

Qualifying Exam

Art Tay

Introduction

- Overview of the problem area.
 - Graphs are an important data structure.
 - * GRAPHS provide an incredibly flexible structure for modeling complex data. Data can naturally appear as graphs, like molecules. We can reduce data to a graph, such as the key points of a image. We can even use graphs to add structure, such as grammatical relationships.
 - GNN models are good at prediction and inference on graph data.
 - * Graph Neural Networks (GNNs) have become a popular choice for prediction and inference on graph data. At their core, GNNs work by iteratively updating node embeddings based on information from neighboring nodes. The idea is to use the graph's structure to engineer better features. This message passing scheme allows GNNs to capture complex dependencies and patterns present within the graph structure. GNN architectures typically consist of multiple layers, each performing message passing and aggregation operations to refine the embeddings. These layers are often followed by pooling and dense prediction layers to produce the final output.
 - There are many important applications for graph classification models.
 - * Some important applications of graph classification include predicting chemical toxicity (Bai et al. 2019), classifying proteins (Gallicchio and Micheli 2019), and even detecting cancer from pathology slides (Xiao et al. 2023).
 - **Problem:** While GNNs achieve remarkable predictive power, their complexity prevents the exaction of the scientific rationale.
- Why is the problem important?
 - Explaining or interpreting GNN predictions would
 - * help with the adoption of such models for critical applications,
 - * prevent adversarial attacks,
 - * detect potential implicit discrimination,
 - * guide scientific as well as machine learning research.
- How does the problem relate to the fundamentals areas of Statistics?
 - Explain-ability vs Interpretability
 - * Yuan et al. (2022)
 - * A model is interpretable if the models decision process can be readily understood by humans. For example, a linear regression model is interpretable because the coefficient clearly define how any prediction get made.

- * A model is explainable if the models prediction can be reasoned post-hoc. Permuting each variable and measuring the variation in the predictions can be used to estimate each variables marginal effect [cite].
- One goal would be to create a GNN type model whose decision process is human interpretable. A straight translation from statistics would be a circuit type analysis [cite]. For graphs, this would mean some form of coefficients on subgraphs producing the prediction.
- Another goal might be to develop a method that determines if a feature is statistical significant to the GNN model. The challenge is that the graph features that matter to researchers aren't necessarily tabular.
- What is the impact of solving this problem?
 - In the application where GNNs have shown strong predictive power, we can exact a testable scientific hypothesis for the nature of the classification.
 - In the application where GNNs have weak predictive power, highlight the potential misunderstandings the model is having.

Notation

- Let G denote a graph.
- Any graph G can be describe by X, A, E . The node feature matrix, edge feature matrix, and adjacency matrix respectively
- Let $X = [X_c, X_d]$, where X_c is the subset of continuous node features and X_d is the subset of one-hot discrete node features.
- Let $E = [E_c, E_d]$, denoted in the same manner.
- Let n represent the number of nodes in the graph and v represent the number of edges.
- Let $\text{feat}_{(\cdot)}$ denote the number of features or columns in the the corresponding feature matrix.
- A is a binary $n \times n$ matrix where $A[i, j] = 1$ indicates that an edge exists between nodes labeled i and j .
- Let $\text{explainee}(G; \Omega) = h_G^{(1)}, \dots, h_G^{(L)}, \rho_G$ be an L layer GNN model with parameters Ω that we would like to explain.
- Let \hat{Y}_G be the predicted class label for graph G predicted from $\text{explainee}()$.
- For any graph Let ν denote the set of node and \mathcal{E} denote the set of edges.

Analysis of Core Papers

GNNInterpreter

(X. Wang and Shen 2024)

- Overview
 - Instance v. Model Level

- * In general, explanation methods serve to elucidate which features within the data influence disparate predictions. These methods typically fall into two categories: instance-level and model-level. Instance-level explanations aim to unveil the model’s rationale behind a particular prediction. In domains such as image and text analysis, a prevalent approach involves masking or perturbing the instance and assessing the impact on the model’s prediction. On the other hand, model-level explanations seek to understand how a model generally distinguishes between classes. In image and text analysis, for instance, one common technique involves treating the input as a trainable parameter and optimizing the model’s prediction towards a specific class. Consequently, the resulting optimized input comprises a set of features strongly associated with the targeted class.
- GNNInterpreter provides model level explanations for GNN in this manner.
- Formally, GNNInterpreter tries to learn the graph generating distribution for each class.
- GNNInterpreter works by optimizing the parameters of a generic graph generating distribution to produce samples that closely match the explaine’s understanding of the targeted class.
- Explanation of the graph generating distribution.
 - Graph generating distributions are hard to specify because there can be discrete and continuous elements of X , E and A . Furthermore, the interactions between these matrices can be complex.
 - The authors tackle these issues by making two simplify assumptions.
 1. Assume that G is a *Gilbert* random graph, every possible edge as an independent fixed probability of occurring.

$$\forall (i, j) \neq (k, l) \Pr(A[i, j] = 1) \perp \Pr(A[k, l] = 1) \quad (1)$$
 2. The features of every node and edge are independently distributed.
 - The author justify these assumptions by:
 1. The other graph distributions aren’t suitable.
 - a. Erdo-Renyi graphs have a fixed number of edges (and nodes, but nodes are also fixed for Gilbert).
 - b. Rado graphs are infinite in size.
 - c. The random dot-product graph model is just a generalization of Gilbert random graphs.
 2. Because the parameters of the independent distributions will be updated jointly using the *explaine* model, the *explaine*’s understanding of the latent correlation structure should be contained in the final estimates.
 - X_c and E_c can be sampled from any continuous distribution that can be expressed as a location-scale family. Separating the stochastic and systematic components is necessary for gradient based optimization. It is commonly known as the “re-parametrization trick”.
 - X_d , E_d as well as A need to be sampled from a continuous distribution for gradient based optimization, but the distribution has to have sampling properties close to a discrete distribution.

- The author assume that the true underlying distribution for every discrete node and edge feature is *categorical*. The categorical distribution is also know as the multi-bernoulli, where every sample has a fixed probability of being in one of the discrete categories.
- Suppose there are D categories with probabilities $\pi_\omega = \frac{\theta_\omega}{\sum_{i \in D} \theta_i}$. Then

$$I = \operatorname{argmax}_{i \in D} \log \theta_i + G^{(i)} \sim \operatorname{Cat}(\pi) \quad (2)$$

where $G^{(i)} \stackrel{i.i.d.}{\sim} \operatorname{Gumbel}(0, 1)$.

- The intuition is that the Gumbel or extreme value distribution is the density of the maximum order statistic of i.i.d. standard normals which makes it a good candidate for model the winning or maximum probability category. Adding Gumbel noise to the logits should maintain the true relative proportions, but enough skewness such that every category has some probability of having the maximum noised logit.
- **Proof 1:** In order for I to be a true categorical distribution, $\Pr[I = \omega] = \pi_\omega$. $I = \omega$ if and only if $\log \theta_\omega + G^{(\omega)} > \log \theta_i + G^i \forall i \in D \setminus \omega$. Let M_i denote a random variable that follows a $\operatorname{Gumbel}(\log \theta_i, 1)$ distribution.

$$\begin{aligned}
\Pr[I = \omega] &= \mathbb{E}_{M_\omega} \prod_{i \in D \setminus \omega} \Pr(M_i < m_\omega) \text{ i.i.d location shifted Gumbel distributions.} \\
&= \mathbb{E}_{M_\omega} \prod_{i \in D \setminus \omega} \exp(-e^{\log \theta_i - m_\omega}) \text{ Gumbel CDF.} \\
&= \mathbb{E}_{M_\omega} \exp\left(-\sum_{i \in D \setminus \omega} e^{\log \theta_i - m_\omega}\right) \\
&= \int_{-\infty}^{\infty} \exp(\log \theta_\omega - m_\omega) \exp(-e^{\log \theta_\omega - m_\omega}) \cdot \exp\left(-\sum_{i \in D \setminus \omega} e^{\log \theta_i - m_\omega}\right) dm \\
&\quad \text{Gumbel PDF.} \\
&= \int_{-\infty}^{\infty} \exp(\log \theta_\omega - m_\omega) \exp\left(-\sum_{i \in D} e^{\log \theta_i - m_\omega}\right) dm \\
&= \int_{-\infty}^{\infty} \theta_\omega \exp(-m_\omega) \exp\left(-e^{-m_\omega} \sum_{i \in D} \theta_i\right) dm \\
&= \pi_\omega \sum_{i \in D} \theta_i \int_{-\infty}^{\infty} \exp(-m_\omega) \exp\left(-e^{-m_\omega} \sum_{i \in D} \theta_i\right) dm \text{ From the above definition of } \pi_\omega \\
&= \pi_\omega \sum_{i \in D} \theta_i \frac{\exp\left(-e^{-m_\omega} \sum_{i \in D} \theta_i\right)}{\sum_{i \in D} \theta_i} \Big|_{-\infty}^{\infty} \\
&= \pi_\omega \sum_{i \in D} \theta_i \frac{1}{\sum_{i \in D} \theta_i} = \pi_\omega
\end{aligned}$$

