CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks

Lucic, A.; ter Hoeve, M.; Tolomei, G.; de Rijke, M.; Silvestri, F.

DOI
10.48550/arXiv.2102.03322

Publication date
2021

Document Version
Submitted manuscript

CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks

Ana Lucic [1] **Maartje ter Hoeve** [1] **Gabriele Tolomei** [2] **Maarten de Rijke** [1 3] **Fabrizio Silvestri** [2]

Abstract

Graph neural networks (GNNs) have shown increasing promise in real-world applications, which has caused an increased interest in understanding their predictions. However, existing methods for explaining predictions from GNNs do not provide an opportunity for recourse: given a prediction for a particular instance, we want to understand how the prediction can be changed. We propose CF-GNNEXPLAINER: the first method for generating counterfactual explanations for GNNs, i.e., the minimal perturbations to the input graph data such that the prediction changes. Using only edge deletions, we find that we are able to generate counterfactual examples for the majority of instances across three widely used datasets for GNN explanations, while removing less than 3 edges on average, with at least 94% accuracy. This indicates that CF-GNNEXPLAINER primarily removes edges that are crucial for the original predictions, resulting in minimal counterfactual examples.

1. Introduction

Advances in machine learning (ML) have led to breakthroughs in several areas of science and engineering, ranging from computer vision (CV) systems, to natural language processing (NLP) systems and conversational assistants. Parallel to the increase in performance of AI-based systems, there is a call for increasing the "understandability" of ML models (Goebel et al., 2018). Having the ability to understand *why* an ML model returns a certain output in response to a given input is important for a range of reasons: knowing why an object recognition application detects the wrong object in a picture might be useful for debugging the application, while explaining why a loan application is rejected is actually a legal requirement in many countries. Having certified methods for interpreting ML-based predictions will help enable their use across a variety of applications (Miller, 2017).

Explainable AI (*XAI*) refers to the set of techniques "*focused on exposing complex AI models to humans in a systematic and interpretable manner*" (Samek et al., 2019); a large body of work on XAI has emerged in recent years (Guidotti et al., 2018b). *Counterfactual (CF) explanations* are used to explain predictions of individual instances with an opportunity for recourse in the form: "If X had been different, Y would not have occurred" (Stepin et al., 2021). Counterfactual explanations are based on *counterfactual examples*: modifications of the input sample that change the output response (i.e., prediction).

As an example of the use case counterfactual explanations, consider an ML application for social media: suppose we want to predict whether a post shared on a social network contains misinformation or not. In a social network, a post is part of an "ecosystem" made up of users, content, groups, et cetera (Halevy et al., 2020). Fake news detection can be modeled as a node classification task, where the posts are the nodes in the graph and the relationships between the nodes are the edges. A counterfactual explanation for why a certain post was classified as "fake" could then show without which of the related posts the post had not been classified as such.

Graph Neural Networks represent the state-of-the-art in many tasks involving graph data, like the task in our motivating example (Wu et al., 2020). Existing methods for explaining the predictions of Graph Neural Networks (GNNs), have not investigated the problem of counterfactual explanations (Yuan et al., 2020b). In this paper, we address this gap and present CF-GNNEXPLAINER: the first method for generating counterfactual explanations for GNNs, which are defined as the minimal perturbations to the input (graph) data such that the prediction changes.

Similar to other CF methods proposed in the literature (Verma et al., 2020; Karimi et al., 2020), CF-GNNEXPLAINER works by perturbing input data at the instance-level. In particular, CF-GNNEXPLAINER iteratively removes edges from the original adjacency matrix based on matrix sparsification techniques. We keep track of the perturbations that lead to a change in prediction and we

[1]Informatics Institute, University of Amsterdam, Netherlands [2]Department of Computer Science, Sapienza University of Rome, Italy [3]Ahold Delhaize, Zaandam, Netherlands. Correspondence to: Ana Lucic <a.lucic@uva.nl>.

Under Review

return the perturbation involving the smallest change in the number of edges.

We evaluate CF-GNNEXPLAINER on three public datasets for GNN explanations and measure the effectiveness of our method using four metrics: coverage, explanation size, subgraph impact (i.e., explanation size relative to graph size), and accuracy of the generated counterfactual explanations. We find that CF-GNNEXPLAINER is able to generate counterfactual examples with at least 94% accuracy, while removing fewer than three edges on average.

Summarizing, we make the following contributions:

1. We formalize the problem of generating counterfactual explanations for GNNs;
2. We propose a novel method for generating counterfactual explanations for GNNs, CF-GNNEXPLAINER;
3. We propose the first experimental setup for evaluating counterfactual GNN explanations.

2. Background

In this section we provide background knowledge on Graph Neural Networks (Section 2.1) and Matrix Sparsification (Section 2.2), both of which are necessary for understanding CF-GNNEXPLAINER.

2.1. GNNs

Graphs are structures that represent a set of entities (nodes) and their relations (edges). Graph Neural Networks (GNNs) operate on graphs, often via message passing, to produce representations that can be used in downstream tasks such as node classification, link prediction and graph classification. We refer to Battaglia et al. (2018) and Chami et al. (2021) for an extensive overview of existing GNN methods.

Now, let f be any GNN. Most GNN methods have the form $f(A, X; W)$, where A is an $n \times n$ adjacency matrix, X is an $n \times p$ feature matrix (with p features), and W are the learned weights of f. In other words, A and X are the inputs of f, and f is parameterized by W.

