Graphfool: Targeted Label Adversarial Attack on Graph Embedding

Jinyin Chen, Guohan Huang, Haibin Zheng, Dunjie Zhang, and Xiang Lin

Abstract—Graph embedding learns low-dimensional representations for nodes or edges on the graph, which is widely applied in many real-world applications. Excessive graph mining promotes the research of attack methods on graph embedding. Most attack methods generate perturbations that maximize the deviation of the prediction confidence. They are difficult to accurately misclassify the instances into the target label, and the nonminimized perturbations are more easily detected by defense methods. To address these problems, we propose Graphfool, a novel targeted label adversarial attack on graph embedding. It can generate adversarial graphs to attack graph embedding methods via classification boundary and gradient information in the target graph embedding method. Graphfool first estimates the classification boundaries of different categories. Then, it calculates the minimum perturbation matrix to misclassify the attacked node according to the target classification boundary. Finally, the adjacency matrix is modified according to the maximum absolute value of the perturbation matrix. Extensive experiments demonstrate that Graphfool achieves the state-of-the-art attack performance with minimum perturbations. Besides, the possible defense experiments further prove that the perturbations generated by Graphfool are more imperceptible.

Index Terms—Deep learning, graph embedding, node classification, targeted label attack.

NOMENCLATURE

\( G = (V, E, X) \) Input graph with nodes \( V \), edges \( E \), and node attribute \( X \).

\( G = (A, X) \) Simplified graph of replacing the \( V \) and \( E \) with the adjacency matrix \( A \).

\( \tilde{A} \) Normalized adjacency matrix.

\( A' \) Adjacency matrix added self-connections.

\( G' = (A', X) \) Adversarial graph with graph structure perturbations.

\( N \) Number of nodes on graph \( G(G') \).

\( C \) Attribute dimension of node \( v \in V \).

\( F \) Category set of nodes on graph \( G(G') \).

\( Y \) Real label confidence list.

\( V_L \) Training node set with labels.

\( f_W(\cdot) \) Graph embedding model with the parameters \( W \), e.g., GCN.

\( \mathcal{L} \) Loss function of graph embedding model \( f_W(\cdot) \).

\( Z \) Output of the graph embedding model \( f_W(\cdot) \).

\( H \) Hidden layer’s dimension of GCN.

\( W_0 \in \mathbb{R}^{C \times H} \) Input-to-hidden weight matrix of GCN.

\( W_1 \in \mathbb{R}^{H \times |F|} \) Hidden-to-output weight matrix of GCN.

\( W_{\text{linear}} \in \mathbb{R}^{N \times |F|} \) Weight matrix of the linear classifier.

\( b_{\text{linear}} \in \mathbb{R}^{N \times |F|} \) Bias matrix of the linear classifier.

\( k \) Attack scale for perturbation-limited attack.

\( S \) and \( \sigma \) Sigmoid and ReLU active functions of GCN.

\( \eta \) Learning rate.

\( \|\cdot\| \) L2 norm.

I. INTRODUCTION

Graph networks are applied in many real-world scenarios, such as social networks [1], traffic networks [2], communication networks [3], and biological networks [4]. The graph embedding, which can learn low-dimensional representations for nodes and edges (edge \( \equiv \) link), provides an effective and efficient manner to analyze graphs [5]. Graph embedding is used in many real-world applications, such as link prediction [6], [7], node classification [8], [9], and community detection [10], [11].

Generally, existing graph embedding techniques transform the graph into a similarity graph and calculate its eigenvectors, e.g., ISO-MAP [12], Laplacian eigenmap [13], and local linear embedding [14]. As the recent development of machine learning, researchers try to apply the deep learning methods to graph embedding [15]. Graph neural network (GNN) is a semi-supervised graph embedding method. It extends the existing
neural networks to the graph domain [16] by extracting representations from the graph structure or features. Recently, many GNNs with excellent performance are proposed, such as graph convolutional network (GCN) [17], graph network (GN) [18], and graph autoencoder (GAE) [19].

With the widespread application of graph embedding methods, their security issues have also become a popular research direction. The concerns about excessive graph mining and the requirement to protect personal privacy have promoted the research of adversarial attacks [20]–[27] on graph embedding methods. According to different modification strategies, the attack methods on graph-structured data can be mainly categorized as graph structure attacks [20]–[22], [24], [27], feature attacks [20], [26], and graph injection attacks [23], [25]. Since graph structure information usually plays a more crucial role in graph embedding [20], [24], existing works pay more attention to graph structure attacks. More specifically, the attackers first obtain the perturbation candidate set through different optimizers, such as gradient learning [22], evolutionary computing [28], and meta-learning [29]. Then, the perturbations with the maximum impact on graph analysis tasks are selected from the candidate set. In this work, we pay attention to the graph structure attack on node classification.

In general, the attacker may want the model to predict the target node as the expected label (label \equiv category), which is more in line with the requirements of realistic scenarios. For instance, in a shopping platform, a user who is disturbed by excessive advertisements can hide as a specific user (the label of the user is whether interested in the specific item) by changing his attributes or creating relationships with specific users. The targeted label attack method helps the user to avoid their privacy interests being detected by the recommendation system. Besides, the user can receive information about the interests that they are willing to disclose rather than the items they are not interested in. However, the existing works mainly focus on obtaining the wrong prediction result of the target example [20]–[22] or reducing the prediction accuracy of the overall graph analysis model [29] as much as possible, rather than the targeted label attack. It means that the users are still bothered by the redundant information, although the privacy is protected by applying the existing methods.

Besides, with the improvement of the robustness enhancement of graph embedding methods [24], [30]–[36], how to improve the concealment of adversarial examples to avoid defense/detection methods is also an urgent problem to be solved. Several works [24], [34], [36] point out that the key to an effective attack on the graph is to add edges between dissimilar nodes. Take the two classic adversarial methods, NETTACK [20] and fast gradient attack (FGA) [22] as examples. NETTACK selects candidate edges and node features by graph properties and generates adversarial attacks with the guidance of two score functions iteratively to fool the classifier, while FGA modifies the edges in the graph via the maximal absolute edge gradient. As shown in Fig. 1(a), NETTACK and FGA prefer to add perturbations between more dissimilar nodes. Other attack methods are also suffered from the same problem since their essence is to generate perturbations that have the maximum impact on the target examples’ prediction confidence. Thus, they ignore the similarity between the connected nodes. Fig. 1(b) shows the defense method based on node similarity. The defense methods [24], [34], [36] effectively defend most attack methods based on the core idea of removing those edges that connect dissimilar nodes. This indicates that the stealthiness of the graph adversarial perturbations at the node similarity level is also worthy of consideration.