Reference: Huijben et al. (2022)

- Using inverse CDF sampling and relaxing the argmax to a Softmax, we can sample one-hot categorical vectors based on two parameters θ_{Cat} , a trainable parameter vector

of length equal to the number of categories, and τ , a hyperparameter that controls the degree of relaxation (smaller value approximate the discrete sampling better, but can result in numerical issues).

$$\text{Softmax}\left(\frac{\theta_{\text{Cat}} - \log(-\log \epsilon)}{\tau}\right), \quad \epsilon \sim U[0, 1] \quad (3)$$

This method, known as the concrete distribution (Maddison, Mnih, and Teh 2017), yields a reasonable smooth gradient w.r.t. to the probability parameters.

- The adjacency matrix can be sampled in a similar manner since the Bernoulli is just a special case of the categorical.

$$\text{sigmoid}\left(\frac{\theta_A + \log \epsilon - \log(1 - \log \epsilon)}{\tau}\right) \quad (4)$$

This is known as the binary concrete distribution (Maddison, Mnih, and Teh 2017).

- Notate the combined graph generating distribution as:

$$G_{\text{gen}} \sim \text{gen}(\Theta)$$

where Θ is the set of all parameters from the independently sampled distributions.

- Prediction objective.

- An obvious objective is to maximize the likelihood that the *explainee* model predicts a sampled graph to be a member of the target class.
- Let $\tilde{\rho}$ denote the desired predicted probability vector. Then the above objective can be expressed as:

$$\mathcal{L}_{\text{pred}}(\Theta \mid G_{\text{gen}}) = \mathbb{E}_{G_{\text{gen}}} \text{CrtEnt}(\text{explainee}(G_{\text{gen}}), \tilde{\rho}) \quad (5)$$

- Embedding objective.

- While the above objective enforces a desirable property, it isn't restrictive enough to make the generated graph realistic. This is because the final prediction, $\rho_{G_{\text{gen}}}$ is compute using only final embeddings, $h_{G_{\text{gen}}}^{(L)}$. Normally $h_{G_{\text{gen}}}^{(L)}$ contains all the needed information from the graphs structure; however, the generation scheme allows the feature distribution to be optimized directly. This means that explanation can ignore the graph structure and optimize towards the desired final embeddings.
- Another way of understanding the problems with the above objective is to consider the *out-of-distribution* (ood) issue. Since the above generation scheme is not restricted by the observed data distribution, the initial generated graphs may be very ood, but clearly on one side of the decision boundary.

- The author find that empirically GNN model exhibit a class preference.

Table 5: The quantitative evaluation results for all 6 datasets including Is_Acyclic. As the quantitative metric, we compute the average class probability of 1000 explanation graphs and the standard deviation of them for the two classes. In addition, the average training time per class of training 100 different GNNInterpreter and XGNN models is also included for efficiency evaluation.

Dataset [Method]	Predicted Class Probability by GNN				Training Time Per Class
Is_Acyclic [XGNN]	Cyclic 0.076 ± 0.000	Acyclic 0.927 ± 0.000			45 s
Is_Acyclic [Ours]	Cyclic 0.999 ± 0.001	Acyclic 1.000 ± 0.000			20 s
Is_Acyclic [Random]	Cyclic 0.143 ± 0.155	Acyclic 0.857 ± 0.155			-
MUTAG [XGNN]	Mutagen 0.986 ± 0.057	Nonmutagen 0.991 ± 0.083			128 s
MUTAG [Ours]	Mutagen 1.000 ± 0.000	Nonmutagen 1.000 ± 0.000			12 s
MUTAG [Random]	Mutagen 0.068 ± 0.251	Nonmutagen 0.932 ± 0.251			-
ColorConsistency [Ours]	Consistent 0.968 ± 0.110	Inconsistent 1.000 ± 0.000			58 s
ColorConsistency [Random]	Consistent 0.017 ± 0.118	Inconsistent 0.983 ± 0.118			-
Cyclicity [Ours]	Red Cyclic 1.000 ± 0.000	Green Cyclic 1.000 ± 0.000	Acyclic 1.000 ± 0.000		49 s
Cyclicity [Random]	Red Cyclic 0.023 ± 0.143	Green Cyclic 0.015 ± 0.118	Acyclic 0.962 ± 0.183		-
Motif [Ours]	House 0.918 ± 0.268	House-X 0.999 ± 0.032	Complete-4 1.000 ± 0.000	Complete-5 0.998 ± 0.045	83 s
Motif [Random]	House 0.000 ± 0.000	House-X 0.000 ± 0.000	Complete-4 0.000 ± 0.000	Complete-5 0.000 ± 0.000	-
Shape [Ours]	Lollipop 0.742 ± 0.360	Wheel 0.989 ± 0.100	Grid 0.996 ± 0.032	Star 1.000 ± 0.000	24 s
Shape [Random]	Lollipop 0.214 ± 0.301	Wheel 0.000 ± 0.000	Grid 0.151 ± 0.307	Star 0.000 ± 0.000	-

(X. Wang and Shen 2024)

For example, random graphs have an average predicted probability of being Non-mutagenic in the MUTAG dataset if 93.2%. This demonstrates why the above objective is insufficient to generate realistic or *in-distribution* (id) graph.

- In order to mitigate this issue, the author proposed additional minimizing the cosine distance between the average embedding of all the observed graph from the targeted class, $\bar{h}_{G_c}^{(L)}$, and the embedding of the generated explanation.

$$\mathcal{L}_{\text{embed}}(\Theta \mid G_{\text{gen}}) = \mathbb{E}_{G_{\text{gen}}} \text{CosDist} \left(\bar{h}_{G_c}^{(L)}, h_{G_{\text{gen}}}^{(L)} \right) \quad (6)$$

- Regularization terms.

- Sparse graphs are easy for humans to interpret. To encourage sparsity the authors employed an L_1 , L_2 , and a budget penalty on the edge probabilities.

$$\mathcal{L}_{\text{Sparsity}}(\theta_A) = \|\theta_A\|_1 + \|\theta_A\|_2 + \text{softplus}(\text{sigmoid}\|\theta_A\|_1 - B)^2 \quad (7)$$

where B is the expected maximum number of edge for generated explanation graphs.

- Connectivity is another desirable property as it ensures a cohesive explanation. To encourage connectivity the author minimize the *KL-Divergence* between edge probabilities that share a common node.

$$\mathcal{L}_{\text{Connect}}(\theta_A) = \sum_{i \in \mathcal{V}} \sum_{j, k \in \mathcal{E}(i)} D_{KL}(\text{sigmoid}(\theta_A[i, j]) \parallel \text{sigmoid}(\theta_A[i, k])) \quad (8)$$

where $\mathcal{E}(i)$ is the set of edges that connect to node i .

- Summary of Results + Figures

- The final generator model is trained by sampling $G_{\text{gen}} \sim \text{gen}(\Theta)$ and then iterative updated Θ via gradient descent on the full loss:

$$\mathcal{L}_{\text{GNNInterpreter}}(\Theta \mid G_{\text{gen}}) = \mathcal{L}_{\text{pred}}(\Theta \mid G_{\text{gen}}) + \mathcal{L}_{\text{embed}}(\Theta \mid G_{\text{gen}}) + \mathcal{L}_{\text{sparsity}}(\Theta \mid G_{\text{gen}}) + \mathcal{L}_{\text{connect}}(\Theta \mid G_{\text{gen}}) \quad (9)$$

Table 2: The quantitative results for 4 datasets. As the quantitative metric, we compute the average class probability of 1000 explanation graphs and the standard deviation of them for every class in all 4 datasets. In addition, the average training time per class of training 100 different GNNInterpreter and XGNN models, is also included for efficiency evaluation.

