Counterfactual Learning on Heterogeneous Graphs with Greedy Perturbation

Qiang Yang
King Abdullah University of Science and Technology
Jeddah, Saudi Arabia
qiang.yang@kaust.edu.sa

Changsheng Ma
King Abdullah University of Science and Technology
Jeddah, Saudi Arabia
changsheng.ma@kaust.edu.sa

Qiannan Zhang
King Abdullah University of Science and Technology
Jeddah, Saudi Arabia
qiannan.zhang@kaust.edu.sa

Xin Gao
King Abdullah University of Science and Technology
Jeddah, Saudi Arabia
xin.gao@kaust.edu.sa

Chuxu Zhang*
Brandeis University
Massachusetts, United States
chuxuzhang@brandeis.edu

Xiangliang Zhang*†
University of Notre Dame
Indiana, United States
xzhang33@nd.edu

ABSTRACT

Due to the growing importance of using graph neural networks in high-stakes applications, there is a pressing need to interpret the predicted results of these models. Existing methods for explanation have mainly focused on generating sub-graphs comprising important edges for a specific prediction. However, these methods face two issues. Firstly, they lack counterfactual validity as removing the subgraph may not affect the prediction, and generating plausible counterfactual examples has not been adequately explored. Secondly, they cannot be extended to heterogeneous graphs as the complex information involved in such graphs increases the difficulty of generating interpretations. This paper proposes a novel counterfactual learning method, named CF-HGExplainer, for heterogeneous graphs. The method incorporates a semantic-aware attentive pooling strategy for the heterogeneous graph classifier and designs a heterogeneous decision boundaries extraction module to find the common logic for similar graphs based on the extracted graph embeddings from the classifier. Additionally, we propose to greedily perturb nodes and edges based on the distribution of node features and edge plausibility to train a neural network for heterogeneous edge weight learning. Extensive experiments on two public academic datasets demonstrate the effectiveness of *CF-HGExplainer* compared to state-of-the-art methods on the graph classification task and graph interpretation task.

CCS CONCEPTS

• **Information systems → Social networks**; • **Computing methodologies → Neural networks**.

*Corresponding authors.
†Dr. Xiangliang Zhang is secondly affiliated with King Abdullah University of Science and Technology, Saudi Arabia.

KEYWORDS

Counterfactual learning, Heterogeneous graphs, Graph neural networks

ACM Reference Format:
Qiang Yang, Changsheng Ma, Qiannan Zhang, Xin Gao, Chuxu Zhang, and Xiangliang Zhang. 2023. Counterfactual Learning on Heterogeneous Graphs with Greedy Perturbation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3580305.3599289

1 INTRODUCTION

Graph Neural Networks (GNNs) have made great success in many graph-related tasks, such as node classification [4], graph classification [26], link prediction [29], and community detection [3]. In addition, they have been widely used in high-stake applications such as chemistry [34], molecular biology [22], physical systems [23], and social networks [8]. However, the predicted results cannot be understood and interpreted in that the reliability of models cannot be satisfied. Therefore, it is urgently needed to design interpretable methods, which can not only achieve good performance but also interpret the prediction. For example, in drug discovery, *if perturbing one functional group on the input molecular graph, will the prediction result be changed?*

Although there exist several interpretable GNN methods [27, 31, 43, 49, 51–54] (also in the survey work [2, 35]), most of them concentrated on generating the subgraph consisting of the extracted important edges for the particular prediction. For example, Ying et al. proposed to identify a compact sub-graph structure and a small subset of node features that have a crucial role in GNN's prediction [51]. Luo et al. proposed a deep neural network-based method to parameterize the generation process of explanations [31]. Yuan et al. explained GNN's predictions by efficiently exploring different sub-graphs with Monte Carlo tree search [52]. In addition, Yang et al. proposed a meta-path-based interpreter using a recurrent neural network to learn relations between node pairs [50]. Although they can achieve good performance and interpret why the models have the corresponding predictions, they still face several challenges: (1) They lack counterfactual validity as removing the sub-graph

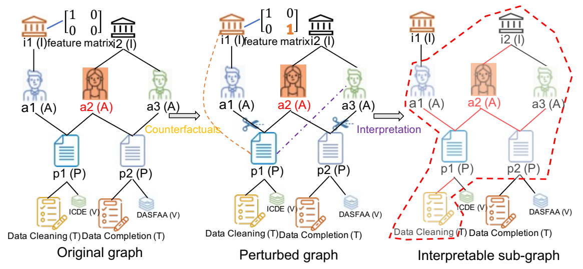


Figure 1: Toy example of counterfactual learning on an academic heterogeneous graph on research interest shift detection of the targeted author a2 (A). Left: the original graph. Middle: the CF perturbed graph. Right: the interpretable sub-graphs indicated by dashed red lines. The identified important edges are highlighted by red lines.

may not affect the prediction. (2) They cannot be easily extended to the heterogeneous graphs because the complex heterogeneous information, i.e., different types of nodes and edges, increase the difficulty of generating the interpretation.

Counterfactual (CF) describes a situation in the form of "If X had not occurred, Y would not have occurred", which can solve the first challenge well. CF has been widely used in tabular data, image data, or text data [20, 42]. However, research on CF interpretation for graph data is insufficient due to the presence of complex structures and node information. For example, Lucic et al. generated CF explanations for GNNs using matrix sparsification techniques by removing minimal edges from the input graph to cause prediction changes [30]. Bajaj et al. proposed generating robust counterfactual explanations on GNNs by explicitly modeling the common decision logic of GNNs on similar input graphs, which are robust to noises [1]. However, They do not fully consider the characteristics of CF; they generate CFs by directly flipping identified important edges. As illustrated in Fig. 1, the perturbation on graph data is limited to randomly removing or adding edges. According to [9], generating good CF examples requires satisfying a set of properties, including *Proximity*: the distance of a CF and original data should be close; *Plausibility*: CFs should be legitimate and reasonable; *Sparsity*: the number of perturbation on nodes/edges should be sparse.

To address these problems, we propose *CF-HGExplainer*, a counterfactual learning method on heterogeneous graphs for explaining the graph classification. Particularly, we first propose a semantic-aware attentive pooling strategy for the heterogeneous graph classifier based on the Relational-GCN [36] (R-GCN). Note that existing graph classification models typically obtain graph representations with readout functions from node representations, such as sum or mean, which treat all nodes/edges equally for homogeneous or heterogeneous graphs [32, 36, 48]. However, this can result in the poor performance of the graph classifier, especially for the classification tasks with target nodes to focus on. To address this, we apply the attention mechanism to calculate the contributions between node pairs, which leads to more accurate graph representations for classification. Then, we design a counterfactual example generator that greedily perturbs the input graph by modifying edges and node features. Moreover, we build the linear decision boundary to classify heterogeneous graphs by explicitly modeling the common decision logic of heterogeneous graph representations and then apply them to make similar graphs closer in the embedding space with the help of heterogeneous edge weights learned by a deep neural network. The contributions of this work are as follows:

- To the best of our knowledge, we are the first to study counterfactual learning on heterogeneous graphs.
- We propose a semantic-aware attentive pooling strategy for the heterogeneous graph classifier based on R-GCN to calculate the contribution of each node pair for graph representations. We design a CF generator that greedily perturbs node features and edges from original graphs to generate proximate, sparse, and plausible CF examples and a heterogeneous decision boundaries extraction module, which are used to train a neural network for edge weights learning.
- We conduct extensive experiments on two public heterogeneous graph datasets to demonstrate that our graph classifier achieves better performance. Also, we show that our CF generator can achieve better performance than the random perturbation, which promotes the model's plausible and understandable.

2 RELATED WORK

Our work is related to heterogeneous graph (HG) representation learning, GNN explanations, counterfactual learning, as well as adversarial attacks.

