Adversarial Attack on Large Scale Graph

Jintang Li, Tao Xie, Liang Chen*, Fenfang Xie, Xiangnan He, Zibin Zheng

Abstract—Recent studies have shown that graph neural networks (GNNs) are vulnerable against perturbations due to lack of robustness and can therefore be easily fooled. Currently, most works on attacking GNNs are mainly using gradient information to guide the attack and achieve outstanding performance. However, the high complexity of time and space makes them unmanageable for large scale graphs and becomes the major bottleneck that prevents the practical usage. We argue that the main reason is that they have to use the whole graph for attacks, resulting in the increasing time and space complexity as the data scale grows. In this work, we propose an efficient **Simplified Gradient-based Attack (SGA)** method to bridge this gap. SGA can cause the GNNs to misclassify specific target nodes through a multi-stage attack framework, which needs only a much smaller subgraph. In addition, we present a practical metric named **Degree Assortativity Change (DAC)** to measure the impacts of adversarial attacks on graph data. We evaluate our attack method on four real-world graph networks by attacking several commonly used GNNs. The experimental results demonstrate that SGA can achieve significant time and memory efficiency improvements while maintaining competitive attack performance compared to state-of-art attack techniques.

Index Terms—Node classification, Adversarial attack, Graph neural networks, Efficient attack, Network robustness

1 Introduction

Recently, with the enormous advancement of deep learning, many domains like speech recognition [1] and visual object recognition [2], have achieved a dramatic improvement out of the state-of-the-art methods. Despite the great success, deep learning models have been proved vulnerable against perturbations. Specifically, Szegedy et al. [3] and Goodfellow et al. [4] have found that deep learning models may be easily fooled when a small perturbation (usually unnoticeable for humans) is applied to the images. The perturbed examples are also termed as "adversarial examples" [4].

Graph structures are ubiquitous in nature and society, there is a great deal of research interest in studying graph data [5], [6], [7], [8], [9]. Undoubtedly, graph plays a crucial role in many high impact applications in the world [8], [10], [11]. Therefore, the importance of the robustness of deep learning models on graph data should not be emphasized too much. However, the adversarial examples do have a significant effect on graph data, which is still a major obstacle to be overcome. So far, much of the current work on attacking graph neural networks has focused on the node classification task [12], [13], [14], [15]. In order to fool a classifier and misclassify specific nodes, attackers may use two attack strategies to achieve the adversarial goals: *direct attack* and *influence attack*. For the direct attack, perturbations on the target node are allowed while the influence attack is limited to its neighboring nodes or beyond [12]. Figure 1 demonstrates a toy example of how deep learning models are fooled by attackers with small perturbations on the graph structure. In this case, the influence attack strategy is adopted.

In this paper, we focus on the *targeted attack* that aims to make a specific node (e.g., a person in social networks [16]) misclassified. In this scenario, Dai et al. [13] study the adversarial attack on graph structure data and propose a gradient-based method, namely GradArgmax, which modifies links based on gradients information of a surrogate model so as to fool the classifiers. In addition, Zügner et al. [12] propose Nettack, which is able to perturb the graph structure as well as node features. Despite great success, there are still some challenges for the attackers.

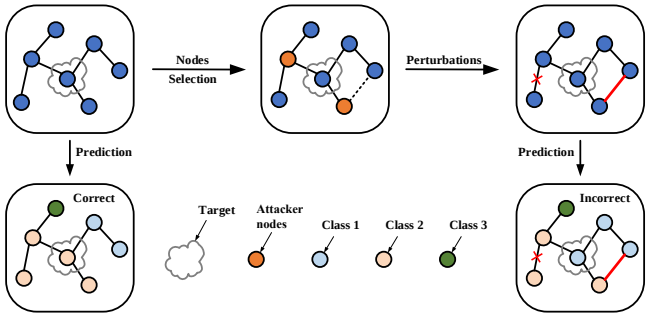


Fig. 1: Adversarial attacks on the graph structure. Attackers tend to flip the edges of attacker nodes and lead to the misclassification of the target node.

Challenges: (i) *Scalability*. Most attacks have to store unnecessary graph information and thus suffer from the rising time and memory costs as the data scale increases. Naturally, they fail to efficiently conduct attacks on a larger graph. However, multi-million-scale graph networks are common in the real-world, so existing methods need to be improved and scaled to larger graphs. (ii) *Evaluation*. A further challenge is to quantify the effects of adversarial attacks on graph data. As the graph data is unable and meaningless to be converted to continuous form, the impacts on the graph data are unsuitable to be measured with ℓ_2-norm or ℓ_∞-norm [17], which is different from that on image data. This makes the evaluation of the attack impacts a difficult problem to solve.

In this work, we attempt to tackle these challenges by our proposed methods. Specifically, our methods include two parts: (i) *SGA framework*. We argue that it is unnecessary to use the whole graph to attack since attackers simply focus on misclassifying several nodes (the target nodes). Besides, due to the lack of time and space efficiency, Graph Convolution Network (GCN) [8] is unsuitable as a surrogate model despite being used frequently

*Corresponding author

in previous works. Inspired by Simplified Graph Convolutional Network (SGC) [18] and gradient-based attack methods [13], [15], we propose a novel **Simplified Gradient-based Attack (SGA)** framework for effective and efficient adversarial attacks. SGA only needs a much smaller subgraph consisting of k-hop neighbors of the target node (k depends on the number of SGC layers and typically set to 2), and sequentially flips edges with the largest magnitude of gradients in this subgraph by utilizing the surrogate model SGC. In addition, we introduce a scale factor to cope with the gradient vanishing during attacks (See Section 4.3). Notably, our experimental evaluation suggests that the simplifications can hardly affect the attack performance. Moreover, the resulting model can remarkably improve the time and space efficiency — even yields up to 1,976 and 5,753 times speedup over Nettack on Pubmed dataset in the direct attack and influence attack settings, respectively. Naturally, SGA can scale to very large graph networks easily. (ii) *Degree Assortativity Change (DAC)*. Previous studies focus primarily on the intrinsic properties of graph [19], [20], but measuring the impact of graph adversarial attacks is however left unexplored so far. To address this problem, we also propose a practical metric named **Degree Assortativity Change (DAC)**, which can measure the intensity of the impact after perturbations being performed.

The main contributions of our works are summarized as follows:

- We propose a novel adversarial attack framework SGA, which extracts a much smaller subgraph centered at the target node, thereby addressing the difficulty of conducting attacks on a large scale graph.
- We notice the problem of gradient vanishing for gradient-based attack methods and solve it by introducing a scale factor to calibrate the model.
- We emphasize the importance of *unnoticeability* for adversarial attacks on graph data and propose, first of all, a practical metric DAC to measure the attack impacts conveniently. This work can also be further developed in the graph domains.
- We conduct extensive experiments on four datasets by attacking several widely adopted graph neural networks. The experimental results show that our attack method has achieved significant improvements in both time and space efficiency while maintaining a significant attack performance compared to other state-of-the-art attack methods.

2 Related Work

Our work includes two parts: adversarial attack on graph data and evaluation of the attack impacts. In this section, we begin with the introduction of previous works of adversarial attacks on graph data and then discuss the details of ensuring the unnoticeability of adversarial attacks.

2.1 Adversarial Attacks on Graph Data

The robustness and security of deep learning models have received widespread attention, there has been a surge of interests in the adversarial attacks [3], [4], [21], [22], [23]. The obtained results suggest that deep learning models are vulnerable to adversarial examples and could be easily fooled by them even under restricted black-box attack scenarios (attackers conduct attacks without any prior knowledge about the target model [24]).

While previous works focus mostly on non-graph structure data (e.g., images and texts), adversarial attacks on graph structure data are considerably less studied because of the discreteness, which means that attackers must limit the attacks in order to maintain the graph property and allow only a few edges to be modified in the graph. To conduct a practical black-box attack on graph data with limited knowledge, attackers often train a surrogate model locally to generate adversarial examples (perturbed graphs) and transfer them to fool the target models.

To be specific, we focus on the task of node classification in this work. By referring to [24], we divide current approaches of adversarial attacks on the graph into two categories:

2.1.1 Gradient-based attack

This is the most commonly used method. Gradients have been successfully used to perform attacks in other domains [3], [25]. Since most of the existing models are optimized with gradient, along or against the direction of gradients is an efficient way to generate destructive adversarial examples. Focusing on the targeted attack, Dai et al. [13] propose GradArgmax, which extracts gradients of the surrogate model and deletes edges with the largest magnitude of the gradient to generate adversarial examples. However, the attacks are restricted to edge deletion only because the whole graph is stored with a sparse matrix (results in the lack of gradients information of non-edges). For the non-targeted attack, Zügner et al. [15] utilize the meta-gradients to solve the bi-level problem underlying the challenge of poisoning attacks (a.k.a, training-time attacks). Similarly, Xu et al. [26] propose PGD structure attack that conducts gradient attacks from a perspective of first-order optimization. Wang et al. [27] introduce the approximate fast gradient sign method, which generates adversarial examples based on approximated gradients.

2.1.2 Non-gradient-based attack

As well as methods focused on gradients, attackers prefer to explore other heuristic algorithms to conduct attacks. Waniek et al. [28] propose "Disconnect Internally, Connect Externally" (DICE), conducting attacks by dropping edges between nodes with high correlations and connecting edges with low correlations. Moreover, Zügner et al. [12] study both poisoning attacks and evasion attacks (a.k.a test-time attack) and further propose Nettack based on a linear GCN model, which maximizes the misclassification loss of the surrogate model greedily and perturbs the graph structure and node features to fool the classifiers. Zhang et al. [29] introduce cross-entropy attack strategy based on the Deep Graph Infomax [30] model. Chang et al. [31] formulate the graph embedding method as a general graph signal process with corresponding graph filter and propose a generalized attacker under restrict black-box attack scenario.