Dataset [Method]	Predicted Class Probability by GNN				Training Time Per Class
MUTAG [XGNN]	Mutagen	Nonmutagen			128 s
	0.986 ± 0.057	0.991 ± 0.083			
MUTAG [Ours]	Mutagen	Nonmutagen			12 s
	1.000 ± 0.000	1.000 ± 0.000			
Cyclicity [Ours]	Red Cyclic	Green Cyclic	Acyclic		49 s
	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000		
Motif [Ours]	House	House-X	Complete-4	Complete-5	83 s
	0.918 ± 0.268	0.999 ± 0.032	1.000 ± 0.000	0.998 ± 0.045	
Shape [Ours]	Lollipop	Wheel	Grid	Star	24 s
	0.742 ± 0.360	0.989 ± 0.100	0.996 ± 0.032	1.000 ± 0.000	

- GNNInterpreter achieve remarkable accuracy on most target classes. Many of the interval are tight and very close to 1, which implies that the examples generated are almost always classified as the targeted class. The explanations for the house motif and the lollipop shape are worse in terms of predictions. Although the author critique the use of predictions as the sole objective, they do not use any other quantitative metric to evaluate the validity of their explanations.
- Qualitatively we can see some limitation in terms of realism.
- For example, for the mutagen class, the explanation correctly identifies the importance of the N02 group; however, the generated graph isn’t realistic and might not even be chemically possible. Furthermore, the non-mutagen example doesn’t display any clear patterns or identifiable structures.
- The explanations for the Cyclicity dataset as well as the Wheel class do not appear to be members of the underlying data distribution.
- GNNInterpreter provides a way of generating example graph that would be classified as a target class by a GNN model, without needing to specify domain specific rules. On the other hand, optimizing predictions and even embeddings does not appear to be a sufficient objective for producing in-distribution graphs. The author correctly point out that optimizing predictions can lead to unrealistic graphs, but they have also inadvertently demonstrated that optimizing embeddings is not necessarily sufficient either.

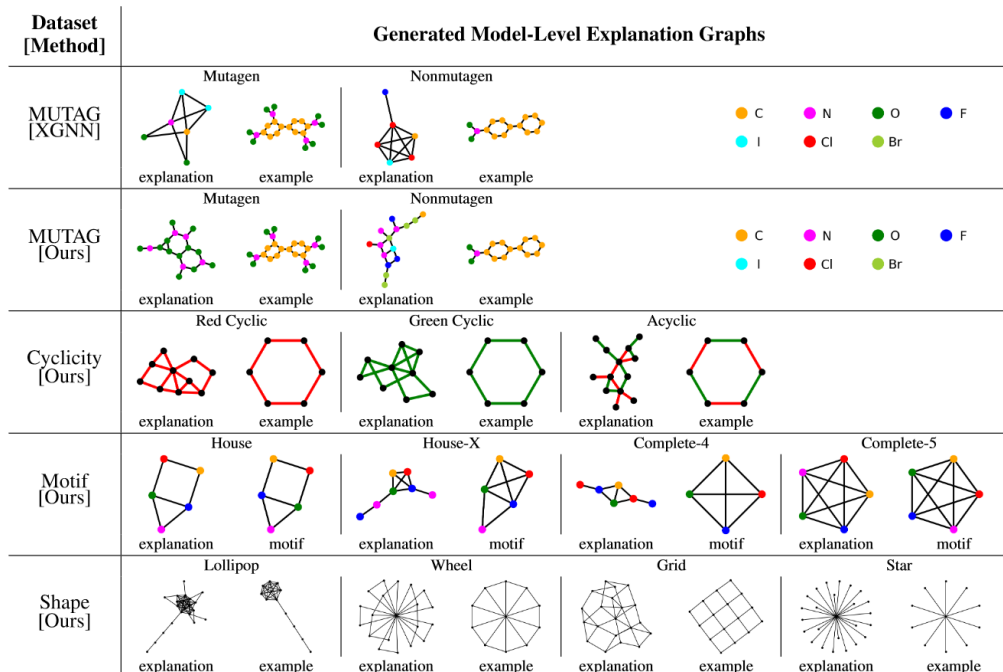


Figure 1: The qualitative results for 4 datasets. For each class in all datasets, the explanation graph with the class probability of 1 predicted by the GNNs is displayed on the left; as a reference, the example graph selected from the training data of the GNNs or the motif is displayed on the right. The different colors in the nodes and edges represent different values in the node feature and edge feature.

D4Explainer

(Chen et al. 2023)

- Overview
 - D4Explainer or in-Distribution GNN explanations via Discrete Denoising Diffusion attempt to directly address the realism of generated graphs in model-level explanation by using the observed data to train a Diffusion model.
 - In the image domain, diffusion model have been shown to produce the most realistic images when compared to other generative AI methods such as Generative Adversarial Networks (GANs).
 - Diffusion model work by iteratively noising an observation until it is pure noise. Then a denoising model is trained to predict the noise added at any given time step. Then new observations can be generated by passing pure noise through the diffusion model in a process known as reverse sampling.
 - Additional label information can be passed to generate observations with similar a label.
- Forward Diffusion
 - The authors here are focused on discrete structural diffusion. D4Explainer generate example graph by noising and de-nosing the adjacency matrices of observed graphs. The sampled graphs have the same features, but different structures.
 - The process of gradually adding noise to the input data is called forward diffusion.

During forward diffusion, random noise is added iteratively until the data becomes pure noise in the final iteration. This ensures that the denoising model can start with pure noise. Forward diffusion is usually a Markov process.

- If we assume that the observed graphs are Gilbert random graphs, like GNNInterpreter, pure noise would mean that $\forall (i, j) A[i, j] \sim \text{Bernoulli}(0.5)$.
- Let $t \in [0, T]$ denote the current iteration. Let β_t be the common probability that any edge changes state at time step t . $(\beta_1, \dots, \beta_T)$ is known as the variance schedule and is a set hyperparameter. Let A_t be a one-hot encoded version of the t^{th} noised adjacency, $[1, 0]$ is the edge exists and $[0, 1]$ otherwise. Then the forward diffusion process can be expressed as:

$$A_t[i, j] \sim q(A_t[i, j] | A_{t-1}[i, j]) = \text{Cat}(A_{t-1}[i, j] \cdot Q_t) \quad (10)$$

or

$$A_t[i, j] \sim q(A_t[i, j] | A_0[i, j]) = \text{Cat}\left(A_0[i, j] \prod_{i=1}^t Q_i\right) \quad (11)$$

where,

$$Q_t = \begin{bmatrix} 1 - \beta_t & \beta_t \\ \beta_t & 1 - \beta_t \end{bmatrix}$$

the t^{th} element-wise transition matrix.

- **Proof 2:** $\lim_{t \rightarrow \infty} q(A_t[i, j] | A_{t-1}[i, j]) \xrightarrow{D} \text{Bernoulli}(0.5)$ (converges in distribution).

$$\begin{aligned} \lim_{t \rightarrow \infty} q(A_t[i, j] | A_{t-1}[i, j]) &= \lim_{t \rightarrow \infty} \text{Cat}\left(A_0[i, j] \prod_{i=1}^t Q_i\right) \\ &= \lim_{t \rightarrow \infty} \text{Cat}\left(A_0[i, j] \prod_{i=1}^t \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 - 2\beta_i \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}\right) \\ &\text{Eigen decomposition.} \\ &= \lim_{t \rightarrow \infty} \text{Cat}\left(A_0[i, j] \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \prod_{i=1}^t 1 - 2\beta_i \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}\right) \\ &= \text{Cat}\left(A_0[i, j] \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 0.5 \end{bmatrix}\right) \\ &\text{Set } \beta < 0.5. \\ &= \text{Cat}([0.5, 0.5]) = \text{Bernoulli}(0.5) \end{aligned}$$

- Backward Diffusion (Denoising Model)

- A denoising model either predicts the error that was added or the value of the original observation. Even though only the adjacency matrix is being noised, we still want to include all available data. Thus the denoising model is parameterized as:

$$p(A_0 | A_t, t, X_0, E_0; \Omega) \quad (12)$$

where Ω is the set of trainable parameters.