In summary, although the existing adversarial attacks on graph embedding methods have achieved satisfying attack performance, they are still challenged in two aspects.

1) The Targeted Label Attack: Existing works focus on reducing the performance of the target graph embedding methods, ignoring the targeted label attack that has wide application prospects in practice.

2) Stealthiness: Most of the existing attack methods sacrifice stealthiness in exchange for attack performance, which makes these attacks easy to be successfully defended via node similarity. Thus, it is an open problem of enhancing the stealthiness of the attack on the premise of ensuring the attack’s performance.

As mentioned above, it is still a challenge to bring up an attack method that considers both the targeted label attack and the stealthiness of the perturbations. In the computer vision area, Moosavi-Dezfooli et al. [37] generated minimum perturbations that are sufficient to change the classification label based on an iterative linearization of the classifier. Inspired by this idea, we propose Graphfool to achieve a precise targeted label attack with minimum perturbations. We generate adversarial perturbations based on the minimum distance from the instance to the constructed classification boundary of the target model. It can minimize the change in the prediction confidence of the target instance, thereby improving the stealthiness of the perturbations at the node similarity level. The main contributions of our work are summarized as follows.

1) We propose a targeted label adversarial attack on graph embedding methods, namely, Graphfool. It achieves a precise targeted label attack with more
imperceptible perturbations by reversing the classification boundary-based optimization procedure of the target graph embedding model.

2) Regarding the stealthiness of Graphfool attack and the processing power of the local network, we propose a perturbation-limited attack to limit the perturbation scale. Meanwhile, it can reduce the complexity of Graphfool.

3) Extensive experiments demonstrate that Graphfool achieves the state-of-the-art attack performance in a stealthier manner at node similarity level. Besides, the possible defense experiments further prove that the perturbations generated by Graphfool are more imperceptible than baselines.

The rest of this article is organized as follows. Section II introduces the related work of the graph embedding, the graph attack, and the graph defense methods. Section III discusses the basic theories and techniques of GCN. Section IV describes the Graphfool technique in detail. Section V evaluates our Graphfool method on several real-world datasets. Section VI concludes this article and describes the future works.

II. RELATED WORK

The related work can be generalized into three categories: graph embedding methods, graph attacks, and graph defense.

A. Graph Embedding Methods

Recently, studies are focused on embedding graph into a low-dimensional vector space based on the word2vec model [38]. They are called shallow graph embedding, such as DeepWalk [6], Node2vec [39], LINE [40], and GraRep [41]. DeepWalk [6] is the first model to learn language from a graph, which uses random walk to sample a sequence for each node and treats these generated sequences as sentences using the skip-gram mechanism. Tang et al. [40] proposed a novel graph embedding method LINE, which is a special case of DeepWalk with the window size of contexts set to one. Inspired by DeepWalk, Grover and Leskovec [39] proposed an extension of DeepWalk, called Node2vec. Node2vec employs a biased second-order random walk model to provide more flexibility for generating the context nodes. Cao et al. [41] proposed an embedding method called GraRep, which can preserve the node proximity by constructing different k-step probability transition matrices.

Furthermore, many deep embedding methods [17], [42], [43] are also proposed, which are generally based on deep learning models, such as convolutional neural network (CNN) [44], [45] and generative adversarial network (GAN) [46]. Kipf and Welling [17] proposed GCN for semisupervised node classification, which scales linearly in the number of graph nodes and learns hidden layer representations by encoding both local graph structure and features of nodes. Similarly, Pham et al. [43] proposed column network (CLN), which is a deep learning model for collective classification. Compared with GCN, CLN emphasizes relation learning, which can process multirelational data.

Monti et al. [47] proposed a unified framework, MoNet, which extends the convolution operation to non-Euclidean domains. To strengthen the extraction of critical information, Wang et al. [48] proposed a nonlocal neural network, which has the ability to capture more detailed information of graph structure. Wang et al. [49] proposed GraphGAN, which combines two classes of graph representation learning techniques.

B. Graph Attacks

Existing attack methods have designed various optimizers to generate effective perturbations to the target graph embedding model. According to different modification strategies, these attack methods mainly focus on graph structure attacks, graph feature attacks, and graph injection attacks.

1) Graph Structure Attacks: Zügner et al. [20] proposed NETTACK, which is the first adversarial attack on graph. It generates the adversarial graph with the guidance of score functions within a limited budget. Zügner and Günnemann [29] further proposed Meta-Self, which regards the graph as an optimistic hyperparameter and uses meta-gradient to optimize without knowledge of the classification model and its training weights. Dai et al. [21] proposed RL-S2V and they modified the graph structure with the prediction feedback from the target classifier based on reinforcement learning. Chen et al. [22] proposed FGA, and it extracts the gradient of pairwise nodes and then selects the pair of nodes with maximum absolute edge gradient to update the adversarial graph. To reduce the error caused by the gradient information on the discrete graph data, Wu et al. [24] introduced an integral gradient to accurately reflect the effect of perturbing certain edges while still benefiting from the parallel computations. Considering the attack in the black-box scenario, GF-Attack [27] regards the graph embedding model as the new graph signal generated by the graph filter and feature transformation, attacking the graph filter instead of the graph embedding model.

2) Graph Feature Attacks: Compared with graph structure attacks, although the graph feature attacks are less effective, they enrich the attack strategies with different requirements. For feature attacks, Bose et al. [50] regarded adversarial attacks as a generative problem. They proposed an encoder–decoder framework (DAGAER) to generate node features with small perturbations. To achieve a more imperceptible attack, Takahashi et al. [26] proposed POISONPROBE, which attacks the node features of the target node’s two-hop or even multihop neighbor nodes.

3) Graph Injection Attacks: The graph injection attacks address the issue that the existing graph structure or features cannot be modified in some cases, e.g., the attacker is unable to modify the database that stores the graph data due to the lack of permission. Wang et al. [23] connected fake nodes to existing nodes based on a greedy algorithm and designed a discriminator to ensure the stealthiness of the perturbations. In addition to the greedy algorithm, NIPA [25] performs the fake node attack by training deep reinforcement learning agents based on reinforcement learning. TDGIA [51] connects
the smooth feature optimization-based nodes with the original nodes, which are chosen by the topological defective edge selection strategy.