2.1 HG Representation Learning

For HG representation learning, significant progress has been made in capturing the topological information of HG using random walk-based models such as HeteSpaceyWalkand SHNE [19, 57]. HIN2Vec is another approach that captures rich semantics of relations and graph structure to learn node representations and meta-paths [14]. SpaceyMetapath, on the other hand, formalizes meta-path-guided random walks as higher-order Markov chain processes and introduces a heterogeneous personalized spacey random walk to achieve the expected stationary distribution among nodes [19]. In contrast to GCN, which considers homogeneous edges, R-GCN addresses edge heterogeneity by learning multiple convolution matrices, with each matrix corresponding to one edge type and utilizing K convolutional layers [36]. To handle heterogeneous graph structures and node attributes, HetGNN employs type-specific RNNs to encode features for each type of neighboring node and aggregates the encoded neighbor representations of different types [56]. GEM incorporates attention into heterogeneous GNNs by leveraging distinct aggregators for each relation and learning attention coefficients to model the importance of different types of nodes [28]. Similarly, HAN, a heterogeneous graph attention network, utilizes high-level semantic attention to differentiate and aggregate information from different meta-paths [45]. Furthermore, drawing inspiration from the success of transformers, HGT, a heterogeneous graph transformer, employs each edge's meta relation to parameterize the transformer-like self-attention architecture [21].

2.2 GNN Explanations

GNN explanations can be classified into two main categories: instance-level and model-level methods. Instance-level explanation methods aim to explain the predictions made by a GNN on specific input graph instances. These methods typically identify a subgraph within the input graph as the explanation. For instance, GNNExplainer achieves explanation by removing redundant edges from an input graph instance, maximizing the mutual information between the distribution of subgraphs and the GNN's prediction [51]. PGExplainer, on the other hand, generates explanations using a deep neural network, also maximizing mutual information [31]. PGMExplainer employs a Bayesian network and random perturbations of node features to explain predictions, fitting an explainable dataset and the original GNN model [43]. Lucic et al. propose a method for generating counterfactual explanations for GNNs, ensuring minimal perturbation to the input graph while changing the prediction [30]. On the other hand, model-level explanation methods focus on explaining GNNs by training a graph generator. The generator aims to produce graph patterns that maximize a specific prediction of the GNN model [52].

2.3 Counterfactual Explanations

Counterfactual explanations have gained widespread usage in tabular, image, and text data domains [20, 33, 42]. These methods typically involve perturbing feature values to generate counterfactual examples. For instance, Pawelczyk et al. propose a method to generate faithful counterfactuals specifically for tabular data [33]. Hendricks et al. proposed a method that examines missing evidence in the input data that could contribute to a different classification decision, thereby generating counterfactual explanations [20]. However, these methods are not designed to handle graph data with interconnected nodes, except for two recent works in [1, 30]. RCExplainer in [1] models the decision logic of GNNs on similar input graphs but only flips edges to generate the counterfactual graphs. CF-GNNExplainer in [30] independently optimizes the counterfactual property for each explanation by removing minimal edges. Despite these advances, counterfactual explanations for heterogeneous graphs remain underexplored.

2.4 Adversarial Attacks

CF learning and adversarial attacks are both related to instances obtained through minimal perturbations, resulting in changes in the predictions of learned models. However, the key distinction lies in their objectives: adversarial examples aim to deceive the model, while CF examples aim to explain the prediction [13] (see survey [38, 46]). In the context of graph data, adversarial attacks typically involve minimal perturbations that degrade the overall performance of the model. In contrast, CF explanations aim to directly optimize the counterfactual property of an explanation to provide insight into the prediction. Zhao et al. propose a structural attack against graph-based Android malware detection techniques with a heuristic optimization model integrated with Reinforcement learning framework [58]. Wang et al. propose to evade detection by manipulating the graph structure [44]. Bojchevski et al. provide the first adversarial vulnerability analysis based on random walks where adversarial perturbations are done on structure [7].

Our approach distinguishes itself from these studies in two key ways: firstly, it represents a heterogeneous graph by accounting for the attention score between nodes. Secondly, it crafts counterfactual examples through the perturbation of the input graphs on both the node features and edges, executed in a greedy fashion.

3 PROBLEM FORMULATION

In this section, we first introduce the concept of heterogeneous graph used throughout this paper, then formally define the problem.

Definition 3.1. **(Heterogeneous Graph)** A heterogeneous graph (HG) is defined as $G = (V, E, O_V, R_E)$ with multiple types of nodes V and links E. Particularly, O_V represents the node types for $v \in V$, and R_E represent the relation types for $e \in E$.

Academic graphs are typical HG including O_V as *author*, *paper*, *venue*, *institution*, and *topic* while R_E can be *author_write_paper*, *author_affiliated-by_institution*, *paper_cite_paper*, *paper_has_topic*, *venue_publish_paper*, and the corresponding inverse relations, as shown in Fig. 1.

Definition 3.2. **(CF Learning on HG for Explanation)**. Let $\mathcal{G}$ be a set of training HG with corresponding labels Y, and $f : \mathcal{G} \rightarrow Y$ be the trained classifier. Given an input HG, denoted as G, the target is to predict its label y, and provide an interpretation for this prediction by forming a sub-graph S consisting of the key edges from E in G. Formally, $\{y, S\} = \mathcal{X}(f, G)$. It is crucial to note that f is fixed while building $\mathcal{X}$ for explainable prediction, and G can be a graph seen in $\mathcal{G}$ or one unseen graph that follows the same distribution as $\mathcal{G}$. The interpretation of the prediction satisfies two main requirements. Firstly, it should be counterfactual, meaning that removing/adding edges or perturbing the nodes in S should significantly impact the performance of the original prediction. Note that the perturbations should be sparse, close, and plausible, ensuring the quality of the counterfactual examples. Secondly, the addition of noise to G, such as removing edges, should not drastically alter S. By meeting these requirements, the counterfactual sub-graph S that contains important edges while being resilient to noise, provides high-quality counterfactual explanations.

4 THE PROPOSED MODEL

The overall architecture of the proposed method is depicted in Figure 2 (a). It first applies a pre-trained heterogeneous graph classifier $f : \mathcal{G} \rightarrow Y$ to extract graph embeddings of original graphs, denoted by SAR-GCN(G_i, W^0), where $W^0 = \mathbf{1}$, indicating all edges in G_i have the same weight (at the same level of importance for predicting the label of G_i). In the heterogeneous graph embedding space, the non-linear decision boundary of the graph classifier is approximated by a set of linear boundaries. In order to identify the sub-graph S of G_i for explanation, edge weight W^t is iteratively learned to assign higher weights to those that should be selected in sub-graph S, and lower to others. The finally converged W^T, resulting in the desired sub-graph S, enables SAR-GCN to rely on the important edges for classifying both the original and counterfactual graphs. As depicted in the middle of Fig. 2 (a), G_i and $\tilde{G}_i$ are separated at two different sides of one linear boundary, as they have different labels. However, their representations do not drift away from their counterpart learned by SAR-GCN with W^0.

4.1 Semantic-aware Attentive Pooling Strategy for Heterogeneous Graph Classifier

Heterogeneous graphs pose a challenge on the graph represention compared to homogeneous graphs due to the heterogeneous nodes