Although our CF-GNNEXPLAINER can operate on any type of GNN, we explain our method with a standard, one-layer Graph Convolutional Network (GCN) for node classification (in our experiments we use a three-layer GCN):

$$f(A, X; W) = \text{softmax}\left[\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW\right], \quad (1)$$

where $\tilde{A} = A + I$, I is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ are entries in the degree matrix $\tilde{D}$, X is the node feature matrix, and W is the weight matrix (Kipf & Welling, 2017). A node's representation is learned by iteratively updating the node's features based on its neighbors' features. The number of layers in f determines which neighbors are included: if there are ℓ layers, then the node's final representation only includes neighbors that are at most ℓ hops away from that node in the graph $\mathcal{G}$. The rest of the nodes in $\mathcal{G}$ are not relevant for the computation of the node's final representation. We define the *subgraph neighbourhood* of a node v as a tuple of the nodes and edges relevant for the computation of $f(v)$ (i.e., those in the ℓ-hop neighbourhood of f): $\mathcal{G}_v = (A_v, X_v)$, where A_v is the subgraph adjacency matrix and X_v is the node feature matrix for nodes that are at most ℓ hops away from v. We then define a node v as a tuple of the form $v = (A_v, x)$, where x is the feature vector for v.

2.2. Matrix Sparsification

CF-GNNEXPLAINER uses matrix sparsification to generate counterfactual examples, inspired by Srinivas et al. (2016). They propose a method for training sparse neural networks: given a weight matrix W, a binary sparsification matrix is learned which is multiplied element-wise with W such that some of the entries in W are zeroed out. Here, the objective is to remove entries in the weight matrix in order to reduce the number of parameters in the model. In our case, instead of learning a sparsification matrix to *zero out weights*, we want to *zero out entries in the adjacency matrix* in order to generate counterfactual explanations for GNNs. This corresponds to removing edges which are crucial for the prediction.

3. Problem Formulation

In this section, we formalize the problem of generating counterfactual explanations for GNNs.

3.1. Counterfactual Explanations

In general, a counterfactual example $\bar{x}$ for an instance x according to a trained classifier f is found by perturbing the features of x such that $f(x) \neq f(\bar{x})$ (Wachter et al., 2018). An optimal counterfactual example $\bar{x}^*$ is one that minimizes the distance between the original instance and the counterfactual example, according to some distance function d. The resulting optimal counterfactual explanation is $\Delta_x^* = \bar{x}^* - x$ (Lucic et al., 2020b). With this in mind, we can now define what it means to generate counterfactual explanations for graphs.

3.2. Counterfactual Explanations for Graphs

For graph data, it is not necessarily enough to simply perturb node features, especially since they are not always available. This is why we are interested in generating counterfactual examples by perturbing the graph structure instead. In other

words, we want to change the relationships between instances, rather than change the instances themselves. Therefore, a counterfactual example for graph data has the form $\bar{v} = (\bar{A}_v, x)$, where x is the feature vector and $\bar{A}_v$ is a perturbed version of the original A_v with some edges removed, such that $f(v) \neq f(\bar{v})$. Although it is possible to perturb x as well, this is not what we focus on in this work, since the existence of $\bar{A}_v$ is one of the most important factors that distinguishes graph data from non-graph data. Moreover, there already exists an extensive body of work that focuses on generating counterfactual examples based on feature perturbations for non-graph data (Verma et al., 2020; Karimi et al., 2020).

Following Wachter et al. (2018) and Lucic et al. (2020b), we generate counterfactual examples by minimizing a loss function of the form:

$$\mathcal{L} = \mathcal{L}_{pred}(v, \bar{v}|f, g) + \beta \mathcal{L}_{dist}(v, \bar{v}), \tag{2}$$

where v is the original node and f is the original model. g is the counterfactual model that generates $\bar{v}$, and $\mathcal{L}_{pred}$ is a prediction loss that encourages $f(v) \neq f(\bar{v})$. $\mathcal{L}_{dist}$ is a distance loss that encourages $\bar{v}$ to be close to v, and β controls how important $\mathcal{L}_{dist}$ is compared to $\mathcal{L}_{pred}$. We want to find $\bar{v}^*$ that minimizes Equation 2: this is the optimal counterfactual example for v.

4. Method

To tackle the problem defined in Section 3, we propose CF-GNNEXPLAINER, which generates $\bar{v} = (\bar{A}_v, x)$ given a node $v = (A_v, x)$. To illustrate our method and avoid cluttered notation, let f be a standard, one-layer GCN for node classification as in Equation 1. (For the experiments on which we report in Section 6, we use a three-layer GCN.)

4.1. Adjacency Matrix Perturbation

First, we define $\bar{A}_v = P \odot A_v$, where P is a binary perturbation matrix that sparsifies A_v. Our aim is to find P for a given node v such that $f(A_v, x) \neq f(P \odot A_v, x)$. To find P, we build upon the method by Srinivas et al. (2016) for training sparse neural networks (see Section 2.2). Our objective is to zero out entries in the adjacency matrix (i.e., remove edges). That is, we want to find P that minimally perturbs A_v, and use it to compute $\bar{A}_v = P \odot A_v$. If an element $P_{i,j} = 0$, this results in the deletion of the edge between node i and node j. When P is a matrix of ones, this indicates that all edges in A_v are used in the forward pass.