- The author employed a Provably Power Graph Network (PPGN) [cite] as their denoising model; however, they have added an additional neural network to learn the time or noise level effect.
- Loss Dist
 - Like most diffusion models, the primary objective is to minimize the distance between the predicted de-noised observation and the original. The author have added an additional weight term to focus the model on noisier or more difficult training examples.

$$\mathcal{L}_{dist}(\Omega \mid A_0) = \sum_{t=1}^T \left(1 - 2\bar{\beta}_t + \frac{1}{T}\right) \mathbb{E}_{\hat{A}_0} \text{CrtEnt}(A_0, \hat{A}_0) \quad (13)$$

where $\hat{A}_0 = p(A_0 \mid A_t, t, X_0, E_0; \hat{\Omega})$, $A_t[i, j] \sim q(A_t[i, j] \mid A_0[i, j])$, and

$$\bar{\beta}_t = \frac{1}{2} - \frac{1}{2} \prod_{i=1}^t (1 - 2\beta_i)$$

the cumulative transition probability.

- Model-level sampling algorithm
 - Use a set of observed graph features, D4Explainer generates model level explanations by sequentially denoising pure noise to generate an adjacency structure that has a high probability of being a member of the targeted class. At each step A_t is denoised to k candidates A_0, k . The candidate with the best predicted probability is then noised to a level of $t - 1$ and the process is repeated. The pseudocode is reproduced below.

Algorithm 1 D4Explainer Model-level Explanation Reverse Sampling Algorithm

Require: $\hat{\Omega}$: trained denoising parameters; $q(A_t \mid A_0)$: forward diffusion process.

Input: N: maximum number of nodes; T: maximum noise level; K: number of candidates per iteration; $\tilde{\rho}$ targeted prediction vector; (X, E) : node and edge features.

Output: \hat{A} : adjacency matrix for model-level explanation.

Sample $A_T[1 : n, 1 : n] \sim \text{Bernoulli}(0.5)$

for t in T to 1 **do**

 Sample candidates $\{\hat{A}_{0,k} \sim p(A_t, t, X, E; \hat{\Omega}) : k \in 1, \dots, K\}$

 Select the best candidate $\underset{j \in 1, \dots, K}{\text{argmin}} \text{CrtEnt}(\text{explaine}(G = (X, A_{0,j}, E)), \tilde{\rho})$

 Sample $A_{t-1}[1 : n, 1 : n] \sim q(A_{t-1}, A_{0,j})$

end for

return A_0

- In-Distribution metrics.
 - To access how well a sampled explanation matches the observed graph distribution, the author compute various maximum mean discrepancy (mmd) statistics.
 - MMD is a general technique to compute the distance between two distributions of data using only observed data. The distributions are estimated using a kernel and then the distance between their means are compared.

- A common way to approximate a density is to use the average of Gaussian distributions centered at each observation:

$$\begin{aligned} f(x)_X &\approx \frac{1}{n} \frac{1}{\sqrt{2\pi}\sigma} \sum_{i=1}^n e^{-\frac{(x-x_i)^2}{2\sigma^2}} \\ &\approx \frac{1}{n} \frac{1}{\sqrt{2\pi}\sigma} \sum_{i=1}^n \phi(x)^T \phi(x_i) \end{aligned}$$

where, $k(x_i, x_j) = \phi^T(x_i) \phi(x_j)$ is the Gaussian kernel function. As long as the chosen kernel is *characteristic* the kernel mean embedding

$$\frac{1}{n} \sum_{i=1}^n \phi(x_i)^T = \frac{1}{n} \phi(\vec{x}) \mathbb{1}$$

is representative of the distribution. Here \vec{x} is the vector of observations and $\mathbb{1}$ is the appropriate dimensional vector of 1s.

- MMD statistics are then simple the l2 distance between the kernel mean embeddings:

$$\begin{aligned} MMD(\vec{x}, \vec{y}) &= \left\| \frac{1}{n} \phi(\vec{x}) \mathbb{1} - \frac{1}{m} \phi(\vec{y}) \mathbb{1} \right\|_2^2 \\ &= \left(\frac{1}{n} \phi(\vec{x}) \mathbb{1} - \frac{1}{m} \phi(\vec{y}) \mathbb{1} \right)^T \left(\frac{1}{n} \phi(\vec{x}) \mathbb{1} - \frac{1}{m} \phi(\vec{y}) \mathbb{1} \right) \\ &= \frac{1}{n^2} \phi(\vec{x})^T \phi(\vec{x}) + \frac{1}{m^2} \phi(\vec{y})^T \phi(\vec{y}) - \frac{2}{nm} \phi(\vec{x})^T \phi(\vec{y}) \end{aligned}$$

- Intuitively $\phi(\vec{x})^T \phi(\vec{x})$ and $\phi(\vec{y})^T \phi(\vec{y})$ can be thought of as the distance within the observations of x and y and $\phi(\vec{x})^T \phi(\vec{y})$ as the distance between. Therefore an MMD statistic close to zero means that the distribution are approximately the same.
- The author use MMD statistics to compare the degree, clustering, and spectrum distributions of the generated explanation against the observed data. The degree distribution of a graph indicates how frequently nodes have different numbers of connections, reflecting the overall connectivity pattern. The clustering coefficient of a node measures the proportion of the node’s neighbors that are also connected to each other, representing local clustering. The spectrum distribution, which is the distribution of eigenvalues of the adjacency matrix or Laplacian matrix, provides insights into the graph’s structural characteristics and dynamic properties.
- Like GNNInterpreter, the author of D4Explainer also value the sparsity of their generate example. They measure the density of a graph as the number of present edges divided by the number of possible edges or

$$\text{Density} = |\mathcal{E}|/|\nu|^2 \quad (14)$$

- Loss CF

- In addition to model-level explanations, D4Explainer can provide *counterfactual explanation* be include an additional term in the loss function. The authors define a counterfactual explanation for a particular observed graph to be G^c such that $\hat{Y}_{G^c} \neq \hat{Y}_G$, but the difference between G^c and G is minimal.

- \mathcal{L}_{dist} ensures that the G^c generated won't deviate too far from the observed graph distribution.
- Adding \mathcal{L}_{CF} minimize the probability that the generated graph is classified the same as the input graph.

$$\mathcal{L}_{CF}(\Omega \mid A_0) = \mathbb{E}_{\hat{A}_0} - \log(1 - \rho_{G^c}[\hat{Y}_G]) \quad (15)$$

where $G = (X, A, E)$ is an observed graph and $G^c = (X, \hat{A}_0, E)$.

- To quantify the quality of generated counterfactual explanations, the author report *Counterfactual Accuracy*, *Fidelity*, and modification ratio.

- * CF-ACC measure the proportion of G^c s that have different predicted labels than there associated observed graph.

$$\text{CF-ACC} = \mathbb{E}_{A_t} I(\hat{Y}_{G^c} \neq \hat{Y}_G) \quad (16)$$

- * Fidelity measures the difference in the predicted probability of G and G^c with respected to the original class label.

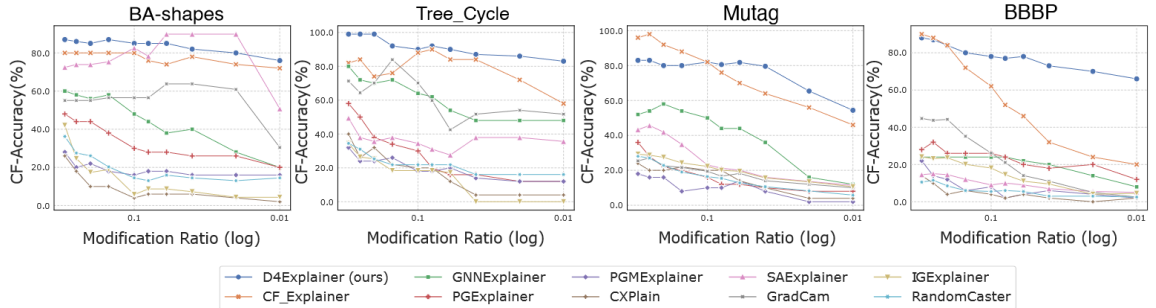
$$\text{Fidelity} = \mathbb{E}_{\hat{A}_0} \rho_G[\hat{Y}_G] - \rho_{G^c}[\hat{Y}_G] \quad (17)$$

- * Modification ratio measures difference in edges between G^c and G relative to the size of the original graph.

$$\text{MR} = \frac{|\sum_{i,j} A[i,j] - \hat{A}_0(i,j)|}{\sum_{i,j} A[i,j]} \quad (18)$$

• Summary of Results + Figures

- Similar to GNNInterpreter, the author report the mean predicted probability for the targeted class. Using a maximum of 6 nodes, the author were able to achieve a mean of 0.832 on the MUTAG dataset.
- Since the counterfactual explanation are generated on a per observation basis, the author's compared their method to other popular instance-level explanation methods.