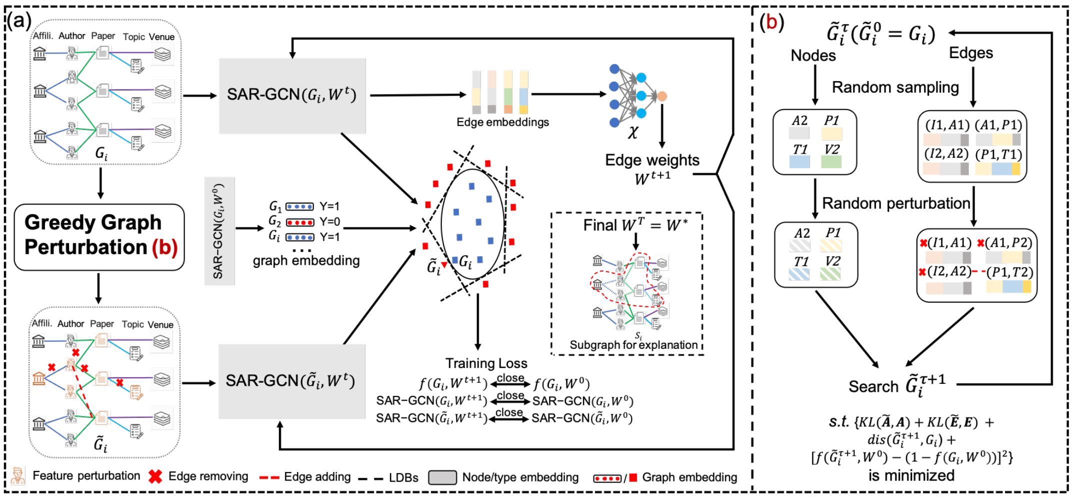

(a)
Affili. Author Paper Topic Venue
G_i
SAR-GCN(G_i, W^t)
Edge embeddings
$\mathcal{X}$
Edge weights W^{t+1}
Greedy Graph Perturbation (b)
SAR-GCN(G_i, W^0)
G_1 Y=1
G_2 Y=0
G_i Y=1
graph embedding
$\tilde{G}_i$ G_i
Final $W^T = W^*$
S_i Subgraph for explanation
Training Loss
$f(G_i, W^{t+1})$ close $f(G_i, W^0)$
SAR-GCN(G_i, W^{t+1}) close SAR-GCN(G_i, W^0)
SAR-GCN($\tilde{G}_i, W^{t+1}$) close SAR-GCN($\tilde{G}_i, W^0$)
SAR-GCN($\tilde{G}_i, W^t$)
$\tilde{G}_i$
Feature perturbation
Edge removing
Edge adding
LDBs
Node/type embedding
Graph embedding
(b)
$\tilde{G}_i^\tau(\tilde{G}_i^0 = G_i)$
Nodes
Edges
Random sampling
A2 P1 T1 V2
(I1, A1) (A1, P1) (I2, A2) (P1, T1)
Random perturbation
(I1, A1) (A1, P2) (I2, A2) (P1, T2)
Search $\tilde{G}_i^{\tau+1}$
s.t. $\{KL(\tilde{\mathbf{A}}, \mathbf{A}) + KL(\tilde{\mathbf{E}}, \mathbf{E}) + dis(\tilde{G}_i^{\tau+1}, G_i) + [f(\tilde{G}_i^{\tau+1}, W^0) - (1 - f(G_i, W^0))]^2\}$
is minimized

Figure 2: (a) The overview of CF-HGExplainer. The interpretable subgraph S is determined by W^T, which is the edge weight learned by $\mathcal{X}$. Using W^T, SAR-GCN relies on the important edges (with high weights in S) for classifying both the original graphs G_i and counterfactual graphs $\tilde{G}_i$. (b) The counterfactual graphs are generated by greedily perturbing the heterogeneous nodes and edges by minimizing the difference between original and counterfactual graphs.

and edges. The existing methods for the homogeneous graph like GCN [24] or heterogeneous graphs, including HAN, HGT, and R-GCN [21, 36, 45], treat each edge equally when pooling for graph representations. However, the contributions of edges in heterogeneous graphs with target nodes can vary significantly. In the context of research interest shift detection, for instance, edges connecting target authors may convey more semantic information than others. To tackle this, we propose a semantic-aware attentive pooling strategy for a heterogeneous graph classifier based on R-GCN, which we refer to as **SAR-GCN** (refer to Fig. 3).

First, following R-GCN, we obtain node representations in G by passing messages between heterogeneous nodes based on the corresponding edges:

$$h_i^{l+1} = \sigma(W_0^l h_i^l + \sum_{r \in R_E} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^l h_j^l), \tag{1}$$

where N_i^r represents the set of neighboring indices of node i under relation $r \in R_E$, $c_{i,r}$ is a normalization constant, and σ is the sigmod function. Next, we calculate the attention scores between nodes based on both their semantics and structural information. These scores determine the importance of node pairs, i.e., edges, when pooling node embeddings using Mean, Max, or Sum. Given a target node n_t and its heterogeneous neighbors $N_t = \{N_t^{r1}, N_t^{r2}, ..., N_t^R\}$, we incorporate node degree information while considering the types of nodes. For each neighbor $N_{t_i} \in N_t^r$, the attention score between n_t and N_{t_i} is computed as:

$$att(n_t, N_{t_i}) = \frac{exp\left(s(\boldsymbol{h_i}, \boldsymbol{h}_{t_i})\right)}{\sum_{r \in R_E} \sum_{j \in N_t^r} exp\left(s(\boldsymbol{h_i}, \boldsymbol{h_j})\right)} \cdot \frac{deg(N_{t_i}) + const}{\sum_{v \in V} \deg(v)}, \tag{2}$$

where $s(\cdot)$ represents a function to calculate the similarity of node embeddings, such as Cosine similarity. $deg(\cdot)$ is the node degree, and $const$ is a constant added to prevent the numerator from becoming zero. Finally, we obtain the graph representation by applying a readout function and considering the attentive scores of nodes for

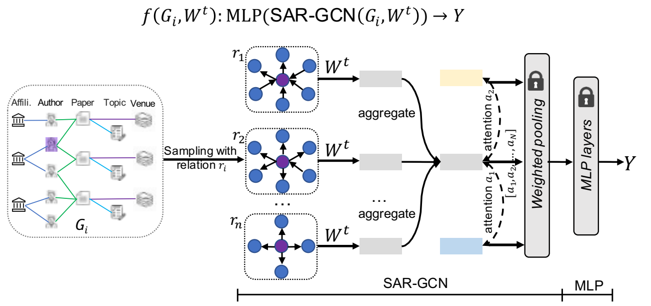


Figure 3: Semantic-aware attentive pooling strategy for the heterogeneous graph classifier. When learning the edge weights W^t, the weighted pooling and MLP layer are fixed.

different edge types as follows:

$$G_{emb} = \sum_{tp \in O_V} Readout(\boldsymbol{h}, att(n_t, N_{t_i})), \tag{3}$$

where tp represents the node type, and N_{t_i} belongs to type tp. The function $Readout(\cdot)$ can be any readout function, such as Mean, Sum, or Max. Our approach benefits from the combination of global structural information, which captures the semantics between nodes under different relations.

4.2 Heterogeneous Decision Boundary Extraction

In our approach, we focus on extracting the decision region of the heterogeneous graph classifier in the d-dimensional output space of SAR-GCN due to its semantic and meaningful information [59]. To elaborate, let $\vartheta_{gc}(\cdot)$ represent the mapping function performed by SAR-GCN from input graph G to its embedding, and $\vartheta_{fc}(\cdot)$ denote the MLP in graph classifier f, mapping from graph embedding to the predicted class. Consequently, the overall prediction made by the heterogeneous graph classifier f can be expressed as $\vartheta_{fc}(\vartheta_{gc}(G))$.

We define $\mathcal{D}$ as the set of linear decision boundaries (LDBs) induced by $\vartheta_{fc}(\cdot)$, which partition the d-dimensional space into multiple convex polytopes, as shown in Fig. 2 (a). When the embeddings of input graphs lie within the same convex polytope, they are assigned the same class label by the classifier. By covering a larger number of graphs, the common decision logic becomes more general, than that derived from single graph instance. Here, we define G_c as the set of graphs predicted to be class c, and $\mathcal{P}$ as a subset of LDBs from $\mathcal{D}$ that partitions the space into convex polytopes. We introduce the notation $r(\mathcal{P}, c)$ to represent the convex polytope induced by $\mathcal{P}$ that covers the largest number of graphs in G_c. Additionally, let $m_c^{\mathcal{P}}$ denote the number of graphs in G_c covered by $r(\mathcal{P}, c)$, and $m_{\hat{c}}^{\mathcal{P}}$ represent the number of graphs in G that are covered by $r(\mathcal{P}, c)$ but are not predicted as class c. The problem of classifying graphs in class c as accurate as possible can be formulated as the constrained optimization problem: $max_{\mathcal{P}\subseteq\mathcal{D}} m_c^{\mathcal{P}}, s.t.\ m_{\hat{c}}^{\mathcal{P}} = 0$.