In general, the optimization goal of the current attack methods is to find the perturbations that maximize the deviation between the prediction confidence and the ground truth, which makes it difficult to accurately misclassify the instances into the target label. Besides, they may cause unnecessary and excessive deviations in the prediction confidence, which are easily detected by some defense methods.

C. Graph Defense

Since the attack methods have revealed the vulnerabilities of graph embedding algorithms, the defense methods for graph embedding are also under intensive development. They mainly enhance the robustness of the graph embedding algorithms by modifying the input data or the model structure.

1) Data-Based Defense: The core idea of these defense methods is to remove the adversarial edges and restore the deleted edges. Entezari et al. [52] used the top singular component to reconstruct the graph to improve the ability of GCN to defend NETTACK attack. Wu et al. [24] measured the similarities between nodes by the Jaccard index and removed the edges that connect very dissimilar nodes. This also improves the robustness of GCN but is suitable for more attack methods. Pro-GNN [36] and GNNGUARD [34] are general defense mechanisms applicable to different GNN models. Pro-GNN iteratively reconstructs a clean graph by preserving the low rank, sparsity, and feature smoothness of the graph. GNNGUARD dynamically updates the edges’ weight according to the correlation between node features and graph structure.

2) Model-Based Defense: These defense methods aim to enhance the robustness of the model itself. As one of the most commonly used defense methods, adversarial training [31], [53], [54] mixes adversarial graphs with normal ones for model training, thus enhancing the model’s ability to resist adversarial attacks at the model parameter level. For the model structure level, RGCN [55] utilizes the Gaussian distribution as the hidden representation of nodes in the graph convolutional layer and assigns attention weights to neighbor nodes based on their variance. Chen et al. [33] trained a distillation GCN model by using the output confidence of the initial GCN as a soft label. Bojchevski and Günnemann [30] optimized the loss function of GNNs, which effectively improved the certifiable robustness.

III. PRELIMINARIES

First, we define notations that are used throughout this article. A graph can be represented by $G = (V, E, X)$, where $V$ is the node set with $|V| = N$ and $E$ is the edge set. $X \in \mathbb{R}^{N \times C}$ is the node attribute matrix, where $C$ denotes the dimension of $X$. Generally, the adjacency matrix $A$ contains the information of $V$ and $E$, so we use $G = (A, X)$ to represent a graph more concisely. The definition of notations is briefly summarized in the Nomenclature section.

A. GCN Model

As an application of traditional CNNs in the graph domain, GCN uses an efficient layer-wise propagation rule based on the first-order approximation of spectral convolutions on graphs and has achieved satisfying performance in the semisupervised node classification. Numerous graph attacks [20], [22], [29], [56] have shown that the adversarial graphs generated with GCN as the target model have excellent transferability, i.e., they can also achieve satisfying attack performance in other graph embedding algorithms. Therefore, we adopt but are not limited to GCN as our target classifier to generate adversarial graphs, thus deceiving various graph embedding algorithms.

Specifically, we employ a two-layer GCN model with softmax classifier, which can be defined as

$$Z = \text{softmax}(\tilde{A}\sigma(\tilde{A}X W_0) W_1)$$  \hspace{1cm} (1)

where $\tilde{A} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$, $A$ is the adjacency matrix, and $\tilde{A} = A + I_N$ is the adjacency matrix of the graph with self-connections. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ denotes the degree matrix of $\tilde{A}$. $W_0 \in \mathbb{R}^{C \times H}$ is the input-to-hidden weight matrix with the hidden layer of $H$ feature maps, and $W_1 \in \mathbb{R}^{H \times |F|}$ is the hidden-to-output weight matrix. The value of $H$ determines the quality of the learned low-dimensional representation $Z$, and it is usually selected based on experimental results. $\sigma$ denotes the ReLU active function.

The loss function is defined as the cross-entropy error over all labeled nodes

$$L = -\sum_{l=1}^{c} \sum_{k=1}^{n_l} Y_{lk} \ln(Z_{lk})$$  \hspace{1cm} (2)

where $V_k$ is the training node set with labels. $|F|$ is the dimension of the low-dimensional representation $Z$, which is equal to the number of categories. $Y$ is the real label confidence list with $Y_{lk} = 1$ if node $v_l$ belongs to category $c_k$ and $Y_{lk} = 0$ otherwise. $Z_{lk} = 1$ denotes the predicted probability that node $v_l$ belongs to $c_k$.

In the training process, the GCN model uses the classical gradient descent to optimize the parameters

$$W_{i}^{m+1} = W_{i}^{m} - \eta \frac{\partial L}{\partial W_{i}^{m}}$$  \hspace{1cm} (3)

where $\eta$ is the learning rate. During each iteration, the weights $W_i$ ($i \in \{0, 1\}$) are updated.

IV. GRAPHFOOL METHOD

Based on the graph embedding algorithms applied to the node classification task, Graphfool is designed to achieve a targeted label attack by adding or removing a few edges of the original graph.

A. Framework of Graphfool

In this section, we take GCN as the target graph embedding algorithm as an example to help us introduce the attack process of Graphfool. Graphfool consists of two parts, which are adversarial graph generation and adversarial attack. The framework of Graphfool is shown in Fig. 2.
Stage1: Adversarial graph generation

1) Adversarial Graph Generation: For the trained GCN model, we use an iterative linearization method to generate the minimum adjacency matrix perturbations that are sufficient to change the classification result of the attacked node. The final adversarial graph is obtained by mixing the perturbations and the original adjacency matrix.

2) Adversarial Attack: We use the generated adversarial example to mislead the classification results of the attacked node. Since GCN can accurately extract the hidden representation of nodes from the graph structure and node attributes, the perturbations generated by the GCN as the target classifier are universal, and the attack has strong transferability.

As shown in Fig. 2, for a graph dataset, we first derive the node classification results based on the adjacency matrix and the trained GCN model in the adversarial graph generation stage. Then, we calculate the classification boundaries \((f_1, f_2, f_3)\) for the attacked node \(v_{\text{tar}}\). We can also derive the minimum distance \(d_1, d_2, d_3\), where \(d_1 > d_3 > d_2\).