The above plot displays that D4Explainer can flip the prediction on ~80% of observed graphs at most modification levels. Missing from the plot is a variance measure. It is possible that the counterfactual accuracy varies more between different data splits or hyperparameter than the other methods.

Models	Mutag				BBBP				NCII			
	Deg.	Clus.	Spec.	Sum.	Deg.	Clus.	Spec.	Sum.	Deg.	Clus.	Spec.	Sum.
RandomCaster	0.1593	0.0247	0.0417	0.2257	0.1693	0.0072	0.0397	0.2162	0.1847	1.9769	0.0404	2.2020
GNNExplainer	0.1614	<u>0.0002</u>	0.0409	0.2025	0.1615	0.0002	0.0395	0.2012	0.1577	0.0005	0.0405	0.1987
SAExplainer	0.0940	0.0032	0.0412	0.1384	0.1594	0.0032	0.0402	0.2028	0.189	0.0002	0.0408	0.2300
GradCam	<u>0.1122</u>	0.0083	0.0416	0.1621	<u>0.0699</u>	0.0026	<u>0.0384</u>	<u>0.1109</u>	0.1638	0.0003	0.0404	0.2045
IGExplainer	0.1292	0.0000	0.0411	0.1703	<u>0.0908</u>	0.0000	0.0394	<u>0.1302</u>	0.4288	0.0002	0.0398	0.4688
PGExplainer	0.1475	<u>0.0002</u>	0.0418	0.1895	0.2014	0.0018	0.0403	0.2435	0.1937	0.0000	<u>0.0396</u>	0.2333
PGMEExplainer	0.1800	<u>0.0002</u>	0.0419	0.2221	0.1916	0.0003	0.0403	0.2322	0.2199	0.0000	<u>0.0404</u>	0.2603
CXPlain	0.1734	1.2706	0.0417	1.4857	0.1768	<u>0.0001</u>	0.0394	0.2163	0.1629	<u>0.0001</u>	0.0404	0.2034
CF-GNNExplainer	0.1172	0.0000	<u>0.0380</u>	0.1552	0.0870	<u>0.0001</u>	0.0393	0.1264	<u>0.1224</u>	<u>0.0001</u>	0.0404	<u>0.1629</u>
D4Explainer	0.1172	0.0000	0.0244	<u>0.1416</u>	0.0530	0.0000	0.0331	0.0861	0.1006	0.0000	0.0353	0.1359

The above table indicates that the generated graph tend to have feature distribution that are close to the original dataset; however, there might be a hidden class bias. The distributions might only be close for the majority class, skewing the averages.

ProtGNN

(Zhang et al. 2021)

- Overview:
 - Up until this point we have been describing post-hoc explanation methods.
 - Rudin (2019) raises several important concerns with this paradigm.
 1. Parity in the outcomes does not imply parity in the reasoning. Explanation may bear strong prediction, but for distinct and sometimes nonsensical reasons. Recall that that completely random graphs can illicit confident predictions.
 2. Feature importance does not indicate how the model is using the feature, just that it is important. For example, a substructure might lead to a prediction because of a strong correlation with the class, but the substructure could also be important because of a strong negative correlation with the other classes.
 - ProtGNN use the prototype learning paradigm to create a GNN model that is *interpretable*.
- Prototype Learning:
 - Prototype learning models make predictions by comparing new inputs to exemplar cases learned during training.
 - This results in an interpretable model because as long as the prediction function applied to the prototype scores is simple, the reasoning for any given prediction will be clear.
 - Intuitively, this can be thought of as using a black-box model to engineer features, which are then passed to a simple white-box model.
- ProtGNN Architecture:
 - ProtGNN initialize m prototype vectors for each of the c classes. Let p_k denote a prototype vector and P_{Y_G} denote the set of prototype vectors assigned to prediction class Y_G .
 - Let $f(G; \Omega_f) = \hat{p}_G \in \mathbb{R}^p$ be a graph encoder function that maps any graph into the prototype vector space, where Ω_f is the set of trainable parameters.

- Prediction Loss:

$$\mathcal{L}_{\text{prot}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mc} \mid G) = \mathbb{E}_G \text{CrsEnt} [\psi(\text{sim}(\hat{p}_G, p_1), \dots, \text{sim}(\hat{p}_G, p_{mc}); \Omega_\psi), Y_G] \quad (19)$$

where ψ is any simple prediction model with trainable parameters Ω_ψ and sim is any vector similarity function. ProtGNN uses a full connected layer with a softmax output activation for ψ , which is similar to a multinomial regression model.

- To address similar out-of-distribution concerns discussed above, the author include 3 additional loss terms to constraints.
- Cluster Loss: Each embedding should be close to at least one prototype assigned to its ground truth class.

$$\mathcal{L}_{\text{clst}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mc} \mid G) = \mathbb{E}_G \min_{p_k \in P_{Y_G}} \|\hat{p}_G - p_k\|_2^2 \quad (20)$$

- Separation Loss: Embeddings should be far away from the prototypes of other classes.

$$\mathcal{L}_{\text{sep}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mc} \mid G) = -\mathbb{E}_G \min_{p_k \notin P_{Y_G}} \|\hat{p}_G - p_k\|_2^2 \quad (21)$$

- Diversity Loss: The prototypes within a class should learn different information.

$$\mathcal{L}_{\text{div}}(\Omega_f, \Omega_\psi, p_1, \dots, p_{mc} \mid G) = \sum_{k=1}^C \sum_{p_i \neq p_j \in P_k} \max(0, \cos(p_i, p_j) - s_{\text{max}}) \quad (22)$$

where s_{max} is a similarity threshold.

- Prototype Projection Using MCTS:

- The learned prototypes are latent embedding vectors which are not directly interpretable. To address this issue, the authors employ a Monte Carlo Tree Search (MCTS) algorithm to find the subgraph within the observed graphs of the prototype’s class that has an embedding closest to a given prototype.
- Specifically the author using MCTS to select an optimal sequence of pruning actions.
- Let $\mathcal{G}_{0, i, \dots, j}$ represent the subgraph resulting from the sequence of pruning actions $\text{act}_i, \dots, \text{act}_j$, where \mathcal{G}_0 , the root of the search tree, is the observed graph.
- The MCTS algorithm works in 4 steps:
 1. An existing node in the search tree $\mathcal{G}_{0, i, \dots, j}$ is *selected*.
 2. $\mathcal{G}_{0, i, \dots, j}$ is *expanded* adding a child node $\mathcal{G}_{0, i, \dots, j, k}$ resulting from action act_k . To limit the search space, the author restrict act_k to be the removal a peripheral node with minimum degree.
 3. Then the reward for the sequence of actions i, \dots, j, k is estimated via a Monte Carlo *simulation* where random actions are taken until a subgraph of the desired size is produced. The subgraph embeddings are then compared to the prototype vector and the similarities are averaged.

4. The child nodes estimated score is then *backpropagated* to update the estimated scores of all its parent nodes.
- After a fixed set of iterations, the decision path the highest resulting score is selected to produce the prototype projection. This process is repeated across all observed graphs in the prototype’s class.
- Conditional Subgraph Sampling:
 - In addition to finding the closest subgraph in the observed class, the authors also suggest a method for efficient finding similar subgraph in each input.
 - This is done by training a neural network to predict an edge mask, conditional on a prototype vector. The resulting induced subgraph’s embeddings are optimized to be also close as possible to the given prototype, thus producing a similar observed subgraph per learned prototype.

$$\hat{A}[i, j] = \text{nn}(G, p_k; \Omega_{\text{nn}}) \quad (23)$$

$$\mathcal{L}(\Omega_{\text{nn}}, \Omega_f, p_k \mid G) = -\mathbb{E}_G \text{sim}(\hat{p}_G, p_k) \quad (24)$$

- The authors denote the version of the model that utilizes this module, in addition to the above losses, as *ProtGNN+*.
- Summary of Results + Figures
 - ProtGNN achieves similar or better graph classification accuracy than state of the art models across 5 standard benchmark datasets.