If we find the optimal $\mathcal{P}$ that solves this problem, we can obtain the desired decision region $r(\mathcal{P}, c)$ through two steps: (1) Counting the number of graphs covered by each convex polytope induced by $\mathcal{P}$; (2) Selecting the convex polytope that covers the largest number of graphs in G_c. However, directly computing $\mathcal{D}$ is computationally intractable since the number of LDBs for a heterogeneous graph classifier can be exponential. To address this, we adopt a sampling strategy to obtain a subset of LDBs, denoted as $\tilde{\mathcal{D}}$. Each LDB can be represented as $a^T x + b = 0$, where a is the weight vector, x is the variable, and b is the bias term. So we can efficiently approximate the decision boundaries and proceed with the optimization problem to find the best decision region for class c while excluding graphs from other classes. Here, a and b can be calculated as follows:

$$\begin{aligned} a &= \frac{\partial(max_i(\vartheta_{fc}(\alpha)) - max_j(\vartheta_{fc}(\alpha)))}{\partial(\alpha)}|_{\alpha=\vartheta_{gc}(G)} \\ b &= max_i(\vartheta_{fc}(\alpha)) - max_j(\vartheta_{fc}(\alpha)) - a^T\alpha|_{\alpha=\vartheta_{gc}(G)}, \end{aligned} \tag{4}$$

where $max_i(\vartheta_{fc}(\alpha))$ and $max_j(\vartheta_{fc}(\alpha))$ are the largest and the second largest values in the vector $\vartheta_{fc}(\alpha)$, respectively. Thus, we can sample a subset of input graphs uniformly, and derive a sample of LDBs as $\tilde{\mathcal{D}} \subset \mathcal{D}$. However, it is still hard to solve the optimization problem due to the strict constraint. We further relax the problem with a tolerance parameter $\delta \geq 0$ as follows:

$$max_{\mathcal{P}\subseteq\mathcal{D}} m_c^{\mathcal{P}},\ s.t.\ m_{\hat{c}}^{\mathcal{P}} \leq \delta, \tag{5}$$

For any two arbitrary sets $\mathcal{M}$ and $\mathcal{N}$ from $\tilde{\mathcal{D}}$, adding a new decision boundary $d \in \tilde{\mathcal{D}}$ to $\mathcal{M}$ may lead to a more decrease than $\mathcal{N}$ in the number of examples induced by $m_c^{\mathcal{M}}$ and $m_c^{\mathcal{N}}$ respectively when $\mathcal{M} \subseteq \mathcal{N}$. Similarly, we can also prove this on $m_{\hat{c}}^{P}$. This is an NP-hard problem, i.e., the Submodular Cost Submodular Cover problem [10]. So we iteratively select the best d that has the most decrease:

$$d = argmin_{d\in\tilde{\mathcal{D}}\setminus\mathcal{P}} \frac{m_c^{\mathcal{P}} - m_c^{\mathcal{P}\bigcup\{d\}} + \mu}{m_{\hat{c}}^{\mathcal{P}} - m_{\hat{c}}^{\mathcal{P}\bigcup\{d\}}}, \tag{6}$$

where $m_c^{\mathcal{P}} - m_c^{\mathcal{P}\bigcup\{d\}}$ is the decrease of $m_c^{\mathcal{P}}$, and $m_{\hat{c}}^{\mathcal{P}} - m_{\hat{c}}^{\mathcal{P}\bigcup\{d\}}$ is the decrease of $m_{\hat{c}}^{\mathcal{P}}$ when adding d into $\mathcal{P}$. Adding a new boundary d to $\mathcal{P}$ may only exclude some graphs from the convex polytope $r(\mathcal{P}, c)$, resulting in non-increasing values of both $m_c^{\mathcal{P}}$ and $m_{\hat{c}}^{\mathcal{P}}$. We further introduce a small positive constant μ to control the decrease. Once we remove all graphs in $\mathcal{P}_c$ for each class, we stop the iteration and return the set of decision boundaries.

4.3 CF Learning on Heterogeneous Graphs

Let $\mathcal{X}$ in *Definition 3.2* be implemented as a $g_\theta(\cdot)$, representing a deep neural network with parameters θ. It is responsible for identifying the interpretable subset of edges in a graph, denoted as $\mathcal{S}$. As shown in Fig. 2 (a), the input to $g_\theta(\cdot)$ is the embedding of each edge, which is derived from the node embeddings and edge type embedding. Let W_{ij} represent the probability that the edge (i, j) exists in the interpretable subgraph $\mathcal{S}$. It is computed as:

$$W_{ij} = g_\theta(e_i, e_j, e_{\psi(ij)}), \tag{7}$$

where e_i and e_j denote the node embeddings of nodes i and j, respectively, obtained from the last layer of SAR-GCN, and $e_{\psi(ij)}$ is the edge type embeddings. The interpretable subgraph $\mathcal{S}$ is acquired by selecting all the edges whose corresponding values in W_{ij} are larger than a predefined threshold, such as 0.5.

The training of g_θ is conducted in an iterative manner. Initially, W^0 is 1 for all edges. The W^t from t-th iteration contributes to the definition of loss functions for letting $\mathcal{X}$ gain the interpretability. First, let y be the prediction from the original f using W^0, and $\hat{y}$ be the prediction from the same f but just using W^t. A prediction loss is defined as:

$$\mathcal{L}_p = -y\log(\hat{y}) - (1 - y)\log(1 - \hat{y}), \tag{8}$$

where $y = f(G_i, w^0)$ and $\hat{y} = f(G_i, w^t)$. Note that the parameters of ϑ_{fc} in f are fixed in the current setting. The W^t participates in the embedding of G_i by SAR-GCN, as shown in Fig. 3. This prediction loss directs W^t to pick up only important edges for SAR-GCN, and resulting in the same prediction result.

For the same reason, a loss function is defined in the graph embedding space. Let $\vartheta_{gc}(G, W^t)$ be the embedding from SAR-GCN with W^t. It is expected that $\vartheta_{gc}(G, W^t)$ and $\vartheta_{gc}(G, W^0)$ always locate at the same side of all linear decision boundaries. We thus define the boundary loss:

$$\mathcal{L}_{bn}^{G} = \sigma(-(\mathcal{P}(\vartheta_{gc}(G, W^t)) \cdot \mathcal{P}(\vartheta_{gc}(G, W^0)))), \tag{9}$$

where $\mathcal{P}(\vartheta_{gc}(G, W^0))$ is the projection results of $\vartheta_{gc}(G, W^0)$ by LDBs in $\mathcal{P}$, presented as a vector. The symbol $\cdot$ is the dot product and σ is the sigmoid function. The loss $\mathcal{L}_{bn}$ is minimized to make the graph embedding $\vartheta_{gc}(G, W^t)$ and $\vartheta_{gc}(G, W^0)$ to lie on the same side of every LDB. The effectiveness of W^t is thus enhanced.

Although Eq. 9 makes $\vartheta_{gc}(G, W^t)$ and $\vartheta_{gc}(G, W^0)$ close enough, it cannot ensure removing edges from S would result in a significant change of the graph prediction label, indicating the importance of edges in S. Therefore, it is essential to investigate the learning of W^t on both the original graph and the counterfactual graph, so as to further boost the capability of $\mathcal{X}$ on identifying the important edges. For a counterfactual graph $\tilde{G}$ of the original graph G, it is also expected that the following loss is minimized:

$$\mathcal{L}_{bn}^{\tilde{G}} = \sigma(-(\mathcal{P}(\vartheta_{gc}(\tilde{G}, W^t)) \cdot \mathcal{P}(\vartheta_{gc}(\tilde{G}, W^0)))), \tag{10}$$

By doing so, the graph embedding of $\tilde{G}$ tends to lie on the different sides of the decision boundaries for G. This further promotes the learning of W^t, thus accurately identifying the important edges. The total boundary loss is thus $\mathcal{L}_{bn} = \mathcal{L}_{bn}^{G} + \mathcal{L}_{bn}^{\tilde{G}}$.

However, perturbing heterogeneous graphs to generate counterfactual graphs $\tilde{G}$ is a challenging task due to the presence of multiple node and edge types. We present our designed greedy graph perturbation method in Fig. 2 (b), and elaborate next on the perturbation process of getting $\tilde{G}$. The algorithm of perturbing heterogeneous graphs is shown in Appendix A.

Since our goal is graph classification, we aim to perturb the target node and its neighbors while maintaining similar distributions of node features compared to the original graph. At the node level, if we perturb the target node, we directly modify its node features according to the desired distribution. If we perturb the neighboring nodes, we first randomly select the node types of the neighbors and then perturb their node features accordingly. At the edge level, we either remove edges that connect the target node or add "rational" edges between the target node and other nodes in the graph. The perturbation process follows a greedy strategy and runs iteratively to find the best candidates on each iteration until a predefined budget Δ is exhausted.