In untargeted label attack, we select the most vulnerable label whose classification boundary \((f_2)\) is closest to \(v_{\text{tar}}\). Then, the perturbation matrix is calculated, and the adversarial graph can be derived by adding/deleting the edges. For the targeted label attack, the only difference is that we should assign the target label of the attacked node in the targeted label attack. Finally, we use the adversarial graphs to test the attack effect of Graphfool using various network embedding algorithms in the adversarial attack stage.

Stage2: Adversarial attack

B. Adversarial Graph Generation

In Section III, we introduce the structure and training processing of a two-layered GCN. We take GCN as the target node classifier and generate adversarial graphs through the proposed Graphfool. The pseudocode of adversarial graph generation is given in Algorithm 1.

1) Perturbations of Adjacent Matrix: For a trained node classifier, it usually learns an \(|F|\)-dimensional low-dimensional representation from a given graph structure and node attributes, where \(|F|\) is the number of node categories. We can define the node classifier as \(f: R^C \rightarrow R^F\) (when the target classifier is GCN, \((1)\) can represent \(f\)). For node \(v_i\), we use the following mapping equation to describe a node classifier:

\[
\hat{k}_i(A) = \arg \max_k f_k(x_i, A) \tag{4}
\]

where \(\hat{k}\) denotes the predicted label of the node \(v_i\) in the classifier \(f\). \(x_i\) is the attribute vector of node \(v_i\), \(A\) is the graph adjacency matrix, and \(f_k(x_i, A)\) is the output of the \(k\)th category of \(f(x_i, A)\).

According to \((4)\), if we want to influence the node classification result of node \(v_i\), we can change \(x_i\) or \(A\). However, it should be noted that modifying edges affects all dimensions of node features during the aggregation [24], while modifying features only affects partial features, and the node features may be continuous, which is difficult to control the attack within the budge [57]. Besides, in realistic systems (e.g., transaction networks), limited by the permissions, it is easier to generate transaction records by transferring money than to tamper the attributes of the target user. For the above considerations, Graphfool conducts attacks by changing the adjacency matrix of the graph. Equation \((4)\) can be modified as

\[
\hat{k}(A) = \arg \max_k f_k(A) \tag{5}
\]

The optimization objective of most existing attack methods is to find the perturbations that have the maximum impact on the prediction confidence of the target node, which will cause excessive and unnecessary deviations in the prediction confidence. Correspondingly, these perturbations will occur between more dissimilar nodes, which makes them easier to detect by several defense methods [24], [34], [36]. Constructing the classification boundaries can help us find the perturbations that “just right” change the prediction confidence.
of the target instance, which may avoid unnecessary deviation of the prediction confidence.

a) Graphfool on a simplified linear classifier: Considering that the general node classifiers are mostly complex nonlinear classifiers, to help understand the attack process of Graphfool, we first simplify the problem and introduce how to find the classification boundary and the minimum perturbations for the linear classifier. We assume a linear classifier as \( f^i(A) = AW^i + b^i \), where \( W^i \in \mathbb{R}^{N \times F} \) and \( b^i \in \mathbb{R}^{N \times F} \) are trained classifier parameters. For the node \( \hat{v} \), the prediction of the classifier can be described as

\[
\forall k: Aw_k + b_k < Aw_{\hat{k}(A)} + b_{\hat{k}(A)}, \quad (k \neq \hat{k}(A)).
\]  

(6)

It means that for a well-trained classifier, the prediction probability corresponding to the true category of \( \hat{v} \) should be greater than the other categories. The attack goal is to make the target node misclassified by adding minimum perturbations \( \mathcal{P} \) on \( A \). Therefore, the attack problem can be modeled as

\[
\arg \min \| \mathcal{P} \|_2 \\
\text{s.t.} \exists k: (A + \mathcal{P})w_k + b_k \geq (A + \mathcal{P})w_{\hat{k}(A)} + b_{\hat{k}(A)}
\]  

(7)

where \( \mathcal{P} \) is the perturbation matrix, and its calculation process will be introduced later. \( w_k(w_{\hat{k}(A)}) \) and \( b_k(b_{\hat{k}(A)}) \) are the \( \hat{k}(A) \)th column of \( W^i \) and \( b^i \), respectively. Equation (7) indicates that when the well-designed perturbations are added to the input, the model will predict the example \( \hat{v} \) instead of \( \hat{v} \).

b) Graphfool on the general nonlinear classifiers: We now extend the Graphfool to the general case of nonlinear differentiable classifiers. Here, we approximate the node classification boundary function using the first-order Taylor expansion of each classifier

\[
f^i(A) - f^i_{\hat{k}(A)}(A) + A \cdot \nabla f^i_{\hat{k}(A)}(A) - A \cdot \nabla f^i_{\hat{k}(A)}(A) = 0 \\
\quad (k \neq \hat{k}(A)).
\]  

(11)

\( \mathcal{P} \) can be derived similarly using (10).

To guide the change of \( A \), we need to calculate the value of \( \mathcal{P} \). Since the adjacent matrix of an undirected graph is symmetric, we symmetrize \( \mathcal{P} \) to obtain \( \hat{\mathcal{P}} \) as shown in the following equation:

\[
\hat{\mathcal{P}}_{ij} = \hat{\mathcal{P}}_{ji} = \begin{cases} 
\mathcal{P}_{ij} + \mathcal{P}_{ji}, & i \neq j \\
0, & i = j.
\end{cases}
\]  

(12)

In (12), the elements in \( \hat{\mathcal{P}} \) have continuous values. For a specific element \( \hat{\mathcal{P}}_{ij} \) in \( \hat{\mathcal{P}} \), its positive/negative value indicates that we should add/delete the edge between the pair of nodes \( (v_i, v_j) \). The larger value of \( |\hat{\mathcal{P}}_{ij}| \) indicates that the added/deleted edge can influence the classification result of the target node more significantly.

2) Adversarial Graph Generator: In this section, we propose an adversarial graph generator based on our adjacent matrix perturbations \( \mathcal{P} \). We modify one edge during each iteration, and the generation process runs for \( K \) iterations.

1) Perturbation Matrix Calculation: During the \( h \)th iteration, we calculate the classification boundary closest to \( A^h \) (\( A^0 = A \)), its target label \( l^h \), and the perturbation matrix \( \hat{\mathcal{P}}^h \) by (8) and (10).