2.2 Unnoticeability of Adversarial Attacks

Typically, in an adversarial attack scenario, attackers not only focus on the attack performance but also seek to be concealed to avoid detection. However, previous works usually conduct attacks under a fixed budget and thought it would be unnoticeable as if the budget is small enough. We argue that it is not sufficient to preserve the properties of graphs and ensure unnoticeability in most cases.

To bridge this gap, Zügner et al. [12] enforce the perturbations to ensure its *unnoticeability* by preserving the graph's degree distributions and feature co-occurrences, restricting them to be marginally modified before and after attacks. However, there is

still no practical metric to measure the impact of attacks on graph data.

On the other hand, researches on the graph (or network) structure have yielded several results with respect to certain important properties, including "small-world effect" [32], [33] and degree distributions [34], [35]. Unlike previous works, Newman et al. [19] and Foster et al. [20] focus on another important network feature, i.e., *assortativity*, and propose a number of measures of assortative mixing appropriate to various mixing types. Generally speaking, assortativity can be defined as the tendency of nodes to connect to each other, it is generally viewed as a metric to probe the properties of a specific graph, and also reflects the graph structure. Despite its popularity in network analysis and worthy of further study, it has not yet been used to measure the impacts of such adversarial attacks on graphs. Therefore, we aim to bridge the gap and apply it to graph adversarial learning.

TABLE 1: Frequently used notations in this paper.

Notations	Descriptions
G	Graph representation of the data
V	Set of vertices in the graph
E	Set of edges in the graph
$\mathcal{C}$	Set of class labels of nodes
N, C, F	Number of nodes, classes and feature dimensions
A	Adjacency matrix of the graph, $N \times N$
X	Feature matrix of the nodes, $N \times F$
D	Diagonal matrix of the degree of each vertex, $N \times N$
f_θ	Graph neural networks model
W	Trainable weight matrix, $F_l \times F_{l+1}$
Z	Prediction probability of nodes, $N \times C$
t	Target node to attack
ϵ	Scale factor to calibrate the model
$\mathcal{D}(u, v)$	The shortest distance between u and v
$\mathcal{N}(t)$	Set of nodes adjacent to node t
k	Radius of the subgraph
$\mathcal{A}$	Set of attacker nodes, $\mathcal{A} = \{t\}$ for direct attack, $\mathcal{A} = \mathcal{N}(t)$ for influence attack
M	Degree mixing matrix
r	Degree assortativity coefficient
Δ	Attack budget
G', A', X'	Perturbed graph, adjacency matrix and feature matrix
$\tilde{\mathcal{L}}_t$, $\tilde{\mathcal{L}}_t^{(sub)}$	Targeted misclassification loss on the graph and subgraph
S	Structure score matrix
$G^{(sub)}$	The k-hop subgraph
$V^{(sub)}$, $E^{(sub)}$	Set of nodes, edges in the subgraph
$A^{(sub)}$	Adjacency matrix of the subgraph
$E^{(exp)}$	Set of expanded edges in the subgraph

3 PRELIMINARY

Before presenting our proposed methods, we will first give the notations of graph data formally, and then introduce the surrogate models — the GCN family, finally clarify the details of other proposed adversarial attack methods. See Table 1 for frequently used notations.

3.1 Notations

Specifically, we focus on the task of semi-supervised node classification in a single, undirected, attributed graph. Formally, we define $G = (A, X)$ as an attributed graph, where $A \in \{0,1\}^{N\times N}$ is a symmetric adjacency matrix denoting the connections of the N nodes, and $X \in \{0,1\}^{N\times F}$ or $X \in \mathbb{R}^{N\times F}$ represent the binary or continuous node features with F dimension. We use V to denote the set of nodes and $E \subseteq V \times V$ the *connected* edges of the graph G; $\mathcal{C} = \{c_i\}$ denotes a set of class labels where c_i indicates the ground-truth label of node i, and we define C as the number of classes in $\mathcal{C}$.

3.2 Graph Convolution Network Family

3.2.1 Vanilla Graph Convolution Network (GCN)

Since a number of existing works [12], [13], [15], [26] use vanilla GCN [8] as a surrogate model to conduct attacks, thus we first introduce GCN and further draw attention on SGC [18] — a simplified variant of GCN. Refer to this work [8], GCN is recursively defined as

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}), \tag{1}$$

where $\tilde{A} = A + I_N$ is the adjacency matrix with self-loops. I_N is a N by N identity matrix, and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is a diagonal degree matrix. $W^{(l)} \in \mathbb{R}^{F_l \times F_{l+1}}$ is a trainable input-to-hidden weight matrix and $H^{(l)} \in \mathbb{R}^{N\times F_l}$ is the matrix of hidden representation (activation), both of which are related to the l^{th} layer. Particularly, $F_0 = F$, $H^{(0)} = X$ is the input of neural network. $\sigma(\cdot)$ represents the element-wise activation function of network and is usually defined as $\mathrm{ReLU}(\cdot) = \max(\cdot, 0)$.

For node classification task, consider GCN with one hidden layer, and let $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, then the forward of GCN with pre-processing step could be taken as

$$Z = f_\theta(A, X) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right), \tag{2}$$

where $Z \in \mathbb{R}^{N\times C}$ is the output matrix of GCN and indicates the prediction probability of nodes belonging to different classes. The softmax activation function, defined as $\mathrm{softmax}(x_i) = \frac{1}{z}\exp(x_i)$ with $z = \sum_j \exp(x_j)$, is applied row-wise.

Given a set of labeled nodes $V_L \subseteq V$ with ground-truth labels $\mathcal{C}_L \subseteq \mathcal{C}$, the goal of GCN is to learn a mapping function $g : V \to \mathcal{C}$ by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta; A, X) = -\sum_{i\in V_L} \ln Z_{i,c_i}, \quad Z = f_\theta(A, X), \tag{3}$$

where $c_i \in \mathcal{C}_L$ is the class label of node i and $\theta = \{W^{(0)}, W^{(1)}\}$ denotes the trainable weights of model. After being optimized with Gradient Descent [36], the weights are learned to predict nodes in an unlabeled set $V_U = V - V_L$.

3.2.2 Simplified Graph Convolutional Network (SGC)

Linearization. The drawback of vanilla GCN is the excessive computational cost of message aggregation between nodes and their neighboring nodes, which is repeatedly and unnecessarily computed during training. To address this problem, Wu et al. [18] theoretically analyze the structure of GCN and further propose a linear variant, namely SGC. SGC replaces the nonlinear activation function $\mathrm{ReLU}(\cdot)$ with identity function and collapses weight matrices between consecutive layers. In this way, the forward of GCN can be simplified as

$$\begin{aligned} Z = f_\theta(A, X) &= \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \\ &= \mathrm{softmax}\left(\hat{A}\hat{A}XW^{(0)}\ W^{(1)}\right) \\ &= \mathrm{softmax}\left(\hat{A}^2XW\right), \end{aligned} \tag{4}$$

where $W = W^{(0)}\ W^{(1)}$ is a collapsed weight matrix. With pre-computing $\hat{S} = \hat{A}^2X$, the complicated GCN structure can be simplified as an input-to-output fully-connected neural network [37] without any hidden units, thereby avoiding redundant computation and greatly reducing training time and memory.

3.3 SOTA Adversarial Targeted Attack Methods

For adversarial attacks on graphs, attackers may conduct *structure attack* or *feature attack* [24] to modify the graph structure or node features, respectively. Following the work [12], we assume that attackers have prior knowledge about the graph structure and node features, including ground-truth labels of nodes. Beyond that, attackers are not allowed to access any additional information about the target models, neither model architecture nor parameters. In this case, attackers often train a transferable surrogate model locally, perform perturbations to fool it to achieve the best result of misclassification, and eventually transfer to other target models. Since we focus on the targeted attack in node classification task, here we briefly introduce other proposed state-of-the-art targeted attack methods: **Nettack** [12] and **GradArgmax** [13].

3.3.1 Nettack

Similar to SGC, Nettack uses a linear variant of two-layer GCN as a surrogate model to conduct targeted attack. Given a target node t and a budget $\Delta \in \mathbb{N}$, Nettack modifies the graph structure and node features aiming to maximize the misclassification loss of the surrogate model:

$$\begin{aligned} &\arg\max_{A',X'}(\max_{c'_t \neq c_t} \ln Z_{t,c'_t} - \ln Z_{t,c_t}), \quad t \in V \\ &s.t. \sum_i \sum_j |X_{i,j} - X'_{i,j}| + \sum_{u<v} |A_{u,v} - A'_{u,v}| \leq \Delta \end{aligned} \tag{5}$$

where $Z = f_\theta(A', X')$ is the output of the surrogate model, A' and X' are the perturbed adjacency matrix and feature matrix, respectively.

3.3.2 GradArgmax

GradArgmax uses vanilla GCN as a surrogate model to maximize the cross-entropy loss described in Eq.(3) by gradient ascent. As the loss function $\mathcal{L}$ and target node t are specified, GradArgmax computes the partial derivative of $\mathcal{L}_t$ with respect to each *connected* edge of the adjacency matrix:

$$\nabla_G = \nabla_A \mathcal{L}_t = \frac{\partial \mathcal{L}_t}{\partial A}, \tag{6}$$

where $\mathcal{L}_t$ denotes the targeted loss w.r.t node t.