The best candidates on each iteration are selected by taking into account the following measures. (1) The label of the perturbed graph $\tilde{G}$, should be different from the original one G, while they should not differ much by connection and features. The following loss function reflects this requirement:

$$L_{c1} = \lambda_p [f(\tilde{G}, W^0) - (1 - f(G, W^0))]^2 + dis(\tilde{G}, G), \quad (11)$$

where $dis(\cdot)$ is a distance function to calculate the distance between the original graph and the perturbed graph on the node features and graph structure. Here for node features, we calculate the cosine similarity while for the graph structure, we count the number of edges which are different from the original one. λ_p is the parameter to control the weights. (2) The perturbation on graphs should be plausible from two perspectives: first, we cannot modify the node features with potential outliers; second, the addition of edges should be limited to existing edge types. Thus, we define another loss function below:

$$L_{c2} = KL(p||q) + \frac{cnt(E, \tilde{E}) + 1}{max(|E|, |\tilde{E}|)}, \quad (12)$$

where $KL(\cdot)$ is the function of KullbacK-Leibler Divergence, p and q are the node features and edge embeddings extracted from $\vartheta_{gc}(\cdot)$ of the original graph G and perturbed graph $\tilde{G}$, respectively. $cnt(\cdot)$ is the function to count the number of edges whose types are out of the existing ones, and E and $\tilde{E}$ are the set of edges of G and $\tilde{G}$. So the overall loss function of counterfactual learning is:

$$\mathcal{L}_{cf} = \alpha L_{c1} + (1 - \alpha) L_{c2}, \ s.t. \ \text{diff}(\tilde{G}, G) \leq \Delta, \quad (13)$$

where α is the parameter to control the importance of two parts and diff$(\cdot)$ count the number of perturbations on nodes and edges.

4.4 Training

We combine the prediction loss, boundary loss, counterfactual loss, and entropy regularization loss to train the interpreter $\mathcal{X}$, realized as $g_\theta(\cdot)$, as follows:

$$\mathcal{L}(\theta) = \sum_{G \in \mathcal{G}} \lambda \mathcal{L}_p(G, \theta) + (1 - \lambda) \mathcal{L}_{cf}(G, \theta) + \mathcal{L}_{bn}(G, \theta) + \mathcal{L}_{dis}(G, \tilde{G}), \quad (14)$$

where λ is the hyper-parameter to control the importance of each term. The entropy regularization loss is defined as follows:

$$\mathcal{L}_{dis}(G, \tilde{G}) = -\frac{1}{|W|} \sum_{i,j} (W_{ij} \log(W_{ij}) + (1 - W_{ij}) \log(1 - W_{ij})), \quad (15)$$

where W is the learned weight matrix. The entropy loss helps to push the value of each entry in W_{ij} to be close to either 0 or 1. When facing the unseen graphs, we only need to get graph embeddings of the original and counterfactual graphs and apply the learned $g(\theta)$ on them to generate the interpretable sub-graphs.

Time Complexity. The running time of the method is positively related to the number of edges in the input graph G, i.e., the time complexity is $O(|E|)$ when interpreting the prediction for G.

5 EXPERIMENTS

In this section, we conduct extensive experiments to answer the following research questions:

- RQ1: Can our model outperform various baseline methods on heterogeneous graphs classification and explanation?
- RQ2: Can our model have a more robust performance compared to other baselines when noise exists?
- RQ3: What is the impact of the different modules on the model performance?
- RQ4: What is the influence of hyper-parameter settings in the proposed model?

5.1 Experimental Settings

Dataset. To prove the validity of the model, we take the research interest shift detection problem as an example, which aims to predict whether the research interest of researchers shifts in different periods and interpret the reason. We evaluate our model on two academic graph datasets, i.e., MAG [37] and AMiner [39]. We discuss the dataset details in Appendix B.

Evaluation Metrics. We adopt three popular evaluation metrics to evaluate the performance of different models including Accuracy, F1 Score, and ROC-AUC for graph classification. Besides, we use fidelity and sparsity to measure the performance of our proposed interpreter and other baselines. *Fidelity* measures the change of the shift detection result if removing the interpretable edges, $Fidelity(G_e, G) = softmax(f(G)) - softmax(f(G_e))$, where G_e is the interpretable sub-graph (S in our case) and $f(.)$ is the classifier. *Sparsity* reflects the percentage of remaining edges after removing the interpretable edges, $Sparsity(G_e, G) = 1 - \frac{|E'|}{|E|}$, where E' is the edge set of the interpretable sub-graph G_e.

Baseline Methods. For the graph classification task, we compare our proposed method with several groups of baselines: (1) Network embedding models including Node2vec [16] and metapath2vec [11]; (2) Basic GNN-based models including GCN [24], GAT [41], and GraphSAGE [17]; (3) The improved GCN-based models including APPNP [25], GIN [47], and ARMA [5]; (4) The transformed methods for graphs from image representation learning including GraphUNet [15], and GraphResNet [18]; (5) Heterogeneous graph

Table 1: Performance comparisons of all methods with standard deviation on MAG and AMiner datasets.

Model	MAG			AMiner		
	Accuracy	F1 Score	AUC Score	Accuracy	F1 Score	AUC Score
Metapath2vec [11]	0.7235 ± 0.0014	0.7218 ± 0.0008	0.7245 ± 0.0009	0.7003 ± 0.0015	0.6992 ± 0.0012	0.7055 ± 0.0014
Node2vec [16]	0.7372 ± 0.0010	0.7305 ± 0.0016	0.7339 ± 0.0009	0.7230 ± 0.0015	0.7212 ± 0.0010	0.7340 ± 0.0011
GCN [1]	0.7747 ± 0.0010	0.7745 ± 0.0014	0.7775 ± 0.0009	0.7312 ± 0.0008	0.7438 ± 0.0012	0.7517 ± 0.0008
GraphSAGE [17]	0.7591 ± 0.0009	0.7585 ± 0.0007	0.7588 ± 0.0006	0.7500 ± 0.0009	0.7625 ± 0.0012	0.7760 ± 0.0009
GAT [41]	0.7920 ± 0.0010	0.7921 ± 0.0008	0.7989 ± 0.0011	0.7712 ± 0.0008	0.7774 ± 0.0009	0.7820 ± 0.0010
APPNP [25]	0.7708 ± 0.0010	0.7618 ± 0.0012	0.7682 ± 0.0012	0.7518 ± 0.0011	0.7485 ± 0.0009	0.7559 ± 0.0011
GIN [47]	0.7882 ± 0.0006	0.7790 ± 0.0008	0.7892 ± 0.0009	0.7615 ± 0.0008	0.7562 ± 0.0010	0.7609 ± 0.0011
ARMA [5]	0.7459 ± 0.0012	0.7379 ± 0.0010	0.7411 ± 0.0012	0.7338 ± 0.0011	0.7308 ± 0.0012	0.7401 ± 0.0010
GraphUNet [15]	0.7870 ± 0.0010	0.7796 ± 0.0011	0.7850 ± 0.0009	0.7796 ± 0.0010	0.7728 ± 0.0010	0.7782 ± 0.0011
GraphResNet [18]	0.7985 ± 0.0010	0.7938 ± 0.0009	0.7975 ± 0.0008	0.7803 ± 0.0009	0.7780 ± 0.0010	0.7807± 0.0009
HAN [45]	0.8012 ± 0.0009	0.8008 ± 0.0008	0.8107 ± 0.0007	0.7830 ± 0.0010	0.7881 ± 0.0014	0.7913 ± 0.0008
GraphTrans [55]	0.8139 ± 0.0007	0.8147 ± 0.0006	0.8290 ± 0.0008	0.8001 ± 0.0007	0.8039 ± 0.0007	0.8078 ± 0.0006
SAR-GCN w/o attn	0.8201 ± 0.0005	0.8219 ± 0.0006	0.8390 ± 0.0004	0.8129 ± 0.0005	0.8108 ± 0.0004	0.8236 ± 0.0005
SAR-GCN	**0.8462 ± 0.0004**	**0.8435 ± 0.0006**	**0.8442 ± 0.0004**	**0.8200 ± 0.0005**	**0.8221 ± 0.0004**	**0.8391 ± 0.0004**

representation models, i.e., HAN [45]; (6) Graph transformer-based model, i.e., GraphTransformer [55]. Details of baselines are discussed in Appendix C.1. So as to illustrate the effectiveness of our model, we compare our proposed method with interpretable graph learning methods including GRAD [51], ATT [40], GNNExplainer [51], PGExplainer [31], RCExplainer [1], and CF-GNNExplainer [30]. More details about these baselines are in Appendix C.2.

Reproducibility. We follow the general workflow of interpretable graph learning including the classification performance and interpretation performance. More experimental settings can be found in the reproducibility supplement of Appendix D.