2) Perturbation Edges Selection: Based on \( \hat{\mathcal{P}}^h \), we select a node pair \( (v_i, v_j) \), which has maximum absolute value \( \hat{\mathcal{P}}_{ij}^h \) to add the perturbation. It is worth noting that if \( \hat{\mathcal{P}}_{ij}^h \) is positive/negative and \( (v_i, v_j) \) are connected/disconnected in \( A^h \), we cannot further add or delete the edge between this node pairs. Hence, we just ignore such node pairs.

3) Adversarial Graph Update: We modify the \( h \)th adjacency matrix with selected perturbation edges, thus generating a new adversarial graph \( A^{h+1} \). The update process can be expressed as

\[
A^{h+1} = A^h + \Psi(\hat{\mathcal{P}}_{ij}^h)
\]  

(13)

where \( A^{h+1} \) and \( A^h \) are the elements of \( A^{h+1} \) and \( A^h \), respectively. \( \Psi(\hat{\mathcal{P}}_{ij}^h) \) denotes the perturbation operation selected for the node pairs \( (v_i, v_j) \) in step 2), which can be adding edge, deleting edge, or keeping it unchanged.

C. Targeted Label Attack

In Section IV-A and IV-B, we introduce our technique to generate adversarial graph based on classification boundary and minimum perturbations of adjacency matrix. Moreover, Graphfool can also perform targeted label attack. Thus, we define a new goal, i.e., for the node \( v_i \), we want to misclassify it into a specific category \( l^t \neq \hat{k}(A_0) \). To distinguish it from the untargeted label attack, we still take the
Algorithm 1 Adversarial Graph Generation

**Input:** Original graph \( G = (A, X) \), number of iterations \( K \).

**Output:** The adversarial graph \( G’ = (A’, X) \).

1. Train the target graph embedding model \( f_W(\cdot) \) on original graph \( G \) to obtain model parameters \( W \) via Eq. 3.
2. Initialize the adjacency matrix of the adversarial graph by \( A^0 = A \);
3. for \( h = 0 \) to \( K - 1 \) do
   4. Construct \( \hat{P}^h \) based on \( A^h \);
   5. Select the perturbation edges which has maximum absolute value in \( \hat{P}^h \);
   6. Update the adjacency matrix \( A^{h+1} \) according to \( A^{h+1}_{ij} = A^h_{ij} + \Psi(\hat{P}^h) \);
9. Return the adversarial graph \( G’ \), with the adversarial adjacency matrix \( A’ \).

linear classifier as an example, and the attack problem can be generalized as

\[
\begin{align*}
\text{arg min } \|P\|_2 \\
\text{s.t. } \forall k(\hat{k}_i \neq l) : (A + P)w_l + b_l \geq (A + P)w_k + b_k.
\end{align*}
\]

In general, during iteration \( h \), when \( f^i(A^h) \neq f^i_{\hat{k}(A^h)}(A^h) \), Graphfool’s goal is to add the minimum perturbations \( P \) so that the node \( u_i \) can cross the classification boundary \( f^i(A) - f^i_{\hat{k}(A)}(A) = 0 \). The minimum perturbation during the \( h \)th iteration can be expressed as

\[
\begin{align*}
P^h = \frac{|f^i(A^h) - f^i_{\hat{k}(A^h)}(A^h)|(w_l - w_{\hat{k}(A^h)})}{\|w_l - w_{\hat{k}(A^h)}\|_2^2}.
\end{align*}
\]

D. Transfer Adversarial Attack

Besides the GCN model, we can also use the generated adversarial graphs to attack other node classification methods. Most node classification algorithms rely upon the connection relationship between nodes. Nodes with strong relationships are typically divided into the same category. Therefore, these algorithms have similar decision boundaries. In that case, the GCN-based adversarial attack can also be effective on many other node classification methods. In Section V-B, we use the adversarial graphs generated by Graphfool to attack other node classification algorithms. The experimental results show the strong transferability of our Graphfool.

V. EXPERIMENTAL RESULTS

To validate our Graphfool, we test it for both untargeted label attack and targeted label attack. Moreover, to demonstrate the stealthiness of Graphfool, we also perform the single-edge attack, the perturbation-limited attack, and the attack under the node similarity-based defense.

A. Experimental Setup

Here, we take the trained GCN as the target node classifier and generate adversarial graphs through Graphfool. We first divide each dataset into three parts: 20% as the training set, 40% as the validation set, and the remaining 40% as the test set to train the GCN. For each attacked node, we set the maximum number of attack iterations \( K \) to 20. In the iterative process, once the attacked node meets our attack requirements, we stop the attack process and output the adversarial graphs. The Adam optimizer is used to optimize the GCN, and the learning rate is searched in \([0.001, 0.1]\). The hidden layer dimension of GCN is set by a hyperparameter search in \( H \in \{16, 32, 64, 128, 256\} \).

In our experiments, the final learning rate of GCN is set to 0.01, and the hidden layer dimensions of the hidden layer are set to 64. We implement our proposed Graphfool with TensorFlow, and our experiments are performed on a machine with i7-7700K 3.5 GHz \times 8 \text{(CPU)}, TITAN Xp 12GiB \text{(GPU)}, 16 GB \times 4 memory \text{(DDR4)}, and Ubuntu 16.04 \text{(OS)}.

1) Datasets: In the experiment, we test different attack techniques on four datasets, which are Cora, Citeseer, Pubmed, and Pol.Blogs. Their statistics are provided in Table I.

2) Attack Success Rate (ASR): We attack a series of nodes in the graph, and the ASR is defined as follows.

\[
\text{ASR} = \frac{N_s}{N_{\text{att}}} \times 100\%
\]

where \( N_s \) is the number of successful attacked nodes and \( N_{\text{att}} \) is the total number of nodes being attacked.

2) Average Modified Link (AML): To attack the target node, we add/delete edges between nodes. The modifications should be minor and imperceptible. Thus, the method
with smaller AML is better. The AML is defined as

\[ \text{AML} = \frac{1}{N_{\text{att}}} \sum_{i=1}^{N_{\text{att}}} L_i \]  

(17)

where \( N_{\text{att}} \) is the total number of nodes being attacked and \( L_i \) is the number of modified edges for the node \( v_i \).

3) Baselines: To validate the attack performance of our Graphfool, we compare it with five graph embedding attack methods as shown below.