Similar to Srinivas et al. (2016), we first generate an intermediate, real-valued matrix $\hat{P}$ with entries in $[0, 1]$, apply a sigmoid transformation, then threshold the entries to arrive at a binary P: entries above 0.5 become 1, while those below 0.5 become 0. In the case of undirected graphs (i.e., those with symmetric adjacency matrices), instead of generating $\hat{P}$ directly, we first generate a perturbation vector which we then use to populate $\hat{P}$ in a symmetric manner.

4.2. Counterfactual Generating Model

We want our perturbation matrix P to only act on A_v, not $\tilde{A}_v$, in order to preserve self-loops in the message passing of f (i.e., we always want a node representation update to include the node's representation from the previous layer). To accommodate this, we first rewrite Equation 1 for our illustrative one-layer case to isolate A_v:

$$f(A_v, X_v; W) = \\ \text{softmax}\left[(D_v + I)^{-1/2}(A_v + I)(D_v + I)^{-1/2} X_v W\right] \tag{3}$$

To generate counterfactuals, we propose a new function g, which is based on f, but it is parameterized by P instead of by W. We update the degree matrix D_v based on $P \odot A_v$, add the identity matrix to account for self-loops (as in $\tilde{D}_v$ in Equation 1), and call this $\bar{D}_v$:

$$g(A_v, X_v, W; P) = \\ \text{softmax}\left[\bar{D_v}^{-1/2}(P \odot A_v + I)\bar{D_v}^{-1/2} X_v W\right] \tag{4}$$

In other words, f learns the weight matrix while holding the data constant, while g is optimized to find a perturbation matrix that is then used to generate new data points (i.e., counterfactual examples) while holding the weight matrix constant. Another distinction between f and g is that the aim of f is to find the optimal set of weights that generalizes well on an unseen test set, while the objective of g is to generate an optimal counterfactual example, given a particular node (i.e., $\bar{v}$ is the output of g).

4.3. Loss Function Optimization

We generate P by minimizing Equation 2. We adopt the negative log-likelihood (NLL) loss for $\mathcal{L}_{pred}$:

$$\mathcal{L}_{pred}(v, \bar{v}|f, g) = -\mathbb{1}\left[f(v) = f(\bar{v})\right] \cdot \mathcal{L}_{NLL}(f(v), g(\bar{v})) \tag{5}$$

Since we do not want $f(\bar{v})$ to match $f(v)$, we put a negative sign in front of $\mathcal{L}_{pred}$, and include an indicator function to ensure the loss is active as long as $f(\bar{v}) = f(v)$. Note that f and g have the same weight matrix W – the main difference is that g also includes the perturbation matrix P.

For $\mathcal{L}_{dist}$, we take d to be the element-wise difference between v and $\bar{v}$. Since we do not perturb the feature values, this corresponds to the difference between A_v and $\bar{A}_v$, i.e.,

Algorithm 1 CF-GNNEXPLAINER: given a node $v = (A_v, x)$ where $f(v) = y$, generate the minimal perturbation of $\bar{v} = (\bar{A}_v, x)$, such that $f(\bar{v}) \neq y$.

Input: node $v = (x, A_v)$, trained GNN model f, counterfactual model g, loss function $\mathcal{L}$, learning rate α, trade-off parameter β, number of iterations K, distance function d.

$f(v) = y$ *# Get GNN prediction*
$\hat{P} \leftarrow J_n$ *# Initialization*

for $k \in range(K)$ **do**
  $v^{(k)}$ = GET_CF_EXAMPLE()
  $\mathcal{L} \leftarrow \mathcal{L}(v, \bar{v}^{(k)})$ *# Eq 2 & Eq 5*
  $\hat{P} \leftarrow P^{(k)} + \alpha \nabla_{\hat{P}} \mathcal{L}$ *# Update* $\hat{P}$
end for

Function GET_CF_EXAMPLE()
  $P \leftarrow \text{threshold}(\sigma(\hat{P}^{(k)}))$
  $\bar{A}_v \leftarrow P \odot A_v$
  $\bar{v}^{(k)}_{cand} \leftarrow (\bar{A}_v, x)$
  if $f(v) \neq f(\bar{v}^{(k)}_{cand})$ **then**
    $\bar{v}^{(k)} \leftarrow \bar{v}^{(k)}_{cand}$
    if $\mathcal{L}_{dist}(v, \bar{v}) \leq \mathcal{L}_{dist}(v, \bar{v}^{(k)})$ **then**
      $\bar{v}^* \leftarrow \bar{v}^{(k)}$ *# Keep track of best CF*
    end if
  end if
  return $\bar{v}^*$

the number of edges removed. For undirected graphs, we divide this value by 2 to account for the symmetry in the adjacency matrices. When updating P, we take the gradient of Equation 2 with respect to the intermediate $\hat{P}$, *not* the binary P.

4.4. CF-GNNEXPLAINER

We call our method CF-GNNEXPLAINER and summarize its details in Algorithm 1: given an instance in the test set v, we first obtain its original prediction from f and initialize $\hat{P}$ as a matrix of ones, J_n, to ensure that initially no edges are deleted yet. Next, we run CF-GNNEXPLAINER for a fixed number of K iterations. To find a counterfactual example, we use Equation 4. First, we compute P by thresholding $\hat{P}$, as explained in Section 4.1. Then we use P to obtain the sparsified adjacency matrix which gives us a candidate counterfactual example. This example is then fed to the original GNN, f, and if f predicts a different output than for the original node, we have found a valid counterfactual example, $\bar{v}$. We keep track of the "best" counterfactual example (i.e., the most minimal according to d), and return this as the optimal counterfactual example $\bar{v}^*$ after K iterations. Between iterations, we compute the loss following Equations 2 and 5, and update $\hat{P}$ based on the gradient of the loss. In the end, we retrieve the optimal counterfactual explanation $\Delta^*_v = v - \bar{v}^*$.