To preserve the discreteness of adjacency matrix A, GradArgmax greedily modifies those edges with Δ-largest magnitude of gradients. In addition, the adjacency matrix is stored as a sparse one in order to avoid excessive computational costs, but only gradients of the *connected* edges are computed, which means that attacks are restricted to the edge deletion only, and the information on surrogate gradients is not fully utilized.

4 SIMPLIFIED GRADIENT-BASED ATTACK

In this work, we simply focus on the structure attack. We follow the attack settings of Nettack [12] which modifies graph structure and node features by flipping them from 0 to 1 or vice verse. Since node features in the real-world dataset may be continuous (not binary), it is difficult to constrain attacks within a given budget $\Delta \in \mathbb{N}$. Note that, our method can be easily extended to the feature attack (either binary or continuous features) as well, just by taking into account the gradients of the input features. In the node classification scenario detailed in Section 3.1, the goal of an attacker is to perturb the original graph $G = (A, X)$ with limited budgets Δ and further lead to a misclassification of target models.

To conduct attacks on the target models without additional prior knowledge, inspired by the adversarial methods mentioned above, the proposed SGA follows four steps: (i) Train a surrogate model locally (Section 4.1). (ii) Extract a k-hop subgraph (Section 4.2). (iii) Compute surrogate gradients via subgraph (Section 4.3). (iv) Choose to add or remove edges iteratively based on gradients (Section 4.4).

4.1 Surrogate Model

The most widely used surrogate model is vanilla GCN, however, it has significant drawbacks as described in Section 3.2. We use a k-layer SGC as our surrogate model instead. SGC removes the nonlinear transition functions (e.g., ReLU) between each layer and only keep the final softmax, hence the output of a k-layer SGC can be formulated as

$$\begin{aligned} Z = f_\theta(A, X) &= \mathrm{softmax}(\hat{A} \dots \hat{A}\hat{A}XW^{(0)}W^{(1)} \dots W^{(k-1)}) \\ &= \mathrm{softmax}(\hat{A}^k XW) \end{aligned} \tag{7}$$

where $W = W^{(0)}W^{(1)} \dots W^{(k-1)}$ is the collapsed weight matrix.

First and foremost, we train SGC on the input graph until convergence with fine-tuned hyper-parameters. In fact, the goal is to obtain the weight matrix $\theta = \{W\}$, which will be used later to guide the attack in our method.

4.2 k-hop Subgraph

Given a target node t with ground-truth class label c_t, the goal of attackers is to get it classified as another class c' where $c' \neq c_t$. Therefore, we design the targeted misclassification loss as follows:

$$\tilde{\mathcal{L}}_t = \max_{c' \neq c_t} \ln Z_{t,c'} - \ln Z_{t,c_t}, \quad t \in V\,, \tag{8}$$

where Z_{t,c_t} indicates the predicted probability that node t belongs to class c_t. According to Eq.(8), to compute the targeted loss $\tilde{\mathcal{L}}_t$, we only need to compute Z_t, a row-vector of Z. Therefore, computing $\{Z_u \,|\, u \in V \text{ and } u \neq t\}$ is unnecessary and redundant. This motivates us to simplify the computation.

Property 1. Given a normalized adjacency matrix $\hat{A}$ with self-loops, i.e., $\hat{A}_{u,u} \neq 0, \forall u \in V$. let $\mathcal{D}(u, v)$ denotes the shortest distance between u and v, thus

$$[\hat{A}^k]_{u,v} \begin{cases} = 0, & \text{if } \mathcal{D}(u,v) > k \\ \neq 0, & \text{if } \mathcal{D}(u,v) \leq k \end{cases} \tag{9}$$

Given a specific target node $t \in V$, it is clear that $\tilde{\mathcal{L}}_t$ depends on Z_t only, where $Z_t = [\hat{A}^k]_t XW$. So what we need is to compute $[\hat{A}^k]_t$, a row-vector of $\hat{A}^k$. According to Property 1, we can construct a k-hop subgraph $G^{(sub)} = (A^{(sub)}, X)$ consisting of:

$$\begin{aligned} &V_s = \{u \,|\, \mathcal{D}(t,u) \leq k\}, \\ &E_s = \{(u,v) \,|\, \mathcal{D}(t,u) \leq k \text{ and } \mathcal{D}(t,v) \leq k\} \end{aligned} \tag{10}$$

The set of nodes V_s is also considered as the k-hop neighbors of target node t.

There is a crucial problem that only the gradients of *connected* edge $e_i \in E_s$ will be computed, resulting in edge deletion only. To enlarge the possible perturbation set, it is also important to consider the gradients of the non-edges that might be connected later. Unfortunately, the possible number of non-edges is nearly N^2 and this makes the computation of gradients very costly.

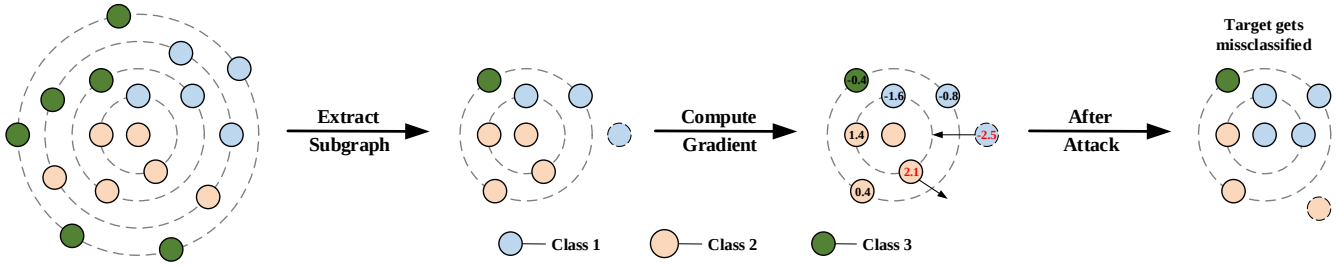


Fig. 2: Simplified gradient-based attack via subgraph. A two-hop subgraph centered at target node is extracted and some potential nodes (dotted-line nodes) outside the subgraph are added as well to enlarge the possible perturbation set. The values within nodes denote the gradients.

Inspired by DICE [38], a straightforward way to attack is "Disconnect Internally, Connect Externally". Following this insight, we will only consider those nodes that belong to different classes to construct a non-edge set.

However, if a graph is large, there are still a great number of nodes that belong to different classes. To avoid excessive computation, we empirically add nodes and edges to fulfill

$$\begin{aligned} V_p &= \{u \,|\, c_u = c'_t, u \in V\}\,, \\ E_p &= \mathcal{A} \times V_p\,, \end{aligned} \tag{11}$$

where $c'_t = \arg\max_{c' \neq c_t} Z_{t,c'}$ is the next most probable class obtained from surrogate model SGC, $\mathcal{A} \subseteq V$ is the set of *attacker nodes* and the perturbations are constrained to these nodes [12]. In particular, we set $\mathcal{A} = \{t\}$ for direct attack and $\mathcal{A} = \mathcal{N}(t)$ for influence attack where $\mathcal{N}(t)$ is the set of neighboring nodes adjacent to t. We term these nodes in V_p as "potential nodes" and edges in E_p as "potential edges" since they may be included in this subgraph later. Eq.(11) shows that we prefer to connect attacker nodes in $\mathcal{A}$ with potential nodes to influence the target node t. Therefore, the targeted loss can be also simplified as

$$\tilde{\mathcal{L}}_t^{(sub)} = \ln Z_{t,c'_t}^{(sub)} - \ln Z_{t,c_t}^{(sub)}\,, \quad t \in V \tag{12}$$

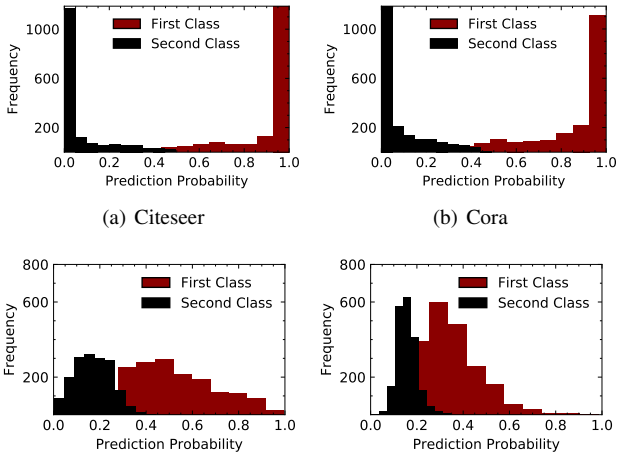
where $\tilde{\mathcal{L}}_t^{(sub)}$ is derived from Eq.(8) which denotes the targeted loss computed via subgraph, $Z_t^{(sub)} = f(A^{(sub)}, X) \in \mathbb{R}^{1\times C}$ denotes the prediction probability vector of target node t via subgraph.

As discussed above, it is clear that $|V_p| = N/C$. As an extreme case suppose there are only a few classes in the dataset, i.e., C is small enough. In this case, it can be derived that $|V_p| = \frac{N}{C} \approx N$, the scale of potential nodes comes larger and unbearable. Consider that we only have a budget Δ for a target node, which means that we can add Δ edges at most. To further improve the efficiency, we will eventually leave Δ potential nodes $\hat{V}_p \subseteq V_p$, where the gradients of potential edges between t and them are Δ-largest, i.e.

$$\begin{aligned} \hat{V}_p &= \{u \,|\, \nabla_{G_{t,u}^{(sub)}} \text{is}\, \Delta\text{-largest},\ u \in V_p\}, \\ \hat{E}_p &= \mathcal{A} \times \hat{V}_p, \end{aligned} \tag{13}$$

where $\nabla_{G_{t,u}^{(sub)}}$ is the gradient of loss $\tilde{\mathcal{L}}_t^{(sub)}$ w.r.t. the edge (t, u).