1) NETTACK [20]: It generates adversarial perturbations for graph structure and node attributes, and it ensures that the perturbations are imperceptible by preserving the degree distribution and attributes co-occurrence probability.

2) FGA [22]: It extracts the gradient of pairwise nodes based on the original graph. Then, it selects the node pairs with the maximum absolute edge gradient to update the adversarial graph.

3) RL-S2V [21]: It is a hierarchical reinforcement learning-based attack method. It learns a Q-function parameterized by S2V to perform a generalized attack.

4) GradArgmax [21]: It calculates the gradient of adjacency matrix based on the output of each hidden layer and the loss function. Then, it adopts a greedy algorithm to select the node pairs to attack the original graph.

5) IG-JSMA [24]: It solves the discreteness problem in graph-structured data by introducing integral gradient, which can accurately reflect the effect of perturbing certain features or edges.

### B. Attack Performance

In this section, we evaluate our Graphfool on four real-world datasets. The attack methods are tested with untargeted label attack, targeted label attack, single-edge attack, and perturbation-limited attack.

1) Untargeted Label Attack: First, we randomly select 20 nodes in each category to form the set of attacked nodes. To analyze the relationship between modified edges and attacked nodes, we consider direct, indirect, and unlimited attacks [22].

1) Direct Attack: This attack method only attacks the edges directly connected to the attacked node.

2) Indirect Attack: This attack method attacks the edges not directly connected to the attacked node.

3) Unlimited Attack: This attack method can remove or add edges between any node pairs. Without loss of generality, we assume that the number of modified edges is less than 20 for each attack. The attack results are shown in Table II. For the unlimited case, Graphfool outperforms the other attack methods in most of the cases, in terms of higher ASR and lower AML. For Cora, Citeseer, and Pubmed, our Graphfool can get 95.74% ASR and 6.73 AML. Since the Pol.Blogs network is denser than the other three datasets (the average degree of Pol.Blogs is close to 12.8, while the others are about 1–3), the attack performance of Graphfool has slightly deteriorated. However, it can still outperform other any other baselines.

For our Graphfool, the unlimited and direct attacks have similar attack performance. However, the indirect attack performs worse. This demonstrates that direct attacks are typically more effective than indirect ones. Here, we also conduct a transferring adversarial attack. We take the GCN as the target classifier to generate the adversarial graphs, which are used as label attack, targeted label attack, single-edge attack, and perturbation-limited attack.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
the input to attack other node classification algorithms. Table II shows the attacking results. Although the adversarial graphs are generated based on GCN, they can also achieve excellent attack performance for other node classification algorithms. This demonstrates the strong transferability of the adversarial graphs generated by Graphfool with the GCN as the target classifier. Due to the excellent classification performance of GCN, the node classification boundary it learns is largely similar to the ones learned by other graph embedding algorithms, which makes the generated perturbations more transferable. Moreover, our adversarial graphs achieve less AML on the GCN model, which indicates that the Graphfool can capture the vulnerability of GCN model more accurately.

For the datasets of Cora, Citeseer, and Pubmed, where the graphs are relatively sparse, indirect attacks can achieve higher attack performance. This implies that we may be able to change the edges far away from the target nodes to perform the attack. In other words, the local structure of these nodes is not necessarily destroyed, making the attack harder to detect. For the dataset of Pol.Blogs, since it is very dense, the performance of graphfool is limited, especially for indirect attack.

2) Targeted Label Attack: In this section, we perform targeted label attack for the Cora, Citeseer, and Pubmed datasets, which have more than two categories. For each dataset, we also randomly select 20 nodes in each category to form the set of attacked nodes. The specific attack strategy of Graphfool is the unlimited attack.

Fig. 3 shows the targeted label attack performance of Graphfool and other baselines. Compared with the ASR of the untargeted label attack, the targeted label performs worse. In addition, for different targeted labels, the attack performance of different attack methods is also different. We plot the classification boundaries of the original Cora, Citesser, and Pubmed datasets in Fig. 4 to make possible explanations for this phenomenon. Combining the results of targeted label attacks, we find that the difficulty of targeted label attack is often positively correlated with the area occupied by different categories. For example, in the Cora’s classification boundaries in Fig. 4(a), the third category of nodes occupies the largest area. Since we randomly select the set of attacked nodes, these nodes are more likely to be attacked into the third category with a larger category area under the perturbation within the same AML. We can draw the same conclusion from the Citesser and Pubmed datasets, which demonstrates that the difficulty of targeted label attack for different target labels is determined by the characteristics of the datasets.

By comparing the attack performance of the Graphfool and the baselines, we can draw a more intuitive conclusion that the adversarial perturbations generated based on the classification boundary can achieve a more effective attack with less AML in most cases. Compared with other baselines, the attack performance of FGA and IG-JSMA based on gradient information is closer to Graphfool, which verifies that gradient information is indeed an effective guide to generate adversarial perturbations. However, even the best-performing IG-JSMA with integrated gradient, its ASR is still about 5%–17% lower than Graphfool, which may be caused by a slight error in approximating the integrated gradient.
3) Single-Edge Attack: In the computer vision area, other than the ASR of adversarial attack, minimizing its perturbations is also an important goal [37]. Similarly, in graph-based attacks, if an attack method could get close performance with fewer modified edges, it has a better attack stealthiness at the perturbation budget level (budget ≡ AML = |A’ − A|). Here, we design a single-edge attack experiment for different datasets to evaluate the Graphfool’s stealthiness at the perturbation budget level. In this case, each attack method could only change one edge of the original graph to generate adversarial graphs. In other words, we set the AML of all attack methods to 1. The set of attacked nodes is the same as the ones in the experiment of untargeted label attacks. We also experiment with direct, indirect, and unlimited attacks.

The attack performance of single-edge attack is shown in Table III. In general, single-edge attack is a special case of untargeted label attack. Therefore, the results in Table III are consistent with those in Table II. For the unlimited and direct cases, Graphfool and direct Graphfool achieve approximately 50% ASR, but in Pol.Blogs, they only get 18.16% ASR. This is also caused by the denseness of the dataset. It is difficult to have a sufficient impact on Pol.Blogs by only one edge modified.