5. Experimental Setup

Since there is no prior work that evaluates counterfactual examples for GNNs, we provide a detailed description of experimental design for evaluating counterfactual examples for GNNs.

5.1. Datasets and Models

We use the TREE-CYCLES, TREE-GRIDS, BA-SHAPES node classification datasets from Ying et al. (2019) to run our experiments for generating counterfactual examples. These are synthetic datasets that were created specifically for the task of explaining predictions from GNNs. Each dataset consists of (i) a base graph, (ii) motifs that are attached to random nodes of the base graph, and (iii) additional edges that are randomly added to the overall graph. They are all undirected graphs. The classification task is to determine whether or not the nodes are part of the motif. The purpose of these datasets is to have a ground-truth for the "correctness" of an explanation: for nodes in the motifs, the explanation is the motif itself (Luo et al., 2020). The dataset statistics are available in Table 1.

TREE-CYCLES consists of a binary tree base graph with cycle-shaped motifs, TREE-GRIDS also has a binary tree as its base graph, with 3×3 grids as the motifs. For BA-SHAPES, the base graph is a Barabasi-Albert (BA) with house-shaped motifs, where each motif consists of 5 nodes (one for the top of the house, two in the middle, and two on the bottom). Here, there are four possible classes (not in motif, in motif: top, middle, bottom). We note that compared to the other two datasets, the BA-SHAPES dataset is much more densely connected – the node degree is more than twice as high as that of the TREE-CYCLES or TREE-GRID datasets, and the average number of nodes and edges in each node's computation graph is order(s) of magnitude larger.

We use the same dataset splits (80% train, 10% validation, 10% test) and training setup as in Ying et al. to train a 3-layer GCN (hidden size = 20) for each node classification task. Our GCNs have at least 87% accuracy on the test set.

5.2. Baselines

It is not possible to compare our method to existing methods for explaining individual predictions from GNNs because these methods provide explanations in the form of relevant subgraphs, not minimal perturbations, i.e., they are not

Table 1: Dataset statistics.

	TREE CYCLES	TREE GRID	BA SHAPES
# classes	2	2	4
# nodes	871	1231	700
# edges	1950	3410	4100
Avg node degree	2.27	2.77	5.87
Avg # nodes in A_v	19.12	30.69	304.40
Avg # edges in A_v	18.99	33.94	1106.24

meant for generating counterfactual explanations (see Section 7). We hope that our method can serve as a meaningful baseline for future work on counterfactual explanations for GNNs.

To evaluate CF-GNNEXPLAINER, we compare against 3 different baselines. The first is a random perturbation. We randomly initialize the entries of $\hat{P} \in [-1, 1]$ and apply the same sigmoid transformation and thresholding as described in Section 4.1. We repeat this K times and keep track of the most minimal perturbation resulting in a counterfactual example. The second baseline only keeps edges in the 1-hop neighbourhood of v, while the third removes all edges in the 1-hop neighbourhood of v.

5.3. Metrics

We generate separate counterfactual examples for each node in the graph, and evaluate these counterfactual examples in terms of four metrics: (i) *Coverage*, (ii) *Explanation Size*, (iii) *Subgraph Impact*, and (iv) *Accuracy*. *Coverage* is the proportion of nodes in the dataset that a method is able to generate counterfactual examples for; higher values are better.

Explanation Size is the number of removed edges. It corresponds to the $\mathcal{L}_{dist}$ term in Equation 2: the difference between the original A_v and the counterfactual one $\bar{A}_v$. Since we want to have *minimal* counterfactual examples, we want a small value for this metric.

Subgraph Impact is the proportion of edges in A_v that are removed. A value of 1 indicates all edges in A_v were removed, therefore we want a value close to 0.

Accuracy is the proportion of explanations that are "correct". Following Ying et al. (2019); Luo et al. (2020), we only compute accuracy for nodes that are originally predicted as being part of the motifs, since accuracy can only be computed on instances for which we know the ground truth explanations. An explanation is considered correct if it exclusively involves edges that are inside the motifs. In our case, this means only removing edges that are within the motifs.

5.4. Hyperparameters

We experiment with different optimizers and hyperparameter values for the number of iterations K, the trade-off parameter β, the learning rate α, and the Nesterov momentum m (when applicable). Specifically, we test the number of iterations $K \in \{100, 300, 500\}$, the trade-off parameter $\beta \in \{0.1, 0.5\}$, learning rate $\alpha \in \{0.005, 0.01, 0.1, 1\}$, and Nesterov momentum $m \in \{0, 0.5, 0.7, 0.9\}$. We test Adam, SGD and AdaDelta as optimizers. We find that for all three datasets, the SGD optimizer gives the best results, with $k = 500$, $\beta = 0.5$, and $\alpha = 0.1$. For the TREE-CYCLES and TREE-GRID datasets, we set $m = 0$, while for the BA-SHAPES dataset, we use $m = 0.9$.