Let $V^{(sub)} = V_s \cup \hat{V}_p$, $E^{(sub)} = E_s \cup \hat{E}_p$, the scale of the subgraph $G^{(sub)}$ comes smaller. We only need to compute $\nabla_{G_{u,v}^{(sub)}}$ for each (non-)edge $(u, v) \in E^{(sub)}$. Either the input graph is sparse or not, it's obvious that $|E^{(sub)}| \ll |E|$. Besides, computing $Z_t^{(sub)}$ using $A^{(sub)}$ can be done in constant time and it further improves the efficiency.



(a) Citeseer (b) Cora

(c) Citeseer (calibrated) (d) Cora (calibrated)

Fig. 3: The prediction probability of the first and next most probable (second) class of SGC on Citeseer and Cora datasets before and after calibration.

Figure 2 shows the simplified gradient-based attack via subgraph, where we not only extract the k-hop subgraph centered at the target node but also add some potential nodes to extend the subgraph. By doing so, we can enlarge the possible perturbation set and compute the gradients of potential edges in this extended subgraph, hence our method is available to add adversarial edges as well.

4.3 Gradient Computation with Calibration

Gradient-based algorithm is an efficient and effective method to attack. Similar to GradArgmax, the gradients of the targeted loss w.r.t the subgraph will be computed as follows:

$$\nabla_{G^{(sub)}} = \nabla_{A^{(sub)}} \tilde{\mathcal{L}}_t^{(sub)}\,, \tag{14}$$

Note that, to ensure $\hat{A}_{u,v}^{(sub)} = \hat{A}_{u,v}, \forall u, v \in V_s$, we must add self-loop for each node $u \in V^{(sub)}$, and normalize $\tilde{A}^{(sub)}$ using $\tilde{D}$ the same as $\tilde{A}$. Following this, apparently $Z_t^{(sub)} = Z_t$ according to Property 1. As described in Eq.(14), the partial derivative

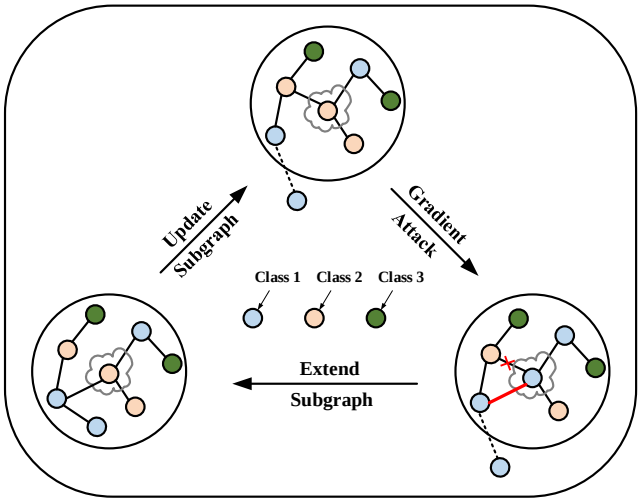


Fig. 4: Iterative gradient-based attack via subgraph expansion ($k = 2$). The largest circle denotes the radius (k) of neighborhood centered at target node. If a node becomes closer to the target node within k-distance, the edges connected to it should be considered as well.

of $\tilde{\mathcal{L}}_t^{(sub)}$ w.r.t the adjacency matrix $A^{(sub)}$ is computed for each (non-)edge $e = (u, v) \in E^{(sub)}$.

However, as stated by Guo et al. [39], most of the modern neural networks are poorly calibrated, and the probability associated with the predicted class label doesn't reflect its ground-truth correctness likelihood. On the contrary, *a neural network appears to be overconfident on the predictions*. As shown in Figure 3(a) and Figure 3(b), the model is so confident that it gives very high probability for the predicted class ($Z_{t,c_t} \to 1$), but for other classes even the next most probable one, it gives an extremely low probability ($Z_{t,c'_t} \to 0$). As a result, the first term of targeted loss $\tilde{\mathcal{L}}_t^{(sub)}$ rounds to minus infinity while the second term rounds to zero. This results in the gradients vanishing and thus the attack performance is negatively affected. To overcome the gradient vanishing problem, a scale factor $\epsilon > 1$ is introduced to calibrate the output:

$$Z_t^{(sub)} = f_\theta(A^{(sub)}, X, \epsilon) = \text{softmax}\left(\frac{\hat{A}^{(sub)k} XW}{\epsilon}\right). \tag{15}$$

By introducing a scale factor ϵ, the output of the model will no longer produce the extreme value 0 or 1 (See Figure 3(c) and Figure 3(d)), thereby avoid the gradients vanishing.

4.4 Iterative Gradient-based Attack

To better capture the actual change of the surrogate loss, we will sequentially compute the gradients after adding or removing an edge [15]. Let $Z_t^{(i)}$ and $Z_t^{(sub_i)}$ denote the prediction of target node t at the i_{th} time using the whole graph and subgraph, respectively. $A^{(sub_i)}$, $V^{(sub_i)}$, $E^{(sub_i)}$ are defined in the same way. Specifically, $A^{(sub_0)} = A^{(sub)}$, $V^{(sub_0)} = V^{(sub)}$, $E^{(sub_0)} = E^{(sub)}$.

Algorithm 1 Simplified gradient-based attack (SGA)

Input:
Graph $G = (A, X)$, attack budget Δ, target node t, ground-truth label c_t, labeled nodes V_L, hyper-parameter k for surrogate model SGC;
Output:
Perturbed graph G'_t w.r.t target node t;
$\theta \leftarrow$ train the surrogate model on labeled nodes V_L;
$c'_t = \arg\max_{c' \neq c_t} Z_{t,c'} \leftarrow$ predict the next most probable class of t ;
Extract the k-hop subgraph centered at t ;
Initialize the subgraph $G^{(sub)} = (A^{(sub)}, X)$ via Eq.(13);
$A^{(sub_0)}, E^{(sub_0)}, V^{(sub_0)} \leftarrow A^{(sub)}, E^{(sub)}, V^{(sub)}$;
for $i = 0$ to $\Delta - 1$ **do**
  $Z_t^{(sub_i)} = f_\theta(A^{(sub_i)}, X, \epsilon)$;
  Compute $\nabla_{G^{(sub_i)}}$ with Eq.(14);
  Compute structure score S with Eq.(17);
  Select $e = (u, v)$ with largest structure score $S_{u,v}$;
  Update $A^{(sub_{i+1})}, E^{(sub_{i+1})}, V^{(sub_{i+1})}$ with Eq.(19);
end for
return G'_t;

At each time i+1, we flip one edge $e = (u, v) \in E^{(sub_i)}$ with the *largest* magnitude of gradient that fulfills the constraints [40]:

$$\begin{cases} \text{Connect } (u, v), & \text{if } \nabla_{G^{(sub_i)}} > 0 \text{ and } A_{u,v}^{(sub_i)} = 0 \\ \text{Disconnect } (u, v), & \text{if } \nabla_{G^{(sub_i)}} < 0 \text{ and } A_{u,v}^{(sub_i)} = 1 \end{cases} \tag{16}$$

To conclude, we define the structure score as follows to simplify Eq.(16):

$$S = \nabla_{G^{(sub_i)}} \odot (-2A^{(sub_i)} + 1), \tag{17}$$

where $\odot$ denotes Hadamard product. The adversarial edge (u, v) is selected based on the *largest* structure score. After an edge is selected, we update the subgraph $G^{(sub_{i+1})}$ as follows:

$$\begin{aligned} V^{(sub_{i+1})} &= V^{(sub_i)}, \\ E^{(sub_{i+1})} &= E^{(sub_i)} - \{e\}, \\ A^{(sub_{i+1})} &= A^{(sub_i)} \oplus e, \end{aligned} \tag{18}$$

where $\oplus$ denotes the "exclusive or" operation, for $e = (u, v)$, if $A_{u,v}^{(sub_i)} = 0$ then $A_{u,v}^{(sub_{i+1})} = 1$ and vice versa. However, it will results in $Z_t^{(sub_i)} \neq Z_t^{(i)}, \forall i > 0$. We explain it with Figure 4, when a potential edge is connected, if a node becomes closer to the target node within k-distance, the edges adjacent to it should be considered as well, so as to *preserve the k-order message aggregation of graph convolution*. Therefore, if an edge $e = (u, v) \in E^{(sub_i)}$ is connected (assume that v is a potential node), we must expand the subgraph as follows:

$$\begin{aligned} V^{(sub_{i+1})} &= V^{(sub_i)} \cup \{v' \,|\, v' \in \mathcal{N}(v) \text{ if } \mathcal{D}(t, v') \leq k\}, \\ E^{(sub_{i+1})} &= E^{(sub_i)} \cup (E^{(exp)} - \{e\}), \\ A^{(sub_{i+1})} &= A^{(sub_i)} \oplus (E^{(exp)} \cup \{e\}), \end{aligned} \tag{19}$$

where $E^{(exp)} = \{(v, v') \,|\, v' \in \mathcal{N}(v) \text{ if } \mathcal{D}(t, v') \leq k\}$ is the expended edges set where nodes become closer within k-distance.

Since the scale of potential nodes is small enough, the subgraph expansion does not negatively affect the efficiency[1]. The details of iterative SGA is summarized in Algorithm 1.

1. In contrast, if an edge is disconnected from the subgraph, the message aggregation will be blocked automatically for those nodes beyond k-distance.

TABLE 2: Time and Space Complexity of Adversarial Attack Algorithms.