In addition, the ASR of Graphfool, FGA, NETTECK, and IG-JSMA in the GCN model is higher than those in other node classification algorithms. The reason may be twofold. On the one hand, attack methods, such as Graphfool, FGA, NETTECK, and IG-JSMA, are GCN-based graph attack methods, so they have more direct attack impact when attacking GCN. On the other hand, for other node classification algorithms, they all have certain randomness. The single-edge attack only changes one edge in the original graph. It may affect more significantly during the random process, thus reducing the ASR for these algorithms.

4) Perturbation-Limited Attack: To improve the attacks’ stealthiness, the perturbations can be limited to a certain scale, which is called a perturbation-limited attack. In perturbation-limited attack, attackers only change the edges of the subgraph, which is composed of the attacked node and its neighbors. For each attacked node, we first calculate its \( k \)-order neighbors (the nodes whose link distance to the attacked node are less than \( k \)) and construct the subgraph accordingly. Then, we perform a graph attack in this subgraph using an unlimited Graphfool attack. The value of \( k \) is corresponding to the size of the modifiable subgraph. Here, we set the attack scale \( k \) from 1 to 5. To quantify the size of these subgraphs in the original graph, for each dataset, we calculate the average ratio of the nodes’ number in each \( k \)th order subgraph to the corresponding number in the original graph. The results are shown in Table IV.
while AML is monotonously decreasing. This is consistent with the general idea that when the size of the constructed subgraph gets larger, the Graphfool attack is more likely to succeed. Moreover, for the sparser dataset Cora, Citeseer, and Pubmed, when $k = 5$, the average sizes of subgraphs are only 14.22%, 2.87%, and 13.59% of the original graphs, respectively, while Graphfool can still achieve average ASR of 67.69%, 49.18%, and 65.82% under the AML of no more than 10, respectively. This implies that even if we limit the perturbation to a small local subgraph of the attacked node, Graphfool can still perform an effective attack. However, for the dense graph Pol.Blogs, the average size of subgraphs becomes 81.88% when $k = 5$. It covers most of the original graph, and the results are close to the unlimited case in Table II.

C. Attack Performance Under the Possible Defense

In Section V-B3, we considered the stealthiness of the perturbations at the perturbation budget level and verified the effectiveness of Graphfool when only one edge is allowed to be modified. However, several defense works [24], [34], [36] indicate that the adversarial perturbations that only limited by AML can still be detected at the node similarity level. To further verify the importance of generating perturbations that have the minimum impact on prediction confidence, we investigate the attack performance of different attack methods under the possible node similarity-based defense. Specifically, we first utilize the unlimited attack strategy to generate the adversarial graphs for untargeted attacks. Then, we learn the nodes’ low-dimensional representations from GCN and calculate the cosine similarity of each pair of nodes in these adversarial graphs. Finally, the edges that connect the node pairs whose node similarity is lower than the set similarity threshold will be deleted, and the defensive graphs are input into a trained GCN to verify their attack performance under the node similarity-based defense.

Fig. 5 shows the attack performance of different attack methods on GCN under different similarity thresholds. Compared with other attack methods, Graphfool can achieve the highest ASR under the same similarity threshold. Combined with Fig. 1(a), for the three attack methods of Graphfool, FGA, and NETTACK, the perturbations generated by FGA have the lowest node similarity. Therefore, in Fig. 5(a), as the similarity threshold increases, the attack performance of FGA first decreases rapidly when the similarity threshold reaches 0.3, while Graphfool starts to decline rapidly after the similarity threshold reaches 0.6, whose perturbed nodes have the highest similarity. This indicates that it is indeed necessary to consider the stealthiness of the perturbations at the node similarity level. Graphfool can generate perturbations between more similar nodes, thereby enhancing the stealthiness of the adversarial perturbations.

In addition, compared to the Cora, Citeseer, and Pubmed datasets, adversarial perturbations on the denser graph Pol.Blogs are more likely to be detected by defense methods with a similarity threshold lower than 0.4. It is because of the obvious differences between the two categories of nodes in Pol.Blogs. Most nodes can only achieve effective attacks by connecting very dissimilar nodes, which makes the perturbations easier to detect. However, in general, the adversarial perturbations generated by Graphfool can better resist the negative impact of the node similarity-based defense.
VI. CONCLUSION

A. Contributions

In this work, we have proposed a targeted label adversarial attack on graph embedding methods to generate effective perturbations that have the minimum impact on the prediction confidence, namely, Graphfool. It achieves precise targeted label attack with more imperceptible perturbations at the node similarity level. Graphfool can take any differentiable graph-based classifier as the attack model and construct the decision boundary according to its classification results, thus generating the minimum distance perturbations that allow the target instance to cross the specified classification boundary. Extensive experiments demonstrate that Graphfool achieves the state-of-the-art attack performance in a stealthier manner at node similarity level. Besides, possible defense experiments further prove that the perturbations generated by Graphfool are more imperceptible than baselines.

B. Limitations

First, although Graphfool achieves the state-of-the-art attack performance even with the node similarity-based defense mechanism, the perturbations can still be defended by the defense mechanism with a high similarity threshold. Second, besides, Graphfool needs to calculate the boundary distance repeatedly, which requires more time.

C. Future Works

For future works, we will work on finding the loopholes of similarity-based defense methods and improve the ability of attack methods against them. In addition, the efficiency of the attack will also be taken into consideration, especially on large-scale datasets, which will reduce the cost of the attacker significantly.