5.5. Resources

We run approximately 375 hours of experiments on one Nvidia TitanX Pascal GPU with access to 12GB RAM. We found that on these datasets, CF-GNNEXPLAINER takes approximately 45 seconds on average to generate one counterfactual example when $K = 500$. All code and datasets are available in the Supplementary Material.

6. Results

We evaluate CF-GNNEXPLAINER in terms of the metrics outlined in Section 5.3. The results are shown in Table 2. In almost all settings, we find that CF-GNNEXPLAINER outperforms the baselines in terms of *Explanation Size*, *Subgraph Impact*, and *Accuracy*, which shows that CF-GNNEXPLAINER satisfies our objective to find minimal counterfactual examples with high precision.

6.1. Coverage

In terms of *Coverage*, CF-GNNEXPLAINER outperforms ONLY-1HOP across all three datasets, and outperforms RM-1HOP for TREE-CYCLES and TREE-GRID. We find that RANDOM has the highest *Coverage* in all cases – it is able to find counterfactual examples for every single node. In the following subsections, we will see that this unfortunately comes at a high cost – RANDOM performs poorly on the other three metrics, and it does not satisfy our objective of finding accurate, minimal counterfactual examples.

6.2. Explanation Size

Figures 1 to 4 show histograms of the *Explanation Size* for each of the four methods tested. We see that across all three datasets, CF-GNNEXPLAINER has the smallest (i.e., most minimal) *Explanation Sizes*. This is especially true when comparing to RANDOM and ONLY-1HOP for the BA-SHAPES dataset, where we had to use a different scale for the x-axis due to how different the *Explanation Sizes*

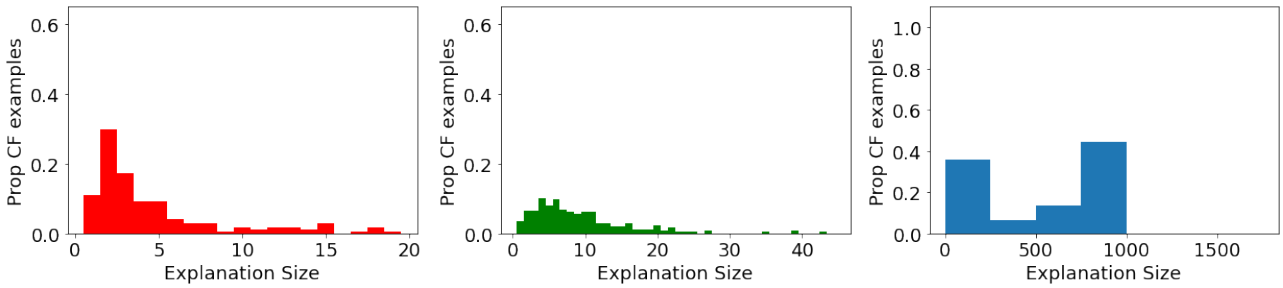

Figure 1: Histograms showing *Explanation Size* from RANDOM for each of the three datasets. Left: TREE-CYCLES, Middle: TREE-GRID, Right: BA-SHAPES. Note the x-axis for BA-SHAPES goes up to 1500.

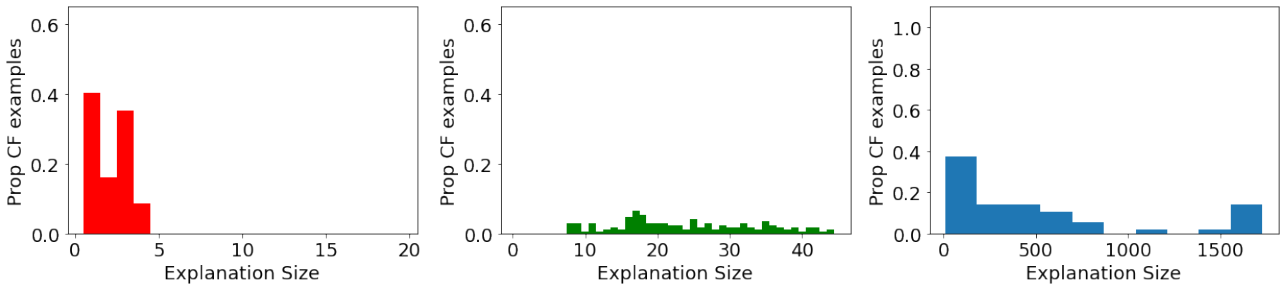

Figure 2: Histograms showing *Explanation Size* from ONLY-1HOP for each of the three datasets. Left: TREE-CYCLES, Middle: TREE-GRID, Right: BA-SHAPES. Note the x-axis for BA-SHAPES goes up to 1500.

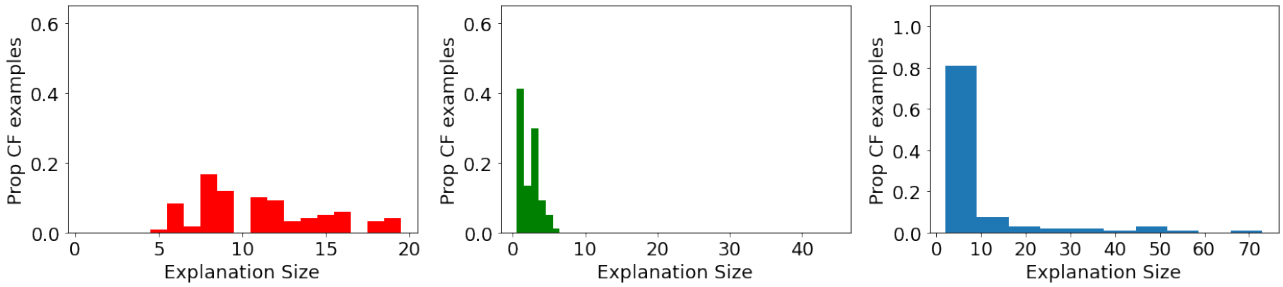

Figure 3: Histograms showing *Explanation Size* from RM-1HOP for each of the three datasets. Left: TREE-CYCLES, Middle: TREE-GRID, Right: BA-SHAPES. Note the x-axis for BA-SHAPES goes up to 70.