Methods	Space complexity	Time complexity
Nettack	$\mathcal{O}(\|E\|)$	$\mathcal{O}(\Delta \cdot \|\mathcal{A}\| \cdot N \cdot d^2)$
GradArgmax	$\mathcal{O}(\|E\|)$ or $\mathcal{O}(N^2)$	$\mathcal{O}(\Delta + \|E\|d^2)$ or $\mathcal{O}(\Delta + N^2 d^2)$
SGA	$\mathcal{O}(d^k + \Delta \cdot \|\mathcal{A}\|)$	$\mathcal{O}(\Delta \cdot (d^k + \Delta \cdot \|\mathcal{A}\|) \cdot d^k)$

4.5 Analysis of Time and Space Complexity

To better describe our method, we compare the complexity of time and space with other state-of-the-art methods: *Nettack* and *GradArgmax*.

4.5.1 Nettack

Due to store the whole graph (as a sparse matrix), the space complexity of Nettack comes to $\mathcal{O}(|E|)$. Besides, Nettack computes the misclassification loss sequentially w.r.t the perturbed graph for each edge in the candidate set, the time complexity is up to $\mathcal{O}(\Delta \cdot |\mathcal{A}| \cdot N \cdot d^2)$ without considering feature attack, where d denotes the average degree of nodes in the graph.

4.5.2 GradArgmax

While storing the whole graph with a sparse matrix, the space complexity of GradArgmax will be $\mathcal{O}(|E|)$. If a dense instantiation of the adjacency matrix is used, the space complexity comes to $\mathcal{O}(N^2)$. It's worth mentioning that, for the adjacency matrix in sparse form, the algorithm is able to delete edges only, while both adding and deleting edges are available for the dense one. Besides, GradArgmax only computes gradients once, so the major time complexity depends on the computation of gradients and it is approximate $\mathcal{O}(\Delta + |E|d^2)$ or $\mathcal{O}(\Delta + N^2 d^2)$ for sparse or dense matrix[2].

4.5.3 SGA

Our method only stores the subgraph with a sparse matrix, thus the space complexity is only $\mathcal{O}(|E^{(sub)}|)$, where $E^{(sub)}$ only consists of edges of k-hop subgraph and a few potential edges, which comes to $|E^{(sub)}| = d^k + \Delta \cdot |\mathcal{A}|$, apparently $|E^{(sub)}| \ll |E|$ for a sparse graph. Moreover, the time complexity depends on the message aggregation between the k-hop neighbors of the target node in $A^{(sub)}$, thereby the time complexity leads to $\mathcal{O}(\Delta \cdot |E^{(sub)}| \cdot d^k)$. Given $|E^{(sub)}| = d^k + \Delta \cdot |\mathcal{A}|$, it can be simplified as $\mathcal{O}(\Delta \cdot (d^k + \Delta \cdot |\mathcal{A}|) \cdot d^k)$. The complexity of the three methods are summarized in Table 2, where the usage of memory refers to the usage of graph (subgraph). It is clear that SGA theoretically achieves both time and space efficiency compared to Nettack and GradArgmax.

5 UNNOTICEABLE ADVERSARIAL ATTACK ON GRAPH DATA

Specifically, in the scenario of adversarial attack, attackers would attempt to perform perturbations by modifying the input data while guaranteeing *unnoticeability* to avoid the detection. Unlike the continuous data that lies in *Euclidean Space*, attackers can evaluate the attack impacts by measuring the changes before and after attack via ℓ_2 norm or ℓ_∞ norm [17]. However, it is difficult to quantify the attack impacts on graph data that lies in such a *non-Euclidean Space*.

An important property of a graph is the degree distribution, which often appears as a power-law distribution [42], i.e., $p(x) \propto x^{-q}$, where q is the scaling parameter. It is easy to tell if two graphs have similar degree distributions after q is given. However, there is no solution for estimating q exactly yet. To this end, Zügner et al. [12] derive an efficient way to check for violations of the degree distribution after attacks. Specifically, they estimate q for a clean graph and q' for a perturbed graph to ensure that the attack is unnoticeable if the following equation is fulfilled:

$$\Lambda\left(q, q'; G, G'\right) < \tau \approx 0.004\,,$$

where Λ is a discriminate function defined in [12], G denotes the original graph and G' the perturbed one.

In this section, we refer to another solution and apply it to measure the attack impacts, *degree assortativity coefficient* [19], [20], it measures the tendency of nodes connected to each other. To clarify this, we first define the degree mixing matrix M, where $M_{i,j}$ denotes the tendency of nodes with degree i connected to nodes with degree j, the value can be counts, joint probability, or occurrences of node degree. Let $\alpha, \beta \in \{in, out\}$ denote the type of in-degree and out-degree, respectively. We first ensure that M fulfills the following rules:

$$\sum_{ij} M_{i,j} = 1, \quad \sum_{j} M_{i,j} = \alpha_i, \quad \sum_{i} M_{i,j} = \beta_j\,, \tag{20}$$

Particularly, in an undirected graph, $\alpha_i = \beta_j, \forall i = j$. Then we define the degree assortativity coefficient by using Pearson Correlation [43]:

$$r(\alpha, \beta) = \frac{\sum_{ij} ij(M_{i,j} - \alpha_i \beta_j)}{\sigma_\alpha \sigma_\beta}, \tag{21}$$

where σ_α and σ_β are the standard deviations of the distributions α and β, respectively. The value of r lies in the range of $-1 \leq r \leq 1$, and $r = -1$ for disassortativity while $r = 1$ for assortativity.

Degree assortativity coefficient r is much easier to compute than degree distribution coefficient q, and it better reflects the graph structure. Based on this, we propose **Degree Assortativity Change (DAC)**, which is defined as

$$\mathrm{DAC} = \frac{\mathbb{E}_r(\,|r_G - r_{G'_{t_i}}|\,)}{r_G}, \quad \forall t_i \tag{22}$$

where G'_{t_i} denotes the perturbed graph w.r.t target node t_i. Attackers will conduct attack for each target node t_i by performing perturbations on the original graph G. DAC measures the average impacts on attacking a group of target nodes $\{t_i\}$, and the smaller the DAC is, the less noticeable the attack will be.

6 EXPERIMENTS

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

RQ1 How much can SGA improve in terms of time and space efficiency and how does it work for the proposed metric DAC?

RQ2 Can SGA achieve a competitive attack performance while using only a certain part of nodes to attack?

RQ3 Can SGA scale to larger datasets and maintain considerable attack performance?

In what follows, we first detail the experimental settings, followed by answering the above three research questions.

2. Refer to [41], the time complexity of GCN is $\mathcal{O}(|E|d^2)$, and the computation of gradients can be accelerated by parallel computing, which is also suitable for SGA.

TABLE 3: Dataset statistics. Only consider the largest connected component of the graph for each dataset.

Statistics	Citeseer	Cora	Pubmed	Reddit
#Nodes	2,110	2,485	19,717	232,965
#Edges	3,668	5,069	44,324	11,723,402
Density	0.082%	0.082%	0.011%	0.022%
Average Degree	3.50	4.08	4.50	99.65

6.1 Experimental Settings

Datasets. We evaluate the performance of our method on four well-known datasets: Citeseer, Cora, Pubmed [44] and Reddit [45]. The first three are commonly used citation networks, where nodes are documents and edges among them are citation relations. The last one is a large scale dataset of online discussion forums where user posts and comments on content in different topical communities. We follow the setting of Nettack [12] and only consider the largest connected component of the graph for each dataset. Besides, we randomly select 20% of nodes to constitute the training set (half of them are treated as the validation set) and treat the remaining as the testing set. Table 3 is an overview of the datasets.

Target Model. We consider poisoning attacks, i.e., models are retrained on the perturbed graph until the convergence after attacks [24], which is more challenging for attackers but reflects real-world scenarios better. To evaluate the transferability of our method, we conduct attack on several commonly used graph neural networks: GCN [8], SGC [18], GAT [46], GraphSAGE [45], ClusterGCN [47]. For each target model, our method is compared with other adversarial attack methods.

- **GCN** [8]. GCN is one of the most representative graph neural networks that learn hidden representations by encoding both local graph structure and node features.
- **SGC** [18]. SGC is a linear variant of GCN that has been proposed recently, which achieves competitive results and even significantly improves the training efficiency.
- **GAT** [46]. GAT enhances GCN by leveraging a masked self-attention mechanism to specify different weights to different neighbor nodes.
- **GraphSAGE** [45]. GraphSAGE is a general inductive framework, which uniformly samples a set of neighbors with a fixed size, instead of using a full-neighborhood set during training.
- **ClusterGCN** [47]. This is the state-of-the-art mini-batch GCN framework. It samples n subgraphs whose nodes have high correlations under a graph clustering algorithm and restricts the message aggregation within these subgraphs.

Evaluation Protocols. For the task of node classification, we aim to perform perturbations and further cause misclassification of target models. To this end, we evaluate the classification accuracy and classification margin (CM). Given a target node t, CM is defined as the probability margin between ground-truth label c_t and the next most probable class label c'_t, which lies in the range of $[-1, 1]$. The smaller CM means better attack performance. In particular, a successful attack is often with CM less than zero.

Baselines. We compare our method with four other baseline methods and conduct attacks using two strategies respectively (i.e., direct attack and influence attack). For a fair comparison, all methods will use the same surrogate model SGC (if necessary) and share the same weights θ. Follow the setting of Nettack [12], the attack budget Δ is set to the degrees of target node t.