REFERENCES

[1] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca, “Network analysis in the social sciences,” Science, vol. 333, no. 5991, pp. 892–895, 2009.
[2] V. Latora and M. Marchiori, “Is the Boston subway a small-world network?” Phys. A, Stat. Mech. Appl., vol. 314, nos. 1–4, pp. 109–113, Nov. 2002.
[3] M. Kistler, M. Perrone, and F. Petrini, “Cell multiprocessor communication network: Built for speed,” IEEE Micro, vol. 26, no. 3, pp. 10–23, Sep. 2006.
[4] J. M. Montoya and R. V. Solé, “Small world patterns in food webs,” J. Theor. Biol., vol. 214, no. 3, pp. 405–412, Feb. 2002.
[5] R. Hong, Y. He, L. Wu, Y. Ge, and X. Wu, “Deep attributed network embedding by preserving structure and attribute information,” IEEE Trans. Syst., Man, Cybern. Syst., vol. 51, no. 3, pp. 1434–1445, Mar. 2021.
[6] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2014, pp. 701–710.
[7] S. Wang, J. Tang, C. Aggarwal, Y. Chang, and H. Liu, “Signed network embedding in social media,” in SDM, 2017.
[8] J. Tang, M. Qu, and Q. Mei, “PTE: Predictive text embedding through large-scale heterogeneous text networks,” in Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2015, pp. 1165–1174.
[9] S. Wang, J. Tang, C. Aggarwal, and H. Liu, “Linked document embedding for classification,” in Proc. 25th ACM Int. Conf. Inf. Knowl. Manage., 2016, pp. 115–124.
[10] F. Tian, B. Gao, Y. Cui, E. Chen, and T. Y. Liu, “Learning deep representations for graph clustering,” in Proc. 28th AAAI Conf. Artif. Intell., 2014, pp. 1293–1299.
[11] K. Allen, L. Labiod, and M. Nadif, “A semi-NMF-PCA unified framework for data clustering,” IEEE Trans. Knowl. Data Eng., vol. 29, no. 1, pp. 2–16, Jan. 2016.
[12] J. B. Tenenbaum, V. de Silva, and J. C. Langford, “A global geometric framework for nonlinear dimensionality reduction,” Science, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.
[13] M. Belkin and P. Niyogi, “Laplacian Eigenmaps and spectral techniques for embedding and clustering,” in Proc. Adv. Neural Inf. Process. Syst., 2002, pp. 585–591.
[14] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” Science, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.
[15] P. Goyal and E. Ferrara, “Graph embedding techniques, applications, and performance: A survey,” Knowl.-Based Syst., vol. 151, pp. 78–94, Jul. 2018.
[16] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond Euclidean data,” IEEE Signal Process. Mag., vol. 34, no. 4, pp. 18–42, Jul. 2017.
[17] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Represent., Toulon, France, 2017, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl&noteId=SU4ayYgl
[18] P. W. Battaglia et al., “Relational inductive biases, deep learning, and graph networks,” 2018, arXiv:1806.01261.
[19] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” 2016, arXiv:1611.07308.
[20] D. Zügner, A. Akbarnejad, and S. Günnemann, “Adversarial attacks on neural networks for graph data,” in Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2018, pp. 2847–2856.
[21] H. Dai et al., “Adversarial attack on graph structured data,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 1115–1124.
[22] J. Chen, Y. Wu, X. Xu, Y. Chen, H. Zheng, and Q. Xuan, “Fast gradient attack on network embedding,” 2018, arXiv:1809.02797.
[23] X. Wang, M. Cheng, J. Eaton, C.-J. Hsieh, and F. Wu, “Attack graph convolutional networks by adding fake nodes,” 2018, arXiv:1810.10751.
[24] H. Wu, C. Wang, Y. Tsytsiuk, A. Docherty, K. Lu, and L. Zhu, “Adversarial examples for graph data: Deep insights into attack and defense,” in Proc. 28th Int. Joint Conf. Artif. Intell., Aug. 2019, pp. 4816–4823.
[25] Y. Sun, S. Wang, X. Tang, T.-Y. Hsieh, and V. Honavar, “Node injection attacks on graphs via reinforcement learning,” 2019, arXiv:1909.06543.
[26] T. Takahashi, “Indirect adversarial attacks via poisoning neighbors for graph convolutional networks,” in Proc. IEEE Int. Conf. Big Data (Big Data), Dec. 2019, pp. 1395–1400.
[27] H. Chang et al., “A restricted black-box adversarial framework towards attacking graph embedding models,” in Proc. AAAI Conf. Artif. Intell., vol. 34, no. 4, 2020, pp. 3389–3396.
[28] J. Chen et al., “GA-based Q-attack on community detection,” IEEE Trans. Comput. Soc. Syst., vol. 6, no. 3, pp. 491–503, Jun. 2019.
[29] D. Zügner and S. Günnemann, “Adversarial attacks on graph neural networks via meta learning,” 2019, arXiv:1902.08412.
[30] A. Bojchevski and S. Günnemann, “Certifiable robustness to graph perturbations,” in Proc. 33rd Int. Conf. Neural Inf. Process. Syst., 2019, pp. 8319–8330.
[31] F. Feng, X. He, J. Tang, and T.-S. Chua, “Graph adversarial training: Dynamically regularizing based on graph structure,” IEEE Trans. Knowl. Data Eng., vol. 33, no. 6, pp. 2493–2504, Jun. 2021.
[32] Y. Zhu, S. Zhang, S. Khan, and M. Coates, “Comparing and detecting adversarial attacks for graph deep learning,” in Proc. Represent. Learn. Graphs Manifolds Workshop, Int. Conf. Learn. Represent., New Orleans, LA, USA, 2019, pp. 1–7.
[33] J. Chen, X. Lin, H. Xiong, Y. Wu, H. Zheng, and Q. Xuan, “Smoothing adversarial training for GNN,” IEEE Trans. Computat. Social Syst., vol. 8, no. 3, pp. 618–629, Jun. 2021.
[34] Y. Zhang and M. Zitnik, “GnnGuard: Defending graph neural networks against adversarial attacks,” in Proc. Adv. Neural Inf. Process. Syst., 2023, vol. 36, pp. 9263–9275.
[35] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, “All you need is low (Rank): Defending against adversarial attacks,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 169–177.
[36] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, “Graph structure learning for robust graph neural networks,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2020, pp. 66–74.
A. J. Bose, A. Cianflone, and W. L. Hamilton, “Generalizable adversarial nets,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, no. 1, 2018, pp. 1–8.

Jinjin Chen received the B.S. and Ph.D. degrees from the Zhejiang University of Technology, Hangzhou, China, in 2004 and 2009, respectively. She studied evolutionary computing at the Ashikaga Institute of Technology, Ashikaga, Japan, from 2005 to 2006. She is currently a Professor with the Zhejiang University of Technology. Her research interests include artificial intelligence security, graph data mining, and evolutionary computing.

Guohan Huang is currently pursuing the master’s degree with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His research interests include graph data mining and applications, and artificial intelligence.

Haibin Zheng received the bachelor’s degree from the Zhejiang University of Technology, Hangzhou, China, in 2017, where he is currently pursuing the Ph.D. degree with the College of Information Engineering. His research interests include deep learning, artificial intelligence, and adversarial attack and defense.

Dunjie Zhang is currently pursuing the master’s degree with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His research interests include data mining and applications, and intelligent computing.

Xiang Lin is currently pursuing the master’s degree with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His research interests are intelligent computation and complex networks.