Datasets	GCN			GIN			GAT		
	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+	Original	ProtGNN	ProtGNN+
MUTAG	73.3±5.8	76.7±6.4	73.3±2.9	93.3±2.9	90.7±3.2	91.7±2.9	75.0±5.0	78.3±4.2	81.7±2.9
BBBP	84.6±3.4	89.4±4.1	88.0±4.6	86.2±1.1	86.5±1.6	85.9±4.0	83.0±2.6	85.9±2.5	85.5±0.8
Graph-SST2	89.7±0.5	89.9±2.4	89.0±3.0	92.2±0.3	92.0±0.2	92.3±0.4	88.1±0.8	89.1±1.2	88.7±0.9
Graph-Twitter	67.5±1.9	68.9±5.9	66.1±6.5	66.2±1.3	75.2±2.8	76.5±3.4	69.6±6.5	64.8±4.0	66.4±3.3
BA-Shape	91.9±1.7	95.7±1.4	94.3±3.7	92.9±0.5	95.2±1.3	95.5±2.4	92.9±1.2	93.4±3.4	93.2±2.0

- ProtGNN is also robust to the choice of embedding model as the author have shown benefits across various GNN architectures.
- The author not that the computational complexity of ProtGNN can be significantly higher than a standard GNN model taking roughly 5 times longer to train. Most of this complexity is attributable to the use of the MCTS.
- On the other hand ProtGNN produces much clearer results. Analogous to a regression model, the prototypes can be thought of as features, the similar subgraphs and similarity scores as the observed feature values, and the class connections as the marginal effects on the prediction.

Input Graph	Similar Subgraphs	Prototypes	Similarity Scores	Class Connection	Contributed Points	Similar Subgraphs	Prototypes	Similarity Scores	Class Connection	Contributed Points
			4.281	\times 1.334	= 5.711			2.505	\times 1.303	= 3.264
			3.572	\times 1.327	= 4.740			1.491	\times 1.225	= 1.826
			3.414	\times 1.240	= 4.233			1.463	\times 1.090	= 1.595
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
Total points to mutagenic: 23.270						Total points to non-mutagenic: 9.575				

- Since prototypes from every class are used for the classification, ProtGNN can also be used for counterfactual type analysis. For any given predictions, researcher can determine why a certain classification was made and why the other classes were not selected.

Synthesis of Core Papers

- GNNInterpreter and D4Explainer are both *post-hoc generative explanation method*.
- Each defines a model-level explanation for a given class c as an estimated graph generating distribution $G_{\text{Gen}, c} \sim f(G \mid \Theta)$ such that $\underset{\Theta}{\operatorname{argmin}} \mathbb{E}_{G_{\text{Gen}}} \text{CrsEnt}(\text{explainee}(G_{\text{Gen}}, \tilde{\rho}_c))$. In other words, the distribution that maximize the probability that a sampled graph is classified by the explainee as the desired class.
- The key difference between the methods is the form of $f(G \mid \Theta)$. This is akin to the parametric (distributional) assumption.
- GNNInterpreter assumes that each graph matrix has a separate distribution with possibly dependent parameters. On the other hand, D4Explainer assumes that the only random variation between graphs is in the adjacency structure.
- Both models assume that every entry of the adjacency matrix follows a Bernoulli distribution, but diverge in terms of estimating the probability. D4Explainer makes use of the observed graphs directly, whereas GNNInterpreter only injects latent embeddings. This makes GNNInterpreter much more sensitive to the underlying explainee model, which could be both a good and bad property.
- Both sets of authors highlight the potential *out-of-distribution* problem that plagues generative explanations. GNN models are not robust to out-of-distribution or unrealistic graph, which is to say that the model can still return a confident classification for a complete random graph.
- D4Explainer partial resolves this issue by restricting generated node and edge features to the observed data. This ensure that the feature values of any generated graph will not be out-of-distribution; however, this limits inference.
- GNNInterpreter attempts to resolve this issue by imposing similarity to the average class embedding; however, embeddings can also suffer from out-of-distribution problems. The

same way an unrealistic or random graph can have similarity predictions, it could also have similar embeddings because of the information loss in the lower dimensional space.

- While GNNInterpreter and D4Explainer place differing restrictions on their respective likelihoods, they should be interchangeable. For example, there is no reason why $\mathcal{L}_{\text{spars}}$ couldn't be included in the loss for D4Explainer.
- The MMD statistic methods could be applied to graph generated by GNNInterpreter.
- ProtGNN in contrast to the first two methods, is a *self interpretable GNN model*. ProtGNN isn't attempting to decipher a predictive model, it is a predictive model.
- The prototype similarity is closely related to GNNInterpreter's $\mathcal{L}_{\text{embed}}$ as they both measure the similarity between graph in a latent vector space. Unlike GNNInterpreter, the embedding function in ProtGNN is trainable. Furthermore, ProtGNN also uses dissimilarity to the non-target classes, but this could be easily added to GNNInterpreter's loss.
- On the other hand, similar to D4Explainer, the graph created from the prototype is restricted to the observed data.
- The similar subgraphs are instance level; however, the class connection weights and prototype projections are model-level.
- D4Explainer produces counterfactual examples by directly compute possible input changes that flip the prediction, while ProtGNN demonstrates how the prediction could have changed by display the similarity to structures indicative of other classes. While not the traditional directly counterfactual analysis, ProtGNN provides model-level insight.

Technical Details

Methodology

- Since GNNInterpreter only makes qualitative assessments of generated structures, we applied the metrics describe in D4Explainer to quantify the properties of the generating distribution.
- We reimplemented GNNInterpreter using an identical generation scheme and loss function.
- We trained the reimplemented model on the MUTAG dataset which consists of nitroaromatic compounds, with the goal being to predict their mutagenicity on Salmonella typhimurium [cite]. Each observation is a graphs representing a chemical structure, where vertices denote atoms and edges represent bonds [cite]. MUTAG is a standard benchmark graph classification dataset.
- For our explaine model, we also implemented a 3-layer GCN model. There are slight variations between the architectures and training hyperparameter, but the accuracies are comparable, 86% (ours) and 92% (theirs).
- We additionally reimplement D4Explainer's counterfactual generator. Although not discussed in the original paper, this model can also be used to produce model level explanations. We simply randomly sample a graph from the opposite class and take the generated counterfactual graph as our explanation.

Results

Table 1

	Predictions	Density	Deg.	Clus.	Spec.
GNNInterpreter Original					
Class 0	1.0 +/- 0.0	NA	NA	NA	NA
Class 1	1.0 +/- 0.0	NA	NA	NA	NA
GNNInterpreter Reimplemented					
Class 0	0.93 +/- 0.01	0.38 +/- 0.002	1.77	1.53	0.07
Class 1	0.98 +/- 0.01	0.32 +/- 0.003	1.41	1.43	0.04
D4Explainer Original					
Aggregated	0.92	0.315	0.12	0.00	0.02
D4Explainer Reimplemented					
Class 0	0.63 +/- 0.23	0.13 +/- 0.03	0.27	0.57	0.05
Class 1	0.66 +/- 0.20	0.13 +/- 0.03	0.08	0.25	0.04

¹ +/- 1 standard deviation; 1000 graphs.

* Empty

- We were unable to achieve 0 variance predictive accuracy, but again the values are comparable (see table ?) given the lack of extensive hyperparameter tuning. The prediction are similarity superior to that of D4Explainer (see table ?).
- As noted by the authors, the explaine model has a bias towards the mutagenic class (class 0), which can explain the greater deviation between the class 0 models.
- Another limiting factor is a lower set maximum number of nodes. We set it to 8 to make the results more comparable to the model-level explanation reported by D4Explainer.
- Surprisingly D4Explainer produces sparser graphs, in terms of density, than GNNInterpreter, despite no explicit penalty in the loss function.
- Based on the MMD statistics, the D4Explainer produces graph with better in-distribution properties. This might be the result of the explicit regularization in GNNInterpreter. Penalize the edge probabilities should result in a different degree distribution. The clustering coefficient might also be effect since fewer edges result in fewer triangle clusters.
- The spectrum distributions are close for both model, since both model enforce similarity in the explaine embedding space which is similar to the nature of the spectrum distribution.
- On the other hand, an argument could be made that a similar degree distribution and clustering coefficient isn't actually desirable. For larger or more densely connected graphs, lower MMD statistics might results in explanations that are too complex to be useful.
- Similar to the original examples display in GNNInterpreter, the generated non-mutagenic graphs do not have clear or realistic structures. Graph (f) highlights that nitrogen in key. We believe that this result is consistent with the original findings about the N02 groups, but because of the more restrictive node limit, our reproduction focuses on nitrogen since multiple oxygens can be found in non-mutagenic chemicals.