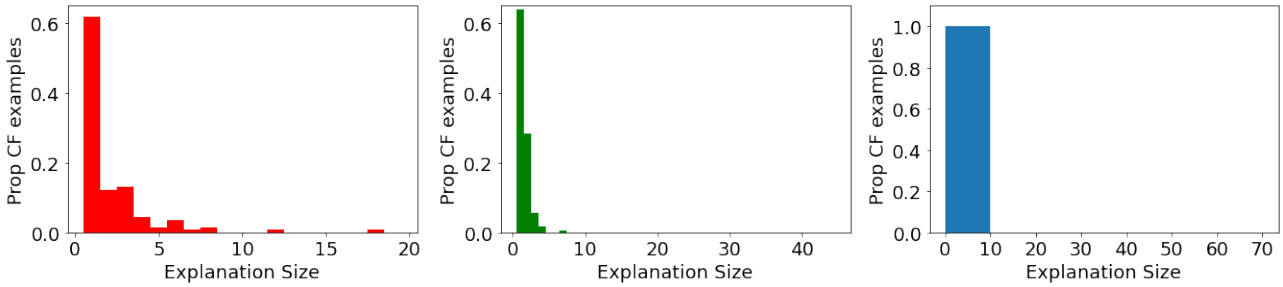

Figure 4: Histograms showing *Explanation Size* from CF-GNNEXPLAINER for each of the three datasets. Left: TREE-CYCLES, Middle: TREE-GRID, Right: BA-SHAPES. Note the x-axis for BA-SHAPES goes up to 70.

were. We postulate that this difference could be because BA-SHAPES is a much more densely connected graph; it has fewer nodes but more edges compared to the other two datasets, and the average number of nodes and edges in the subgraph neighbourhood is order(s) of magnitude larger (see Table 1). Therefore, when performing random perturbations, there is lots of opportunity to perturb edges that do not necessarily need to be perturbed, leading to much larger *Explanation Sizes*. When there are many edges in the subgraph neighbourhood, removing everything except the 1-hop neighbourhood, as is done in ONLY-1HOP, also results in large *Explanation Sizes*. In contrast, the loss function used by CF-GNNEXPLAINER ensures that only a few edges are perturbed, which is the desirable behavior.

6.3. Subgraph Impact

CF-GNNEXPLAINER outperforms all three baselines for all three datasets in terms of *Subgraph Impact*. We note that this metric is much lower for CF-GNNEXPLAINER and RM-1HOP in comparison to the other two methods, which aligns with the results from *Explanation Size*.

6.4. Accuracy

We observe that CF-GNNEXPLAINER has the highest *Accuracy* for the TREE-CYCLES and TREE-GRID datasets, whereas RM-1HOP has the highest *Accuracy* for BA-SHAPES. However, we are unable to calculate the accuracy of RM-1HOP for the other two datasets since it is unable to generate *any* counterfactual examples for nodes in the motifs, likely contributing to the low *Coverage* on those datasets.

We observe *Accuracy* levels upwards of 94% for CF-GNNEXPLAINER across *all* datasets, indicating that it is consistent in correctly removes edges that are crucial for the initial predictions in the vast majority of cases (see Table 2).

6.5. Summary of the results

Taking a holistic view of the results, we find that for all three datasets, CF-GNNEXPLAINER can generate counterfactual examples for the majority of nodes in the test set, while only removing a small number of edges. For nodes where we know the ground truth (i.e., those in the motifs) we achieve at least 94% *Accuracy*.

Although RANDOM can generate counterfactual examples for every node, they are not very minimal or accurate. The latter is also true for ONLY-1HOP – in general, it has the worst scores for *Explanation Size*, *Subgraph Impact* and *Accuracy*.

RM-1HOP is the most competitive baseline, but it performs poorly in terms of *Coverage* for the TREE-CYCLES and TREE-GRID datasets, and its *Accuracy* on these datasets is unknown since it is unable to generate *any* counterfactual examples for nodes in the motifs.

Table 2: Experimental results comparing CF-GNNEXPLAINER (denoted CF-GNN in the table) and the RANDOM baseline.