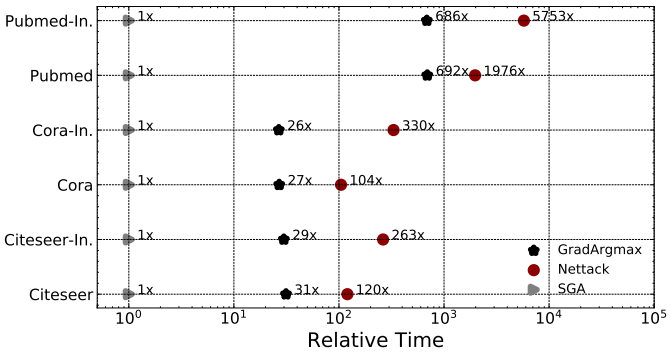


Fig. 5: Performance over running time on three datasets for direct and influence attack.

- **Random Attack (RA)**. RA randomly adds or removes edges between attacker nodes and other nodes in the graph with probability p_1. This is the simplest way to conduct attacks.
- **DICE** [38]. DICE is originally a heuristic algorithm for disguising communities. In our experiment, DICE randomly decides whether to connect or disconnect an edge with probability p_2 between attacker nodes and other nodes in the graph. Besides, there is a constraint that only nodes belonging to the same class/different classes will be disconnected/connected.
- **GradArgmax** [13]. Since the attack budget Δ is defined as the degrees of target node t, the original GradArgmax will remove all edges connected with t for direct attack, which is unreasonable and unfair to compare. Therefore, we use the variant of GradArgmax, which requires a dense instantiation of the adjacency matrix and computes the gradients of all N^2 edges.
- **Nettack** [12]. Nettack is the strongest baseline that can modify the graph structure and node features. As we focus on the structure attack, we restrict Nettack to modify the graph structure with budget Δ only.

Parameter Settings. The hyper-parameters of target models are fine-tuned in the clean graph for each dataset. Particularly, for SGA, the radius $k = 2$ and the scale factor $\epsilon = 5.0$ for each dataset, and the number of added potential nodes is set to Δ when generating adversarial examples. For RA and DICE, p_1 and p_2 are both fixed at 0.5. All models are implemented in Tensorflow[3], running on a NVIDIA RTX 2080Ti GPU.

Attack Settings. Our experiments involve two parts: (i) *Generating adversarial examples*. For each dataset, we first randomly select 1,000 nodes from the testing set as target nodes and generate adversarial examples for them separately using different attack methods. (ii) *Conduct attack through adversarial examples*. After adversarial examples are generated, we conduct poisoning attacks on the surrogate model SGC, and further transfer it to other commonly used graph neural networks.

3. https://www.tensorflow.org/

TABLE 4: Average time (s), memory usage, and DAC to generate adversarial examples using both attack strategies. The statistics of random algorithms RA and DICE are omitted. OOM stands for a method that could not be used to attack due to the limitations of GPU RAM.

Methods	Citeseer			Cora			Pubmed			Reddit		
Direct Attack	Time	Memory	DAC	Time	Memory	DAC	Time	Memory	DAC	Time	Memory	DAC
GradArgmax	0.188	17MB	7.9E-2	0.297	23MB	2.9E-3	13.856	1483MB	4.2E-3	N/A	OOM	-
Nettack	0.722	66KB	3.2E-2	1.151	66KB	1.4E-3	39.526	769KB	1.3E-3	N/A	180MB	-
SGA	0.006	2KB	3.4E-2	0.011	2KB	1.7E-3	0.020	4KB	1.8E-3	12.236	14MB	2.5E-6
Influence Attack	Time	Memory	DAC	Time	Memory	DAC	Time	Memory	DAC	Time	Memory	DAC
GradArgmax	0.239	17MB	1.7E-2	0.321	23MB	1.1E-3	14.412	1483MB	9.2E-4	N/A	OOM	-
Nettack	2.104	89KB	1.6E-2	3.969	89KB	9.4E-4	120.821	769KB	8.1E-4	N/A	180MB	-
SGA	0.008	5KB	1.7E-2	0.012	6KB	1.0E-3	0.021	7KB	8.7E-4	14.571	14MB	1.9E-6

TABLE 5: Accuracy(%) of direct attack and influence attack on two small scale datasets. Here the best results are **boldfaced**.

Methods	Citeseer					Cora				
Direct Attack	SGC	GCN	GAT	GraphSAGE	Cluster-GCN	SGC	GCN	GAT	GraphSAGE	Cluster-GCN
Clean	71.8	71.6	72.1	71.0	71.4	83.1	83.5	84.2	82.0	83.2
RA	62.0	63.6	59.0	61.9	62.8	67.5	68.8	69.7	68.8	68.1
DICE	55.0	57.8	54.6	56.6	57.1	58.4	60.2	61.8	59.7	59.5
GradArgmax	12.2	13.2	21.5	32.4	34.8	25.2	32.6	34.5	30.9	45.6
Nettack	3.7	6.0	20.5	**27.9**	30.9	**1.0**	2.4	17.6	27.2	17.8
SGA	**1.8**	**3.8**	**20.3**	30.2	**19.9**	1.5	**2.1**	**15.9**	**25.8**	**17.6**
Influence Attack	SGC	GCN	GAT	GraphSAGE	Cluster-GCN	SGC	GCN	GAT	GraphSAGE	Cluster-GCN
RA	71.2	71.1	71.4	68.8	70.8	80.9	83.0	86.0	81.4	83.0
DICE	69.4	71.1	70.2	67.5	68.4	80.5	82.4	85.4	79.8	82.7
GradArgmax	47.8	48.1	51.2	46.7	55.4	65.1	70.8	73.8	74.7	72.9
Nettack	**31.4**	39.2	49.3	41.9	49.8	48.4	56.4	63.2	63.5	60.0
SGA	33.1	**38.5**	**45.2**	**40.2**	**48.1**	**46.2**	**56.2**	**62.6**	**62.8**	**57.1**

6.2 Performance on Generating Adversarial Examples (RQ1)

As described in Table 2, our method is highly efficient in terms of time and space complexity compared to other methods. For running time, here we theoretically analyze how much improvement our method can achieve over Nettack, the state-of-the-art adversarial attack method. Following the parameter settings described in Section 6.1, i.e., $k = 2$ and $\Delta = d$, the relative time complexity can be approximated as

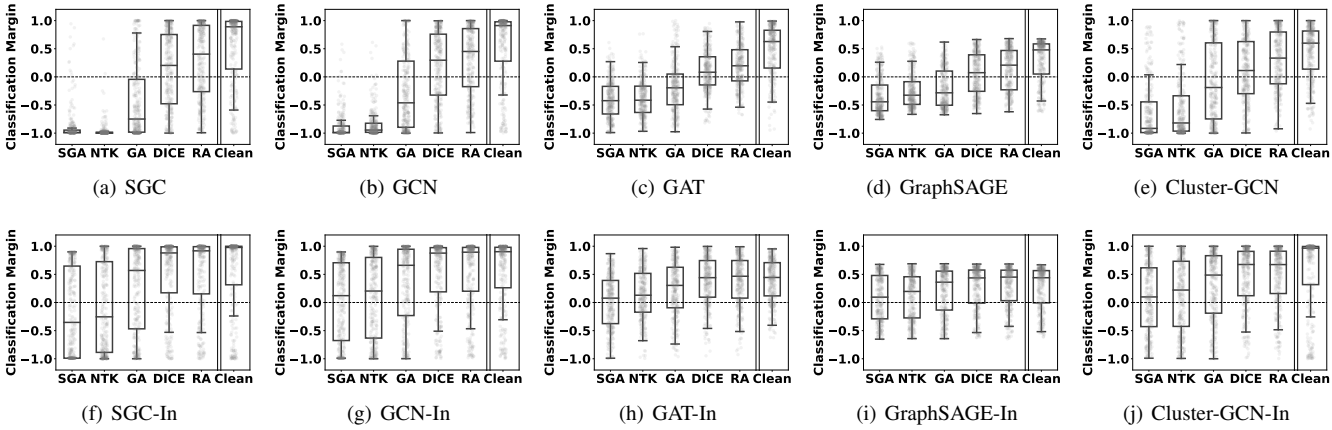
$$\frac{\mathcal{O}(d \cdot |\mathcal{A}| \cdot N \cdot d^2)}{\mathcal{O}(d \cdot (d^2 + d \cdot |\mathcal{A}|) \cdot d^2)} = \mathcal{O}(\frac{|\mathcal{A}| \cdot N}{d^2 + d \cdot |\mathcal{A}|}) \,. \tag{23}$$

This means that SGA can theoretically achieve $\frac{|\mathcal{A}| \cdot N}{d^2 + d \cdot |\mathcal{A}|}$ times speedup compared to Nettack, and it will be higher for larger and sparser graphs.

Table 4 shows the performance of generating adversarial examples with direct and influence attack strategies, respectively. Here the training time of the surrogate model is excluded and memory usage refers to the usage of storing the graph (subgraph). Please note that we do not compare the running time and memory usage with RA and DICE since they both are random algorithms and comparison doesn't make sense. Because the first three datasets are sparse enough with small degrees on average (see Table 3), the two-hop subgraph is much smaller and SGA has a remarkable improvement in efficiency. Intuitively, we plot the performance of Nettack and GradArgmax over their running time relative to that of SGA on three datasets in Figure 5. Considering the Citeseer dataset (statistics are in Table 3) and the direct attack setting (i.e., $|\mathcal{A}| = 1$), the theoretical speedup comes to 134 times according to Eq.(23), which is approximately consist with the experimental results (120 times).