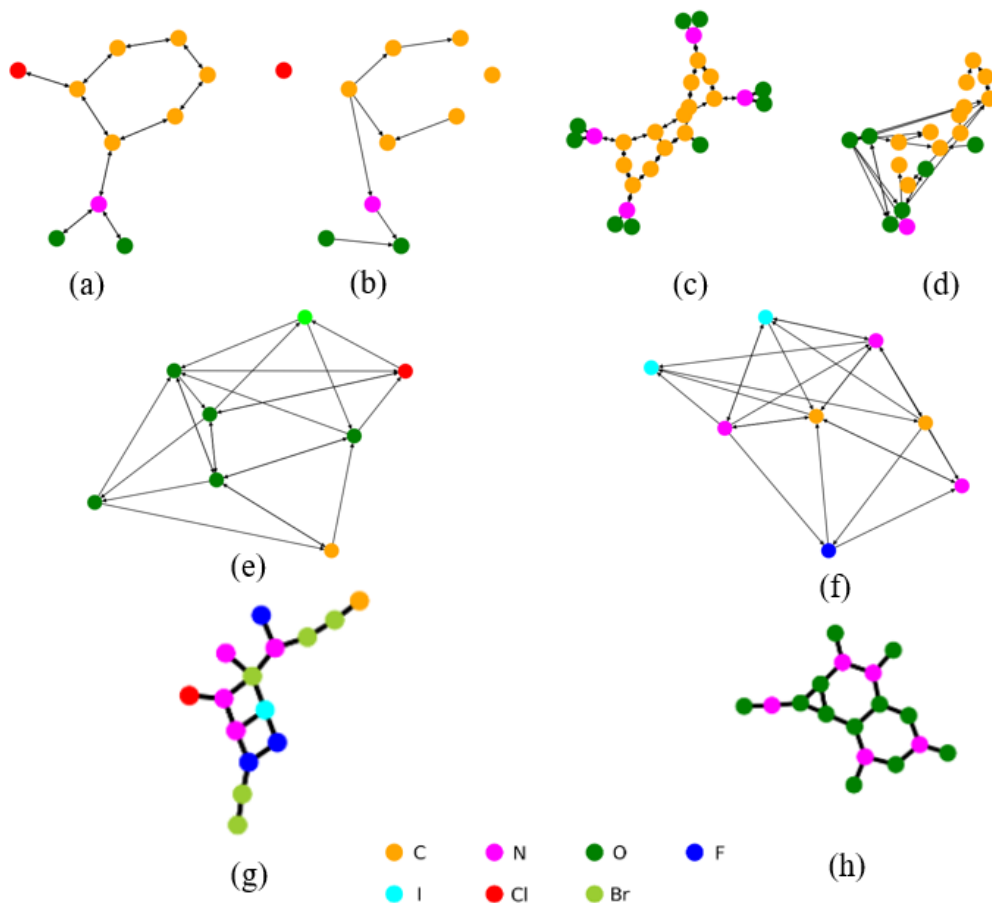


Figure 1: (a) Observed graph from class 0, $\text{Pr} = 0.92$; (b) Generated counterfactual explanation of (a) from D4Explainer reimplemented, $\text{Pr} = 0.67$; (c) Observed graph from class 1, $\text{Pr} = 0.96$; (d) Generated counterfactual explanation of (b) from D4Explainer reimplemented, $\text{Pr} = 0.96$; (e) Generated explanation for class 0 from GNNInterpreter reimplemented, $\text{Pr} = 0.94$; (f) Generated explanation for class 1 from GNNInterpreter reimplemented, $\text{Pr} = 1.0$; (g) Generated explanation for class 0 from GNNInterpreter (Wang and Shen 2024, fig. 1, pg.7), $\text{Pr} = 1.0$; (h) Generated explanation for class 1 from GNNInterpreter (Wang and Shen 2024, fig. 1, pg.7), $\text{Pr} = 1.0$.

- The reimplement D4Explainer has a CF-ACC in line with the original paper, 76.32% at 0.41 log MR.
- The reimplement D4Explainer produces the least dense graphs, but has the least discriminative predictions. The latter makes sense, since the goal of the counterfactual loss is to flip the prediction, not necessarily maximize it.
- One the other hand, this model demonstrates that it is possible to generate sparse predictions and maintain a similar degree distribution and clustering.
- Visually the counterfactual explanation make sense. Graph (b) eliminated the red or chlorine atom which is typically associated with non-mutagenic compounds [XGNN]. Graph (d) corrupted 3 of the 4 N02 groups, which is in line with GNNInterpreter’s findings that multiple N02 groups is a strong indicator of mutagenicity.
- This results lead to two main conclusions.
 - First, the generator model should make use of the graph level observations. GNNInterpreter only use information about the graph structure through the explaine, never on the raw scale.
 - * This lack of graph level information result can result in unrealistic distributional properties.
 - * This could be done by using the observed graphs to predict the distributional parameters.
 - * Alternatively, the generator could be replaced with a diffusion model without any need to modify the loss.
 - Second, the generator model should use class or prediction information.
 - * D4Explainer’s model-level sampling algorithm is less efficient and less accuracy than GNNInterpreter’s direct optimization during training.
 - * The reimplemented sampling algorithm demonstrates that direct denoising can still maintain low MMD statistics.
 - * A possible modification would be to add the $\mathcal{L}_{\text{pred}}$ and $\mathcal{L}_{\text{embed}}$ from GNNInterpreter to the loss.
 - * Alternatively, one could embedding the class label, similar to the way time is embedded, in order to establish a conditional diffusion model, which has worked extremely well in other applications.
 - * Additionally, it would be easy to explore diffusing the node and edge features as well as like *DiGress* [cite].

Future Directions

Graph Edit Distance

- GNNInterpreter generates graphs with predictions very close to the target class, but lack good in-distributions properties.

- D4Explainer generates explanations with more similar features distributions to the observed data; however, the predicted probabilities show more variance. Furthermore, while the model parameters of GNNInterpreter’s generator can be interpreted, D4Explainer introduces another black-box model.
- We propose using a differentiable approximation to the graph edit distance as a way of generically enforcing structural consistency.
- GED measures the similarity between two graphs by counting the minimum number of edits required to make the graphs isomorphic (or subgraph isomorphic). Similar to the modification ratio for counterfactual explanations.
- This has the key advantage of being human interpretable. For example, the percentage of atoms and bonds that need to be changed for two molecules to match is more relatable than the cosine distance between their embeddings.
- Finding the exact optimal GED is a known NP-complete problem, and most solutions are not differentiable (Bougleux et al. 2015).
- Fortunately, approximating GED as a quadratic assignment problem (QAP) solves both the time complexity and derivative issue (R. Wang et al. 2024).

Let G_1 and G_2 be graphs we wish to compute the GED between. QAP approximates this by solving the following optimization problem:

$$\mathcal{L}_{\text{GED}} = \max_M \text{vec}(M)^T K(G_1, G_2) \text{vec}(M) \quad (25)$$

here M is a $(n_1 \times n_2)$ binary node matching matrix and K is a $(n_1 n_2 \times n_1 n_2)$ quadratic affinity matrix that encode both node and edge similarity information. The efficiency of QAP comes from the fact that when any two nodes are matched together, the above objective will account for the possible edges that could connect the nodes. In order to compute GED, each entry of K should range from $[-1, 0]$, where -1 signals the need for a complete edit and 0 means no edits are required (R. Wang et al. 2024). We use the following similarity metric for every pair of nodes and edges:

$$\begin{aligned} & -0.25 [1 - \phi(\text{Continuous Features})] + \\ & -0.5 [1 - \phi(\text{Discrete Features})] \end{aligned} \quad (26)$$

where $\phi(\cdot)$ is the normalized cosine distance. $1 - \phi(\cdot)$ will return 2 if the vectors are in opposite directions, 1 if the vectors are orthogonal, and 0 if they are pointing in the same direction; however, one-hot encoded vectors will never be in opposite directions, so the different types of feature must be weighted correctly. Computing affinity this way allows for K to be differentiable w.r.t. generated graph features. Furthermore we use the reweighted random walks algorithm (Cho, Lee, and Lee 2010) to solve the above objective, which generates an M that is differentiable w.r.t. to K .