5.2 Results and Analysis

5.2.1 Performance Comparison (RQ1). We first report the graph classification results in Table 1, and have the following findings. In network embedding methods, Node2vec outperforms Metapath2vec on both datasets, since random walk with LSTM can capture more semantics of nodes. In addition, GAT achieves better performance on both two datasets due to the usage of the attention mechanism, comparing to GraphSAGE. APPNP achieves similar performance compared to the original GCN, which is consistent with our expectations. Additionally, GIN gets better performance than the original GCN, which demonstrates the effect of graph isomorphism. For the transformed methods, GraphUNet and GraphResNet both achieve comparable performance. This indicates these models can capture the structure information better. HAN has a better performance compared to the above methods since it considers aggregating information from neighbors coming from meta-paths. GraphTransformer improves the performance by 1% - 2% over HAN due to the strong capability to identify useful connections between unconnected nodes. Our proposed model, i.e., SAR-GCN, achieves the best performance in all cases. Compared to GraphTransformer, it has 2%, 2%, and 1% improvement on average in terms of Accuracy, F1-score, and AUC score, respectively. We also prove the effectiveness of the semantic-aware attentive pooling strategy by replacing it with equal weights. The reasons can be summarized as follows: (1) The R-GCN uses the graph structure better when passing messages among neighbors. (2) The semantic-aware attentive pooling strategy helps to improve graph embeddings better than the equal contributions of all nodes.

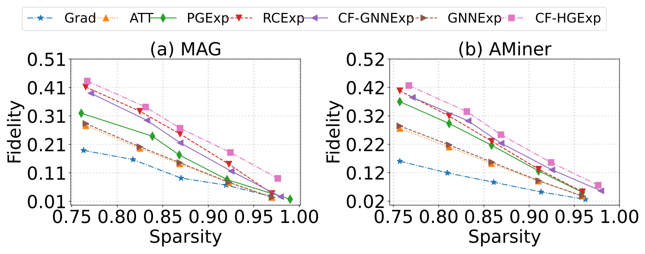


Figure 4: Interpretability evaluation.

For the interpretation task, we compare the performance of all interpretable models on two key metrics: sparsity and fidelity. As shown in Fig. 4, Grad and ATT get worse performance since they cannot identify the important edges accurately. GNNExp performs better because it leverages the recursive neighborhood-aggregation scheme of GNN to identify important graph pathways and node features. Compared to GNNExp, PGExp performs better because of the usage of a parameterized explanation network to collectively provide explanations. In the group of counterfactual learning methods, they perform better than the above-mentioned methods. RCExp performs better than CF-GNNExp since the usage of decision boundaries and counterfactual learning helps to learn accurate edge weights with neural networks. Our proposed model achieves the best fidelity performance at all levels of sparsity on both datasets. The reasons are two folds: (1) The heterogeneous counterfactual generator by perturbing the node features and edges can generate more plausible interpretations than the simple perturbation. (2) The heterogeneous decision boundary extracted by the heterogeneous graph classifier helps to align the graph representation well.

5.2.2 Robustness Comparison (RQ2). We evaluate the robustness of all methods after adding the noises to input graphs. Especially, given an input graph and the learned interpretable sub-graph, we first randomly add noises to node features and edges, then compare whether the prediction on the noisy graph is consistent with the prediction on the original graph. Finally, we use two metrics, i.e.,

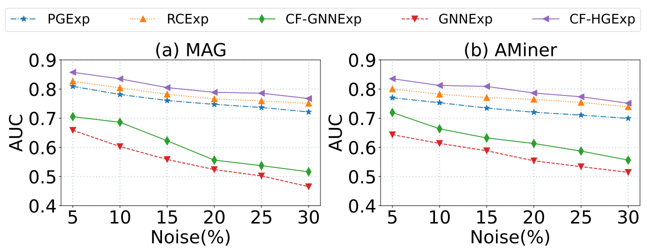


Figure 5: Robustness evaluation for AUC when varying the percentage of noise.

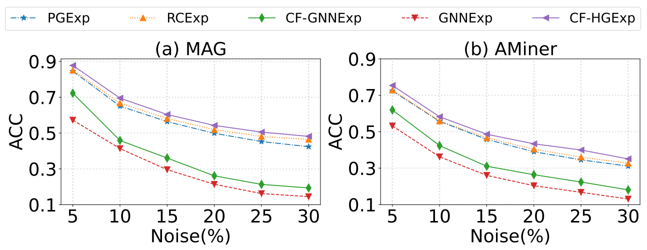


Figure 6: Robustness evaluation for Accuracy when varying the percentage of noise.

ROC-AUC score, and edge accuracy to illustrate the robustness of models. As shown in Fig. 5, when varying the percentage of noise on two datasets CF-GNNExp is more robust than GNNExp on edge AUC score since the counterfactual graph is generated with the small number of edge deleting resulting in a better graph representation. We see that PGExp is generally more robust because it uses the neural network to get the interpretation on all graphs. Compared to PGExp, RCExp is slightly robust due to the contribution of the linear decision boundary. Our model is the most robust to the noises because the heterogeneous decision boundary helps to capture the semantics of graphs and the counterfactual learning model can distinguish important node features and edges. Similarly, we can see in Fig. 6, our model is still robust compared to other methods to the different percentages of noises on the edge accuracy. This means that our model can identify the node features and edges which are crucial to the interpretations.

5.2.3 Ablation Study (RQ3). The ablation study evaluates the effectiveness of different components in our model, including the usage of L_{c1} and L_{c2} in the counterfactual loss. In particular, we compare performances of different model variants: (1) CF-HGExp w/o L_{c1} comes from the removal of Proximity. (2) CF-HGExp w/o L_{c2} comes from the removal of Plausibility. As depicted in Figure 7 (a) of the MAG dataset, we have the following observations: (1) No matter which part we remove in CF-HGExp, the model's performance drops. It indicates the effectiveness of the L_{c1}, i.e., Proximity, and the L_{c2}, i.e., Plausibility, which is consistent with our expectation. (2) When we remove the Plausibility in CF-HGExp, i.e., CF-HGExp w/o L_{c2}, the performance drops more dramatically than the Proximity, i.e., CF-HGExp w/o L_{c1}. This means the usage of valid perturbation is more influential when generating counterfactual graphs. Similar trends can be observed in Fig. 7 (b) of the AMiner dataset.

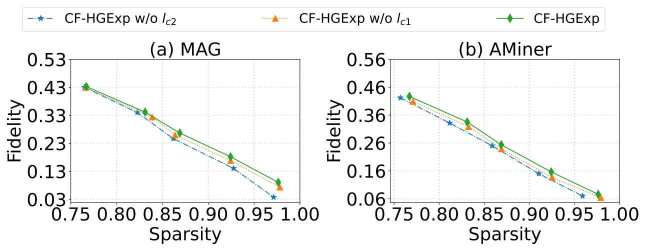


Figure 7: The effectiveness of the counterfactual loss.

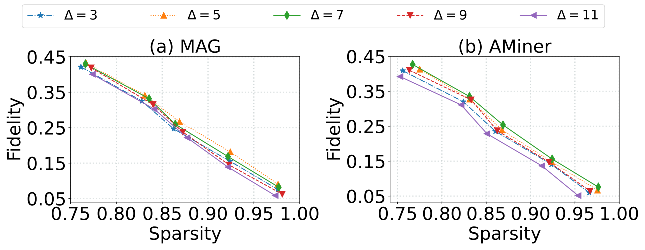


Figure 8: The influence of the budget limitation.

5.2.4 Hyper-Parameter Analysis (RQ4). Since the counterfactual learning needs to satisfy the property of sparsity, we analyze the influence of the sensitivity of the budget, i.e., Δ in Eq. (14). Based on the experiences and experimental results, we limit the number of perturbations on nodes and edges ranging from 3 to 11 with 2 strides on both datasets. As shown in Fig. 8, we find that the performance of our model first went up and arrived at the peak (5 for MAG and 7 for AMiner), and then dropped. This indicates that an optimal number of perturbations strikes a balance between identifying relevant edges and maintaining sparsity in the explanations. However, beyond a certain point, further increasing the budget leads to a decline in performance. This is expected because an excessive number of perturbations may introduce irrelevant edges and dilute the significance of the most important edges.