Table 4 indicates that our method is much more efficient than GradArgmax and Nettack in terms of time and space complexity. Even if the dataset grows in size, SGA remains high efficiency as if the graph is sparse enough. On Pubmed dataset, our method can even yield up to three orders of magnitude speedup over Nettack. Also, SGA is much more memory efficient. On the contrary, Nettack becomes less efficient, especially when performing influence attacks on a larger dataset, the reason is that the larger scale of attacker nodes $\mathcal{A}$ and candidate edges set. As for GradArgmax, the running time and memory usage are similar between direct and influence attacks, since it computes the gradients of the whole graph all the time and the candidate edges set are always the same. In addition, both of them failed on Reddit dataset, a large and dense graph, but SGA still achieves high efficiency.

In respect to the metric DAC, Table 4 shows that: (i) Direct attacks have greater impacts than influence attacks, it is clear and interpretable since perturbations are restricted in the neighborhood of the target node, leading to a significant degree changes especially the target node. Naturally, the concentrated perturbations will exert a greater influence; (ii) Nettack achieves the most unnoticeable influence because it is enforced to preserve the graph's degree distribution during attacks. Our method leverages the subgraph instead of the whole graph without any constraints on the degree distribution, thus achieving a slightly worse result but



(a) SGC (b) GCN (c) GAT (d) GraphSAGE (e) Cluster-GCN

(f) SGC-In (g) GCN-In (h) GAT-In (i) GraphSAGE-In (j) Cluster-GCN-In

Fig. 6: Results on Cora dataset using direct attack (top) and influence attack (-In, bottom). "NTK" is short for Nettack and "GA" is short for GradArgmax.

still better than GradArgmax. Note that, GradArgmax considers all N^2 edges as a candidate set to modify, it will largely affect the degree distribution of nodes in the graph, and the attack becomes more noticeable.

6.3 Performance on Attacking Graph Neural Networks (RQ2)

For the target model SGC, whose details of the architecture are transparent (but not for its weights), it can be approximately treated as a white-box attack if it is used as a surrogate model. However, for other graph neural network models, this is a complete black-box attack without any prior knowledge.

As shown in Table 5, we report the percentages of target nodes that are correctly classified, where *clean* stands for results in the original graph. The classification performance of SGC drops significantly against direct attacks. SGC is more vulnerable than other classifiers against influence attacks. Unlike SGC, other models are more robust even in the direct attack setting. There are four possible reasons to explain this: (i) The nonlinear activation function. SGC and GCN are similar except for activation functions. SGC drops the nonlinear activation in the hidden layer and collapses weight matrices between consecutive layers. Although SGC behaves more efficiently during training, the simplification leads to lower robustness than GCN. (ii) The details of other models' architecture are not exposed to attackers like SGC, so the performance depends on the transferability of attacks; (iii) The message aggregation methods are more robust than simple graph convolution. For instance, GAT introduces the attention mechanism and enables (implicitly) to specify different weights to different nodes in a neighborhood. For GraphSAGE, the most robust one in most cases, we argue that it is probably on account of the *concatenate* operation on the node's message and the neighborhoods', which can be regarded as a residual term between layers. Note that we only consider structure attack rather than feature attack, the node feature of itself is unperturbed. By doing so, the influence of attacker nodes will be largely alleviated. (iv) Pre-processing on the input graph. For Cluster-GCN, it samples several subgraphs whose nodes have high correlations using a graph clustering algorithm and restricts the message aggregation within these subgraphs, which also alleviates the influence of attacker nodes.

In Figure 6, we can observe that SGA achieves a considerable performance in most cases compared with different attack methods. Nettack, a strong baseline, also yields a significant performance as reported in [12]. Most remarkably, even in attacking other robust graph neural networks (GAT, GraphSAGE, Cluster-GCN), most of the classifiers are strongly affected by perturbations especially performed by SGA and Nettack. The obtained results have proved the vulnerability of graph neural networks and are consistent with the previous studies. Not surprisingly, influence attacks achieve a worse performance compared with direct attacks. As expected, random algorithms RA and DICE both have a slight attack effect on attacking different graph neural networks. Both GradArgmax and SGA are gradient-based methods, we can also see that GradArgmax performs worse than our method SGA although the whole graph is used to attack. The possible explanations for the results are as follows: (i) The misclassification loss of SGA considers the loss of the next most probable class label as well as the ground-truth label, which is available to better exploit the vulnerability of graph neural networks. (ii) As detailed in Section 4.3, the surrogate model appears to make an overconfident output and it causes the vanishing of gradients. SGA introduces a scale factor to calibrate the surrogate model, the gradient vanishing is alleviated and thus achieves a better performance. (iii) SGA focuses on a part of nodes — the k-hop subgraph centered as the target node with some potential nodes. As a result, the generated perturbations will be more concentrated and cause the misclassification of target classifiers much easier.

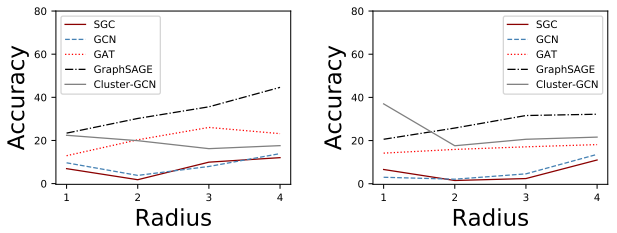
6.4 Scalability for Larger Datasets (RQ3)

As illustrated above, SGA achieves a considerable performance on attacking most graph neural networks on two small-scale datasets Cora and Citeseer, and SGA also ensures time and space efficiency. To evaluate the scalability of SGA, we extend our experiments to two larger datasets — Pubmed and Reddit, the data statistics are described in Table 3. Particularly, Reddit is a relatively dense graph with node degrees up to 99.65 on average, which means that the attack budgets are much higher than other datasets in our settings, and it naturally brings more challenges on time and memory usage.

As shown in Table 4, GradArgmax and Nettack become slower, and more memory usage is required on larger datasets.

TABLE 6: Accuracy(%) of direct attack and influence attack on two large scale datasets. Here the best results are **boldfaced**.

Methods	Pubmed					Reddit	
Direct Attack	SGC	GCN	GAT	GraphSAGE	Cluster-GCN	SGC	GCN
Clean	84.2	86.5	85.3	86.1	85.7	93.8	93.6
RA	76.6	78.2	75.0	81.5	75.8	88.6	89.4
DICE	65.0	67.1	57.8	72.3	64.0	84.2	88.2
GradArgmax	14.8	15.8	16.9	42.7	48.3	-	-
Nettack	2.0	3.4	10.9	32.4	19.9	-	-
SGA	**1.6**	**2.0**	**9.3**	**30.6**	**15.7**	**1.2**	**5.4**
Influence Attack	SGC	GCN	GAT	GraphSAGE	Cluster-GCN	SGC	GCN
RA	84.1	86.2	85.3	87.2	84.6	93.6	93.5
DICE	83.6	85.9	84.7	85.8	84.4	93.7	93.7
GradArgmax	73.2	77.8	72.7	84.0	76.2	-	-
Nettack	61.4	72.0	70.1	85.4	69.2	-	-
SGA	**63.4**	**70.7**	**70.0**	**82.6**	**67.8**	**78.9**	**83.6**



(a) Citeseer (b) Cora

Fig. 7: Performance of SGA on attacking different graph neural network models with various radius on Citeseer and Cora datasets.

Especially for the Reddit dataset, both GradArgmax and Nettack have failed due to high time and space complexity. On the contrary, SGA has similar time and memory usage on Pubmed dataset as it is sparse as Citeseer and Cora. Even on Reddit dataset, SGA has high efficiency in the direct attack (12.236s) and the influence attack (14.571s) settings, respectively.

Furthermore, we conduct attacks on the same target models by the generated adversarial examples. As shown in Table 6, SGA achieves state-of-the-art results in all cases, a significant performance decrease is observed on most of the target models especially SGC. As the data scale grows, random algorithms RA and DICE have little or no effect on target models. On the largest dataset Reddit, GradArgmax and Nettack have failed to conduct attacks. But for SGA, the obtained results show that only a small part of target nodes are correctly classified by SGC and GCN in the direct attack setting. In the influence attack setting, SGA also has an obvious effect on the target models compared to RA and DICE. Results on Pubmed and Reddit datasets suggest that our method can easily scale to larger datasets and significantly degrade the performance of target models.

6.5 Ablation Study

In order to explore the effect of different radius on SGA, we report the ablation test on Citeseer and Cora datasets from the various radius of the subgraph, i.e., $k = 1, 2, 3, 4$. In Figure 7, it can be observed that $k = 2$ achieves the best performance on attacking SGC and GCN on both datasets. But for other target models, a better performance is achieved when $k = 1$ or $k = 3$.

However, if $k = 1$ the subgraph includes only the first-order neighboring nodes of the target node (except for potential nodes), SGA can only delete the edges that are directly connected to the target node but it is not allowed in the influence attack setting. As the radius of the subgraph gets larger, the number of nodes and edges increases exponentially, and it would require more time and memory usage to perturb the graph. So we need to make a trade-off between efficiency and performance. Given that the best classification performance can be achieved by a two-layer GCN or SGC in most datasets, SGA can exploit the vulnerability of graph neural network models with only a two-hops subgraph, and the efficiency of attack is also preserved.

7 Discussions

In this section, we will make a discussion about our proposed methods: SGA and DAC. Specifically, we detail how the proposed SGA can be generalized to other types of attack, i.e., feature attack and node injection attack. Besides, we discuss how to better ensure the unnoticeability of attackers by adopting the DAC metric during attacks.