Metric	Method	TREE CYCLES	TREE GRID	BA SHAPES
Coverage	RANDOM	**1.00**	**1.00**	**1.00**
	ONLY-1HOP	0.68	0.68	0.40
	RM-1HOP	0.53	0.39	0.78
	CF-GNN	0.79	0.93	0.61
Explanation Size	RANDOM	4.70	9.06	503.31
	ONLY-1HOP	15.64	29.30	504.18
	RM-1HOP	2.11	2.27	10.56
	CF-GNN	**2.09**	**1.47**	**2.39**
Subgraph Impact	RANDOM	0.21	0.25	0.42
	ONLY-1HOP	0.87	0.91	0.95
	RM-1HOP	0.11	0.08	0.03
	CF-GNN	**0.10**	**0.06**	**0.002**
Accuracy	RANDOM	0.63	0.77	0.17
	ONLY-1HOP	0.45	0.72	0.18
	RM-1HOP	–	–	**0.99**
	CF-GNN	**0.94**	**0.96**	0.96

7. Related Work

In this section we cover the related work that is relevant for CF-GNNEXPLAINER: methods for GNN explanations (Section 7.1), adversarial attacks on graphs (Section 7.2), and counterfactual explanations in general (Section 7.3).

7.1. Methods for GNN explanations

Several approaches have been proposed to explain the predictions of GNN models – a recent survey of the most relevant work is presented by Yuan et al. (2020b). However, *none* of the existing GNN explanation methods are based on counterfactual explanations, like the one we propose in this work.

GNNExplainer (Ying et al., 2019) generates local, post-hoc explanations in the form of (i) a subgraph and (ii) a subset of node features deemed important to the prediction. It works by sampling many potential subgraphs and then choosing the one with the highest mutual information with the original graph. Reliance on a sampling procedure results in different explanations for the same node when running GNNExplainer multiple times on the same node for the same model. Like the work in (Ying et al., 2019), CF-GNNEXPLAINER also generates post-hoc, local explanations for GNNs. However, CF-GNNEXPLAINER is not based on finding a relevant subgraph, but on finding a

minimal set of edge removals that result in an alternative prediction.

GraphMask (Schlichtkrull et al., 2020) is a post-hoc method for explaining edge importances in each GNN layer. This technique provides local explanations in the form of relevant walks based on erasure search, i.e., looking for the largest subgraph that can be completely discarded. GraphMask operates by training a classifier to predict whether an edge can be dropped without affecting the original predictions. CF-GNNEXPLAINER is similar to GraphMask in that it also provides post-hoc, local explanations, but differs in the form those explanations come in, i.e., CF-GNNEXPLAINER generates counterfactual explanations whereas GraphMask does not.

GraphLIME (Huang et al., 2020) extends the LIME algorithm (Ribeiro et al., 2016) to deep GNN models and studies the importance of different node features for node classification tasks. Given a target node in the input graph, GraphLime considers its N-hop neighboring nodes and their predictions as its local dataset. Then a non-linear surrogate model, Hilbert-Schmidt Independence Criterion (HSIC) Lasso (Yamada et al., 2014), is employed to fit the local dataset. Finally, the subset of important features to explain the HSIC Lasso predictions are considered as the explanations of the original GNN prediction. This differs from CF-GNNEXPLAINER since (i) these are not counterfactual explanations, and (ii) the focus of the explanations is on the node features as opposed to the graph structure.

Contrastive GNN Explanation (Faber et al., 2020) is a method specific to explaining graph classification. Similar to case-based reasoning, explanations for a graph based on *other* graphs from the training set: in particular, this resorts to finding the parts of the graph that make it distant to other graphs with a different label and close to other graphs with the same label. Graph similarity is measured in terms of Optimal Transport (OT) distance. CF-GNNEXPLAINER also provides example-based explanations, but there are some important differences, namely: (i) Contrastive GNN Explanation does not generate new examples but rather locates existing ones in the training set, (ii) the examples are not counterfactual. Moreover, Faber et al. (2020) focus on graph classification while our main task is node classification.

GCN Explanation Baldassarre & Azizpour (2019) and Pope et al. (2019) propose explainability approaches for GCNs by extending common CNN explanation techniques. Explanations come in the form of subgraphs, identifying nodes with positive and negative contributions to the prediction. These approaches differ from CF-GNNEXPLAINER since their explanations are not counterfactual.

XGNN (Yuan et al., 2020a) and **XAI for Graphs** (Schnake et al., 2020) provide *global explanations* for GNNs, which explain the model as a whole. These are in contrast to CF-GNNEXPLAINER which provides local explanations for individual predictions.

7.2. Adversarial attacks on graphs

Adversarial attacks (Sun et al., 2018) are also related to counterfactual examples: they both represent instances obtained from minimal perturbations to the input, which in turn induce changes in the prediction made by the learned model. One difference between the two is in the intent: adversarial examples are meant to fool the model, while counterfactual examples are meant to explain the prediction (Lucic et al., 2020b). In the context of graph data, adversarial attack methods try to make minimal perturbations to the *overall graph* with the intention of degrading model performance. They are not necessarily meant to generate adversarial examples for individual nodes.

7.3. Counterfactual Explanations

There exists a substantial body of work on counterfactual explanations for tabular, image, and text data (Verma et al., 2020; Karimi et al., 2020). Some methods treat the underlying classification model as a black-box (Laugel et al., 2017; Guidotti et al., 2018a; Lucic et al., 2020a), whereas others make use of the model's inner workings (Tolomei et al., 2017; Wachter et al., 2018; Ustun et al., 2019; Kanamori et al., 2020; Lucic et al., 2020b). However, all of these methods are based on perturbing *feature values* to generate counterfactual examples – they are not equipped to handle graph data with relationships (i.e., edges) between data points. CF-GNNEXPLAINER is the first method to provide counterfactual examples for graph data.