6 CONCLUSION

In this paper, we develop a novel counterfactual learning method on heterogeneous graphs. First, we propose a semantic-aware attentive pooling strategy based on the Relational-GCN for graph representation. Second, we perturb the input graphs, i.e., edges and node features, with a greedy strategy to iteratively select the most suitable edges or nodes. Moreover, we use the heterogeneous decision boundary to explicitly model the common decision boundaries of heterogeneous graph and apply them to find the important heterogeneous edges as the interpretation with the help of the generated counterfactual graphs. Extensive experiments on two public datasets demonstrate that our model provides good interpretations for graph classification.

ACKNOWLEDGMENTS

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No BAS/1/1635-01-01.

REFERENCES

[1] Mohit Bajaj, Lingyang Chu, Zi Yu Xue, Jian Pei, Lanjun Wang, Peter Cho-Ho Lam, and Yong Zhang. 2021. Robust counterfactual explanations on graph neural networks. In *NeurIPS*. 5644–5655.

[2] Federico Baldassarre and Hossein Azizpour. 2019. Explainability techniques for graph convolutional networks. In *ICML Workshop*.

[3] Punam Bedi and Chhavi Sharma. 2016. Community detection in social networks. *DMKD* 6, 3 (2016), 115–135.

[4] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. 2011. Node classification in social networks. In *SNDA*. 115–148.

[5] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. 2022. Graph neural networks with convolutional arma filters. *TPAMI* 44, 7 (2022), 3496–3507.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3 (2003), 993–1022.

[7] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial attacks on node embeddings via graph poisoning. In *ICML*. 695–704.

[8] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*. 1082–1090.

[9] Yu-Liang Chou, Catarina Moreira, Peter Bruza, Chun Ouyang, and Joaquim Jorge. 2022. Counterfactuals and causability in explainable artificial intelligence: Theory, algorithms, and applications. *Information Fusion* 81 (2022), 59–83.

[10] Victoria Crawford, Alan Kuhnle, and My Thai. 2019. Submodular cost submodular cover with an approximate oracle. In *ICML*. 1426–1435.

[11] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. Metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. 135–144.

[12] Yuxiao Dong, Reid A Johnson, and Nitesh V Chawla. 2015. Will this paper increase your h-index? Scientific impact prediction. In *WSDM*. 149–158.

[13] Timo Freiesleben. 2022. The intriguing relation between counterfactual explanations and adversarial examples. *Minds and Machines* 32, 1 (2022), 77–109.

[14] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. Hin2vec: Explore meta-paths heterogeneous information networks for representation learning. In *CIKM*. 1797–1806.

[15] Hongyang Gao and Shuiwang Ji. 2022. Graph U-Nets. *TPAMI* 44, 9 (2022), 4948–4960.

[16] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *KDD*. 855–864.

[17] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1025–1035.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[19] Yu He, Yangqiu Song, Cheng Li, Jianxand Ji, Jian Peng, and Hao Peng. 2019. Hetespaceywalk: A heterogeneous spacey random walk for heterogeneous information network embedding. In *CIKM*. 639–648.

[20] Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. 2018. Generating counterfactual explanations with natural language. In *ICML*.

[21] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *WWW*. 2704–2710.

[22] Wolfgang Huber, Vincent J Carey, Li Long, Seth Falcon, and Robert Gentleman. 2007. Graphs in molecular biology. *BMC bioinformatics* 8, 6 (2007), 1–14.

[23] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. In *ICML*. 2688–2697.

[24] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[25] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*.

[26] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *KDD*. 1666–1674.

[27] Wanyu Lin, Hao Lan, and Baochun Li. 2021. Generative causal explanations for graph neural networks. In *ICML*. 6666–6679.

[28] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *CIKM*. 2077–2085.

[29] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *PASMA* 390, 6 (2011), 1150–1170.

[30] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. 2022. CF-GNNExplainer: Counterfactual explanations for graph neural networks. In *ICAIS*. 4499–4511.

[31] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. 2020. Parameterized explainer for graph neural network. In *NeurIPS*. 19620–19631.

[32] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.

[33] Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. 2020. Learning model-agnostic counterfactual explanations for tabular data. In *WWW*. 3126–3132.

[34] Douglas EV Pires, Tom L Blundell, and David B Ascher. 2015. pkCSM: Predicting small-molecule pharmacokinetic and toxicity properties using graph-based signatures. *JMC* 58, 9 (2015), 4066–4072.

[35] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. 2019. Explainability methods for graph convolutional neural networks. In *CVPR*. 10772–10781.

[36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*. 593–607.

[37] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *WWW*.

[38] Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S Yu, Lifang He, and Bo Li. 2022. Adversarial attack and defense on graph data: A survey. *TKDE* 1 (2022), 1–20.

[39] Jie Tang, Jing Zhang, Juanzi Yao, Limand Li, Li Zhang, and Zhong Su. 2008. Arnetminer: Extraction and mining of academic social networks. In *KDD*. 990–998.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[42] Sahil Verma, John Dickerson, and Keegan Hines. 2020. Counterfactual explanations for machine learning: A review. In *NeurIPS Workshop*.

[43] Minh Vu and My T Thai. 2020. PGM-Explainer: Probabilistic graphical model explanations for graph neural networks. In *NeurIPS*. 12225–12235.

[44] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking graph-based classification via manipulating the graph structure. In *SIGSAC*. 2023–2040.

[45] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW*. 2022–2032.

[46] Bingzhe Wu, Jintang Li, Chengbin Hou, Guoji Fu, Yatao Bian, Liang Chen, and Junzhou Huang. 2022. Recent advances in reliable deep graph learning: Adversarial attack, inherent noise, and distribution shift. *arXiv preprint arXiv:2202.07114* (2022).

[47] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.

[48] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *KDD*. 1365–1374.

[49] Qiang Yang, Changsheng Ma, Qiannan Zhang, Xin Gao, Chuxu Zhang, and Xiangliang Zhang. 2023. Interpretable Research Interest Shift Detection with Temporal Heterogeneous Graphs. In *WSDM*. 321–329.

[50] Qiang Yang, Qiannan Zhang, Chuxu Zhang, and Xiangliang Zhang. 2022. Interpretable relation learning on heterogeneous graphs. In *WSDM*. 1266–1274.

[51] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating explanations for graph neural networks. In *NeurIPS*. 9240–9251.

[52] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. XGNN: Towards model-level explanations of graph neural networks. In *KDD*. 430–438.

[53] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2023. Explainability in graph neural networks: A taxonomic survey. *TPAMI* 45, 5 (2023), 5782–5799.

[54] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. On explainability of graph neural networks via subgraph explorations. In *ICML*. 12241–12252.

[55] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *NeurIPS*. 5644–5655.

[56] Chuxu Zhang, Chao Song, Dongjand Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *KDD*. 793–803.

[57] Chuxu Zhang, Ananthram Swami, and Nitesh V Chawla. 2019. Shne: Representation learning for semantic-associated heterogeneous networks. In *WSDM*. 690–698.

[58] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. 2021. Structural attack against graph based android malware detection. In *SIGSAC*. 3218–3235.

[59] Daniel Zügner and Stephan Günnemann. 2019. Certifiable robustness and robust training for graph convolutional networks. In *KDD*. 246–256.

A PROCEDURE OF COUNTERFACTUAL GRAPH GENERATION

The pseudo-code for the counterfactual graph generation is as follows.

```
Algorithm 1: Counterfactual Graph Generation
   Input: A graph $G$ and its perturbation budget limitation $\Delta$
   Output: The counterfactual graph $\tilde{G}$
1  The temporal cost for perturbation $\delta = 0$
2  $\tau \leftarrow 0$, and $\tilde{G}^0 \leftarrow G$
3  while $\delta \leq \Delta$ do
4      Randomly sample node type and edge type from $O_V$ and
         $R_E$, respectively
5      Randomly sample a batch of nodes from $V$ and edges
         from $E$ and perturb them by masking node features
         and adding/removing edges as the candidate $\tilde{V}$ and $\tilde{E}$,
         respectively
6      Construct the temporal perturbed graph $\tilde{G}^\tau$ with $\tilde{V}$, $\tilde{E}$
         and remaining parts in $G$
7      Calculate the KL divergence of perturbed nodes and
         edges with $KL(\tilde{G}^\tau, G) = KL(\tilde{\mathbf{A}}, \mathbf{A}) + KL(\tilde{\mathbf{E}}, \mathbf{E})$
8      Calculate $L_{c1}$ and $L_{c2}$ using Eq. 11 and Eq. 12,
         respectively
9      Calculate the counterfactual loss $L_{cf}$ with Eq. 13 to train
         the neural network $g(\theta)$
10     $\delta +=$ diff$(\tilde{G}^\tau, G)$
11     Update $\tilde{G}$ with $\tilde{G}^\tau$
12     $\tau = \tau + 1$
13 end while
14 return $\tilde{G}$
```

