2019, pp. 27–42.

Blake Gaines is a Ph.D. student studying under Prof. Jinbo Bi in the School of Computing at the University of Connecticut, where he also completed a BS in Computer Science and a BA in Mathematics. His Ph.D. research has been focused on the geometric properties of neural networks for explainability. He can be contacted at blake.gaines@uconn.edu

Chunjiang Zhu is an assistant professor in the Department of Computer Science at the University of North Carolina at Greensboro. He received his Ph.D. in Computer Science from City University of Hong Kong and completed his postdoc training at the University of Connecticut. His research interests include Machine Learning and Theory, Graph Algorithms, and AI for Science and Education. He can be contacted at chunjiang.zhu@uncg.edu

Jinbo Bi is the Frederick H Leonhardt Professor of Computer Science in the School of Computing at the University of Connecticut. With over 20 years of experience in AI and biomedical research, Prof. Bi has made significant contributions to the fields of machine learning, artificial intelligence, and their applications in healthcare and life science. She can be contacted at jinbo.bi@uconn.edu