8. Conclusion

We propose CF-GNNEXPLAINER, the first method for generating counterfactual explanations for any GNN by generating a perturbation matrix that sparsifies the adjacency matrix. We find that our method is able to generate counterfactual explanations that are (i) minimal, both in terms of the absolute number of edges removed (*Explanation Size*), as well as the proportion of the subgraph neighbourhood that is perturbed (*Subgraph Impact*), and (ii) accurate, in terms of removing edges that we know to be crucial for the initial predictions. We evaluate our method on three commonly used datasets for GNN explanation tasks and find that these results hold across all three datasets.

For future work, we plan to incorporate node feature perturbations in our framework and extend CF-GNNEXPLAINER to accommodate both edge and graph classification tasks. We also plan to investigate the potential of adapting graph attack methods for generating counterfactual explanations.

References

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

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, October 2018.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*, 2021.

Faber, L., Moghaddam, A. K., and Wattenhofer, R. Contrastive Graph Neural Network Explanation. *ICML 2020 Workshop on Graph Representation Learning and Beyond*, pp. 6, 2020.

Goebel, R., Chander, A., Holzinger, K., Lecue, F., Akata, Z., Stumpf, S., Kieseberg, P., and Holzinger, A. Explainable AI: The new 42? In *CD-Make 2018*, pp. 295–303, 2018.

Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., and Giannotti, F. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, May 2018a.

Guidotti, R., Monreale, A., Turini, F., Pedreschi, D., and Giannotti, F. A survey of methods for explaining black box models. *arXiv preprint arXiv:1802.01933*, 2018b.

Halevy, A., Ferrer, C. C., Ma, H., Ozertem, U., Pantel, P., Saeidi, M., Silvestri, F., and Stoyanov, V. Preserving integrity in online social networks. *arXiv preprint arXiv:2009.10311*, 2020.

Huang, Q., Yamada, M., Tian, Y., Singh, D., Yin, D., and Chang, Y. GraphLIME: Local Interpretable Model Explanations for Graph Neural Networks. *arXiv preprint arXiv:2001.06216*, January 2020.

Kanamori, K., Takagi, T., Kobayashi, K., and Arimura, H. DACE: Distribution-Aware Counterfactual Explanation by Mixed-Integer Linear Optimization. *IJCAI*, pp. 2855–2862, 2020. doi: 10.24963/ijcai.2020/395.

Karimi, A.-H., Barthe, G., Schölkopf, B., and Valera, I. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv preprint arXiv:2010.04050*, 2020.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, February 2017.

Laugel, T., Lesot, M.-J., Marsala, C., Renard, X., and Detyniecki, M. Inverse Classification for Comparison-based Interpretability in Machine Learning. *arXiv preprint arXiv:1712.08443*, December 2017.

Lucic, A., Haned, H., and de Rijke, M. Why does my model fail? Contrastive local explanations for retail forecasting. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pp. 90–98, 2020a.

Lucic, A., Oosterhuis, H., Haned, H., and de Rijke, M. Focus: Flexible optimizable counterfactual explanations for tree ensembles, 2020b.

Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. Parameterized explainer for graph neural network. *NeurIPS*, 2020.

Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *arXiv preprint arXiv:1706.07269*, 2017.

Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. Explainability methods for graph convolutional neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10764–10773, Long Beach, CA, USA, June 2019. IEEE.

Ribeiro, M. T., Singh, S., and Guestrin, C. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

Samek, W., Montavon, G., Vedaldi, A., Hansen, L. K., and Müller, K.-R. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019.

Schlichtkrull, M. S., De Cao, N., and Titov, I. Interpreting graph neural networks for NLP with differentiable edge masking. *arXiv preprint arXiv:2010.00577*, October 2020.

Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., and Montavon, G. XAI for graphs: Explaining graph neural network predictions by identifying relevant walks. *arXiv preprint arXiv:2006.03589*, June 2020.

Srinivas, S., Subramanya, A., and Babu, R. V. Training sparse neural networks. *arXiv preprint arXiv:1611.06694*, November 2016.

Stepin, I., Alonso, J. M., Catala, A., and Pereira-Fariña, M. A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, 9:11974–12001, 2021.

Sun, L., Dou, Y., Yang, C., Wang, J., Yu, P. S., He, L., and Li, B. Adversarial attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*, 2018.

Tolomei, G., Silvestri, F., Haines, A., and Lalmas, M. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 465–474, 2017.

Ustun, B., Spangher, A., and Liu, Y. Actionable Recourse in Linear Classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 10–19, 2019.

Verma, S., Dickerson, J., and Hines, K. Counterfactual explanations for machine learning: A review. *arXiv preprint arXiv:2010.10596*, 2020.

Wachter, S., Mittelstadt, B., and Russell, C. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harvard Journal of Law & Technology*, 31(2):841–888, 2018.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

Yamada, M., Jitkrittum, W., Sigal, L., Xing, E. P., and Sugiyama, M. High-dimensional feature selection by feature-wise kernelized lasso. *Neural Comput.*, 26(1): 185–207, 2014.

Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. GNNExplainer: Generating explanations for graph neural networks. *arXiv preprint arXiv:1903.03894*, November 2019.

Yuan, H., Tang, J., Hu, X., and Ji, S. XGNN: Towards model-level explanations of graph neural networks. *arXiv preprint arXiv:2006.02587*, June 2020a.

Yuan, H., Yu, H., Gui, S., and Ji, S. Explainability in graph neural networks: A taxonomic survey. *arXiv preprint arXiv:2012.15445*, 2020b.