B DATASET

In this work, our task is to predict the heterogeneous graph labels and interpret the prediction from the heterogeneous graph classifier with CF. To prove the validity of the model, we take the research interest shift detection problem as an example, which aims to predict whether the research interest of researchers shifts in different periods and interpret the reason. We evaluate our model on two academic graph datasets, i.e., MAG [37] and AMiner [39], which are one kind of heterogeneous graph. The data are processed below. We select papers that were published between 1990 and 2020 in the field of computer science and roughly divide these papers into 8-time spans, each for 3 or 4 successive years. We remove papers that have short abstracts (less than 60 words) or other missing information, like authors, venues, etc. We use LDA [6] to extract the research topics of each paper following [12] since topic modeling can understand the topic distributions well from the papers' abstracts to help the content analysis. The number of topics is set to 70 for both MAG and AMiner datasets, as it leads to the best coherent performance. For each paper, the top-3 representative topics are selected according to the per-paper topic distribution from LDA. The label is set by aggregating topics of authors' major papers published in t (e.g., those papers with citation ≥ 6). To decide whether the author's research interests have shifted, we compare the distributions of topics in two successive time spans $t-1$ and t. If the ratio of new topics in t is larger than 0.5, we believe the interests of this author have changed. The ground-truth label is set to 1 for the shift, and 0 otherwise. We construct the ego network of authors with 2-hop neighbors as the temporal input graphs. We split our data with the ratio 7:2:1 for training, testing, and validation. The dataset statistics are reported in Table 2.

Table 2: Statistics of datasets used in this work.

	#Graphs	#Nodes	#Edges	#Labels
MAG	5,457	89,717	234,384	2
AMiner	2,410	72,805	231,038	2

C BASELINES

C.1 Graph Classificatin

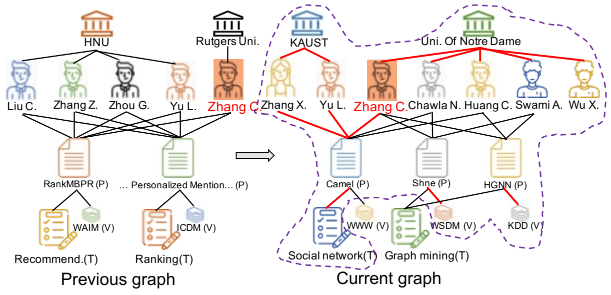
For the graph classification task, we compare our proposed method with several baseline methods below:

- Metapath2vec formalizes meta-path based random walks to construct the heterogeneous neighborhood of a node and then leverages Skip-gram model to learn node embeddings [11].
- Node2vec learns node representations by optimizing a neighborhood information preserving objective with biased random walks for balancing the exploration-exploitation tradeoff [16] .
- GCN is one of the earliest GNN model. We uses the source code from [1] to do graph classification instead of the basic GCN [24].
- GAT learns the importance of different nodes within a neighborhood leveraging masked self-attentional layers when dealing with different-sized neighborhoods without the necessity of knowing the entire graph structure upfront [41].
- GraphSAGE is a general inductive framework that leverages node feature information to efficiently generate node embeddings [17].
- APPNP uses the relationship between graph convolutional networks (GCN) and PageRank to derive an improved propagation scheme based on personalized PageRank [25].
- GIN uses a provably maximally powerful GNN under the neighborhood aggregation framework [47].
- ARMA uses the auto-regressive moving average filter to better captures the global graph structure [5].
- GraphUNet [15] is a representative transformed method to learn graph representations.
- GraphRestNet [18] is another representative transformed method.
- HAN uses a heterogeneous GNN based on the hierarchical attention to learn node embeddings [45].
- GraphTransformer uses Graph Transformer Networks to generate new graph structures and learn effective node representation on the new graphs [55].

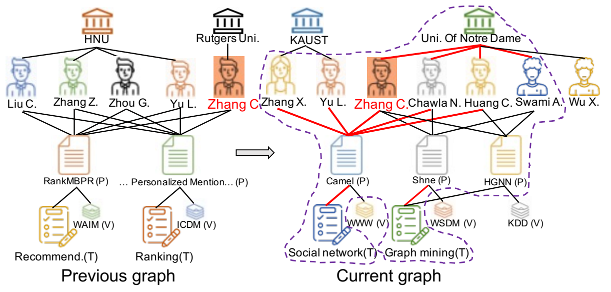
C.2 Graph Interpretation

So as to illustrate the effectiveness of our model, we compare our proposed method with interpretable graph learning methods.

- GRAD learns weights of edges by computing gradients of GNN's objective function w.r.t. the adjacency matrix [51].



(a) RCExp



(b) CF-HGExp

Figure 9: Interpretable results of selected target author Zhang C. where important edges identified by our model and RCExp are highlighted in dotted purple lines and the sub-graphs circled out by red dash lines are the constructed interpretable graph.

Table 3: Time cost (in seconds) of explanation for a single unseen graph sample in two datasets.

Dataset	PGExp	RCExp	CF-GNNExp	GNNExp	CF-HGExp
MAG	0.0068±0.0011	0.0060±0.0008	2.0410±0.2644	0.6125±0.2165	0.0051±0.0003
AMiner	0.0217±0.0023	0.0201±0.0017	7.5953±0.3106	2.3875±0.3058	0.0182±0.0015

- ATT utilizes self-attention layers to distinguish edge attention weights from GAT [40]. Each edge's importance is obtained by averaging its attention weights across all attention layers.
- GNNExplainer is a post-hoc method providing explanations for every single instance [51], GNNExp for short.
- PGExplainer adopts a deep neural network to parameterize the generation process of explanations [31], PGExp for short.
- RCExplainer proposes to generate robust counterfactual explanations on GNNs by explicitly modeling the common decision logic of GNNs on similar input graphs [1], RCExp for short.
- CF-GNNExplainer proposes generating CF explanations for GNNs with minimal perturbation to the input graph to change the predictions [30], CF-GNNExp for short.

D REPRODUCIBILITY

We follow the general workflow of interpretable graph learning including the classification performance and interpretation performance. For a fair comparison, we train all models with the Adam optimizer. The learning rate is 0.001. The embedding size is 32. The epoch is set as 100. We use an early stop strategy, dropout layer (drop rate is 0.25), and batch normalization to get rid of the over-fitting problem. If the performance of validation data is not improved within 15 epochs, we stop the training. The hidden size of the fully connected layer is 128. In GNN-based methods, the number of aggregation layers is 2. In GraphTransformer, the hidden size for the feed-forward layer is 1024 and the number of self-attention layers is 1. λ_p in Eq. (12), α in Eq. (14) and λ in Eq. (15) are 0.5, 0.5, and 0.5 for both datasets. We run all methods 5 times to calculate the standard deviation.

E ADDITIONAL EXPERIMENTAL RESULTS

E.1 Case Study

We provide an interpretable example of our model by randomly selecting an input graph from the MAG dataset, to better show our model's results and promote the understanding of why the interests of target authors change at different periods. We also provide the graph of the target author at the previous period at the left hand. Note that: (1) The author with a yellow background is the target node. (2) The important edges recognized by models are highlighted with a dotted purple line. As shown in Fig. 9, we find that some edges have higher importance, though there exist several same types of edges. By comparing with the graph from the previous period of the target author, we find that: (1) The author Zhang C. moved to a new institution and collaborated with other authors. These collaborators can be the advisor or colleagues of the target author. (2) The target author co-worked with new co-authors whose research interests affect the interests. (3) RCExp involves nodes and edges which are not strongly related to the shift. But our model CF-HGExp can select the most suitable nodes and edges to construct the sub-graph. These indicate that the generated sub-graph by CF-HGExp has a better ability to interpret the prediction by the heterogeneous graph model.

E.2 Efficiency

We evaluate the average time cost, in seconds, for generating an explanation for a single unseen graph sample on two datasets, as shown in Table 3: We observe that CF-HGExp is as efficient as RCExp and PGExp, as they can be used directly on unseen data without requiring any retraining. Furthermore, they are significantly faster than CF-GNNExp and GNNExp.