- We conducted a simulation study to compare graph level and latent similarity, or \mathcal{L}_{GED} and $\mathcal{L}_{\text{embed}}$.
- We generated 4 clusters of graph data. Every graph had one continuous edge feature sampled from a standard normal, one discrete node and edge feature sampled from a Bernoulli, and an adjacency matrix also Bernoulli. The only variation between the clusters was the maximum number of nodes as well as the mean of the continuous node feature distribution. Cluster 0 had 75-100 nodes, cluster 1 had 50-75 nodes, and cluster 2 as well as 3 had 10-35 nodes. This was

designed to mimic a classic binary graph classification dataset where cluster 0 and 1 are the observed classes and clusters 2 and 3 are generated subgraph explanations. The embedding model was a 3 layer graph convolution network with a neural network type convolution that utilized every feature.

- During each of the 50 trials, the GCN was trained using 50 samples (each) from only clusters 0 and 1. We then sampled 50 test graphs from each of the 4 cluster and computed 95% t intervals for GED and embedding cosine distance as percentages of their respective upper bounds. Since ($\mu_0 = -1, \mu_1 = 1, \mu_2 = 0, \mu_3 = -5$) we would expect an unbiased distance metric to maintain the following relationship,

$$\text{dist}(1, 2) \approx \text{dist}(0, 2) < \text{dist}(0, 1) < \text{dist}(0, 3) < \text{dist}(1, 3) \quad (27)$$

based on the distance between cluster means.

Table 2: Simulation Results

	Cluster 0	Cluster 1
Embeddings		
Within	(0.0008, 0.0015)	(0.0005, 0.0009)
Between	(0.5662, 0.6264)	(0.5689, 0.6289)
Cluster 2	(0.2894, 0.3589)	(0.1282, 0.209)
Cluster 3	(0.018, 0.0253)	(0.6521, 0.7097)
GED		
Within	(0.4147, 0.419)	(0.462, 0.4674)
Between	(0.6589, 0.6639)	(0.6586, 0.6646)
Cluster 2	(0.7843, 0.7947)	(0.7571, 0.7715)
Cluster 3	(0.6164, 0.6286)	(0.8256, 0.8352)

¹ Node Means: $\mu_0 = -1, \mu_1 = 1, \mu_{out1} = 0, \mu_{out2} = -5$.

² 95% t intervals on % difference.

Expected: $\text{dist}(1, 2) \approx \text{dist}(0, 2) < \text{dist}(0, 1) < \text{dist}(0, 3) < \text{dist}(1, 3)$

GED: $(0.75, 0.77) \approx (0.78, 0.79) > (0.65, 0.66) > (0.61, 0.62) < (0.82, 0.83)$

Embedding: $(0.12, 0.20) < (0.28, 0.35) < (0.56, 0.62) > > (0.018, 0.0253) < (0.65, 0.7)$

The most worrisome bias arises between the distances from cluster 0 to 1 and from cluster 0 to 3. Despite cluster 0's mean being twice as close to cluster 1, the embedding distance indicates that cluster 0 is (53, 60) percentage points farther away from cluster 1. Although GED also displayed this bias, it was considerably less pronounced, registering at (1, 5) percentage points. This implies that a generator initialized with a mean of -5 would produce samples with embeddings confidently assigning them to cluster 0, despite having a feature distribution far from the truth. Relying solely on embedding distance for training also increases the likelihood of the generator becoming

trapped in a misleading local optimum due to the substantial bias. Although GED exhibits a statistically significant bias, its magnitude is considerably smaller, making it less likely to disrupt training. Another troubling bias observed is that the embedding distance between clusters 1 and 2 is approximately 10 percentage points less than the embedding distance between clusters 0 and 2, even though the mean of cluster 2 is equidistant from the means of clusters 1 and 0. This could stem from cluster 1 having fewer nodes making samples more similar in terms of size to cluster 2. However, since researcher typically cap the size of examples for interpretability, an objective with a degree bias could be problematic. On the contrary, GED introduces a bias absent in the embedding space. The edit distance between clusters 0 and 2 is actually greater than the distance between clusters 0 and 1. While biased concerning mean locations, this direction is aligned with the maximum node size. Despite this, the differing directions of the biases of GED and embedding distance imply that they capture unique structural nuances, potentially complementing each other.

- \mathcal{L}_{GED} could also be added to the loss function for D4Explainer as well.

Graph Prototype Masking Network

- ProtGNN has two main limitations:
 1. Prototype projections are computationally expensive.
 2. Prototype projections cannot produce out-of-sample insights.
- These two issues can be resolved by using graph prototypes as opposed to vectors. Define each prototype to be a graph,

$$p_k = (X_{p_k}, A_{p_k}, E_{p_k}) \quad (28)$$

- $f(G; \Omega_f) = \hat{p}_G$ would need to be replaced with a model similar to the denoising model in D4Explainer, that can output graph matrices.
- The model can be trained using graph level similarity via GED and/or the original loss terms of ProtGNN applied to the prototypes embeddings.
- This eliminates the need for the upward prediction back to the graph space, which addresses issue 1.
- Since $X_{p_k}, A_{p_k}, E_{p_k}$ are free trainable parameters, they could contain structure that weren't entirely observed in the original dataset.
- We expect sample size to be a limiting factor. Although ProtGNN uses fairly high dimensional vectors for their prototypes, graph prototypes might still introduce exponentially more parameters. The size and sparsity of the graph prototypes could be enforced, but it would introduce additional hyperparameters. If the dataset is extremely large, then the additional parameters should be okay.

References

- Bai, Yunsheng, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. 2019. “Unsupervised Inductive Graph-Level Representation Learning via Graph-Graph Proximity.” <https://arxiv.org/abs/1904.01098>.
- Bouleux, Sébastien, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. 2015. “A Quadratic Assignment Formulation of the Graph Edit Distance,” no. arXiv:1512.07494 (December). <http://arxiv.org/abs/1512.07494>.
- Chen, Jialin, Shirley Wu, Abhijit Gupta, and Rex Ying. 2023. “D4Explainer: In-Distribution GNN Explanations via Discrete Denoising Diffusion,” no. arXiv:2310.19321 (October). <https://doi.org/10.48550/arXiv.2310.19321>.
- Cho, Minsu, Jungmin Lee, and Kyoung Mu Lee. 2010. “Reweighted Random Walks for Graph Matching.” In *Computer Vision – ECCV 2010*, edited by Kostas Daniilidis, Petros Maragos, and Nikos Paragios, 492–505. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-15555-0_36.
- Gallicchio, Claudio, and Alessio Micheli. 2019. “Fast and Deep Graph Neural Networks.” <https://arxiv.org/abs/1911.08941>.
- Huijben, Iris A. M., Wouter Kool, Max B. Paulus, and Ruud J. G. van Sloun. 2022. “A Review of the Gumbel-Max Trick and Its Extensions for Discrete Stochasticity in Machine Learning,” no. arXiv:2110.01515 (March). <https://doi.org/10.48550/arXiv.2110.01515>.
- Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. 2017. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables,” no. arXiv:1611.00712 (March). <https://doi.org/10.48550/arXiv.1611.00712>.
- Rudin, Cynthia. 2019. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead,” no. arXiv:1811.10154 (September). <https://doi.org/10.48550/arXiv.1811.10154>.
- Wang, Runzhong, Ziao Guo, Wenzheng Pan, Jiale Ma, Yikai Zhang, Nan Yang, Qi Liu, et al. 2024. “Pygmtools: A Python Graph Matching Toolkit.” *Journal of Machine Learning Research* 25 (33): 1–7. <https://jmlr.org/papers/v25/23-0572.html>.
- Wang, Xiaoqi, and Han-Wei Shen. 2024. “GNNInterpreter: A Probabilistic Generative Model-Level Explanation for Graph Neural Networks,” no. arXiv:2209.07924 (February). <https://doi.org/10.48550/arXiv.2209.07924>.
- Xiao, Guanghua, Shidan Wang, Ruichen Rong, Donghan Yang, Xinyi Zhang, Xiaowei Zhan, Justin Bishop, et al. 2023. *Deep Learning of Cell Spatial Organizations Identifies Clinically Relevant Insights in Tissue Images*. Preprint. In Review. <https://doi.org/10.21203/rs.3.rs-2928838/v1>.
- Yuan, Hao, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2022. “Explainability in Graph Neural Networks: A Taxonomic Survey,” no. arXiv:2012.15445 (July). <https://doi.org/10.48550/arXiv.2012.15445>.
- Zhang, Zaixi, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. 2021. “ProtGNN: Towards Self-Explaining Graph Neural Networks,” no. arXiv:2112.00911 (December). <https://doi.org/10.48550/arXiv.2112.00911>.