7.1 *Feature Attack*

SGA was initially designed for adversarial structure attack, it can be easily generalized to the feature attack (either binary or continuous features) scenario as well. Feature attack is simpler than structure attack since the structure of the subgraph is fixed and there is no need to update the subgraph after an attack. Specifically, there are two cases: (i) binary node features. This is the most typical case. For example, in a typical citation graph, nodes are documents, edges are citation links, and node features are bag-of-words feature vectors. It is easy to conduct feature attacks, just by taking into account the gradients of the input features. Similar to structure attack, we can choose to add or remove the feature based on the surrogate gradients. (ii) continuous node features. This is another case where node features can be derived from other node embedding methods (e.g., DeepWalk [48] and Node2Vec [49]). We can treat it as continuous data by adopting gradient ascent to attack. This case is not suitable for our experimental settings since it is difficult to constrain such feature attacks within a given budget $\Delta \in \mathbb{N}$.

7.2 *Node Injection Attack*

Node injection attack is a new attack scenario [27], where attackers can add a group of vicious nodes into the graph so that topological of the original graph can be avoided and the victim node will be misclassified. SGA is not available for such an attack. The injected vicious node can be regarded as a singleton node because it isn't initially connected to any nodes. Therefore, the strategy of extracting a k-hop subgraph doesn't work in this case. However, the gradient-based method can be adapted to this scenario, which can be implemented by computing the gradients of all the non-edges between the injected node and other benign ones. Note that the computation still has a high time and memory usage as the graph scale grows. In the future, we plan to investigate more ways to improve the efficiency of conducting the node injection attack.

7.3 *Unnoticeability Constraint*

DAC is proposed as a practical metric to show the intensity of the impact after perturbations being performed. Although DAC overcomes the disadvantages of the traditional metrics, there

is room for improvement. Specifically, the evaluation of attack impacts via DAC has occurred after attacks, while an attacker may prefer to adopting it as an unnoticeability constraint during attacks. For example, the results of DAC can be used as an additional loss of the attack. However, the computation of DAC is costly, which motivates us to improve it in future work.

8 CONCLUSION AND FUTURE WORK

In this work, we study the adversarial attacks on the (attributed) graph and present a novel simplified gradient-based attack framework targeted on the node classification task. Specifically, we aim to poison the input graph and lead to the misclassification of graph neural networks on several target nodes. Based on the extensive experiments, our method SGA, which simply leverages a k-hop subgraph of the target node, has achieved high efficiency in terms of time and space and also obtained competitive performance in attacking different models compared to other state-of-the-art adversarial attacks. It can be also observed that SGA scales to larger datasets and achieves a remarkable attack performance. In addition, we emphasize the importance of measuring the attack impacts on graph data and further propose DAC as a measure. The results show that DAC works as a practical metric and SGA can also achieve a relatively unnoticeable attack impact.

Our works mainly focus on the node classification task and the targeted attack scenario. In future work, we aim to extend our method and generalize it to other graph analysis tasks with more flexibility.

ACKNOWLEDGMENTS

The research is supported by the Key-Area Research and Development Program of Guangdong Province (No. 2020B010165003), the National Natural Science Foundation of China (No. U1711267), the Guangdong Basic and Applied Basic Research Foundation (No. 2020A1515010831), and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (No. 2017ZT07X355).

REFERENCES

[1] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*. IEEE, 2013, pp. 6645–6649.

[2] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *BMVC 2015*. BMVA Press, 2015, pp. 41.1–41.12.

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[5] F. Xie, L. Chen, Y. Ye, Y. Liu, Z. Zheng, and X. Lin, "A weighted meta-graph based approach for mobile application recommendation on heterogeneous information networks," in *Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11236. Springer, 2018, pp. 404–420.

[6] F. Xie, L. Chen, Y. Ye, Z. Zheng, and X. Lin, "Factorization machine based service recommendation on heterogeneous information networks," in *2018 IEEE International Conference on Web Services, ICWS 2018, San Francisco, CA, USA, July 2-7, 2018*. IEEE, 2018, pp. 115–122.

[7] L. Chen, Y. Liu, X. He, L. Gao, and Z. Zheng, "Matching user with item set: Collaborative bundle recommendation with deep attention network," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pp. 2095–2101.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[9] L. Chen, Y. Liu, Z. Zheng, and P. S. Yu, "Heterogeneous neural attentive factorization machine for rating prediction," in *CIKM 2018*. ACM, 2018.

[10] F. Ye, J. Liu, C. Chen, G. Ling, Z. Zheng, and Y. Zhou, "Identifying influential individuals on large-scale social networks: A community based approach," *IEEE Access*, vol. 6, pp. 47 240–47 257, 2018.

[11] J. Liu, Y. Li, G. Ling, R. Li, and Z. Zheng, "Community detection in location-based social networks: An entropy-based approach," in *2016 IEEE International Conference on Computer and Information Technology, CIT 2016, Nadi, Fiji, December 8-10, 2016*. IEEE Computer Society, 2016, pp. 452–459.

[12] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. ACM, 2018, pp. 2847–2856.

[13] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 1123–1132.

[14] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 695–704.

[15] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[16] H. Chen, H. Jin, and S. Wu, "Minimizing inter-server communications by exploiting self-similarity in online social networks," *IEEE TPDS*, vol. 27, no. 4, pp. 1116–1130, 2016.

[17] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[18] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019.

[19] M. E. Newman, "Mixing patterns in networks," *Physical Review E*, vol. 67, no. 2, p. 026126, 2003.

[20] J. G. Foster, D. V. Foster, P. Grassberger, and M. Paczuski, "Edge direction and the structure of networks," *Proceedings of the National Academy of Sciences*, vol. 107, no. 24, pp. 10 815–10 820, 2010.

[21] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 19–35.

[22] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *SIGSAC*. ACM, 2018, pp. 619–633.

[23] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[24] L. Chen, J. Li, J. Peng, T. Xie, Z. Cao, K. Xu, X. He, and Z. Zheng, "A survey of adversarial learning on graph," *arXiv preprint arXiv:2003.05730*, 2020.

[25] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *CVPR*. IEEE Computer Society, 2016, pp. 2574–2582.

[26] K. Xu, H. Chen, S. Liu, P. Chen, T. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," in *IJCAI*. ijcai.org, 2019, pp. 3961–3967.

[27] J. Wang, M. Luo, F. Suya, J. Li, Z. Yang, and Q. Zheng, "Scalable attack on graph data by injecting vicious nodes," *CoRR*, vol. abs/2004.13825, 2020.

[28] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan, "Hiding individuals and communities in a social network," *Nature Human Behaviour*, vol. 2, no. 2, pp. 139–147, 2018.

[29] Q. Zhang, J. Fang, J. Zhang, J. Wu, Y. Xia, and Z. Zheng, "Cross entropy attack on deep graph infomax," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[30] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR (Poster)*. OpenReview.net, 2019.

[31] H. Chang, Y. Rong, T. Xu, W. Huang, H. Zhang, P. Cui, W. Zhu, and J. Huang, "A restricted black-box adversarial framework towards attacking graph embedding models," in *AAAI*. AAAI Press, 2020, pp. 3389–3396.

[32] J. Travers and S. Milgram, "An experimental study of the small world problem," in *Social Networks*. Elsevier, 1977, pp. 179–197.

[33] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *nature*, vol. 393, no. 6684, p. 440, 1998.

[34] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[35] L. A. N. Amaral, A. Scala, M. Barthelemy, and H. E. Stanley, "Classes of small-world networks," *Proceedings of the national academy of sciences*, vol. 97, no. 21, pp. 11 149–11 152, 2000.

[36] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, August 22-27, 2010 - Keynote, Invited and Contributed Papers*. Physica-Verlag, 2010, pp. 177–186.

[37] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[38] D. Cai, Z. Shao, X. He, X. Yan, and J. Han, "Mining hidden community in heterogeneous social networks," in *Proceedings of the 3rd international workshop on Link discovery*. ACM, 2005, pp. 58–65.

[39] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1321–1330.

[40] J. Chen, X. Lin, Z. Shi, and Y. Liu, "Link prediction adversarial attack via iterative gradient attack," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 4, pp. 1081–1094, 2020.

[41] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[42] A. Bessi, "Two samples test for discrete power-law distributions," *arXiv preprint arXiv:1503.00643*, 2015.

[43] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási, "The human disease network," *Proceedings of the National Academy of Sciences*, vol. 104, no. 21, pp. 8685–8690, 2007.

[44] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

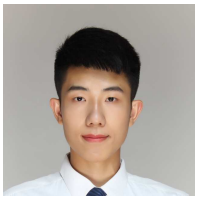
[45] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[46] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR (Poster)*. OpenReview.net, 2018.

[47] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *KDD*, 2019, pp. 257–266.

[48] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*. ACM, 2014, pp. 701–710.

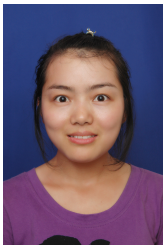
[49] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*. ACM, 2016, pp. 855–864.



Jintang Li received the bachelor's degree at Guangzhou Medical University, Guangzhou, China, in 2018. He is currently pursuing the master's degree with the School of Electronics and Communication Engineering, Sun Yat-sen University, Guangzhou, China. His main research interests include graph representation learning, adversarial machine learning, and data mining techniques.

Tao Xie received the bachelor's degree at Sun Yat-sen University, Guangzhou, China, in 2020. He is currently pursuing the master's degree with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His main research interests include recommendation systems, machine learning and data mining techniques.

Liang Chen received the bachelor's and Ph.D. degrees from Zhejiang University (ZJU) in 2009 and 2015, respectively. He is currently an associate professor with the School of Computer Science and Engineering, Sun Yat-Sen University (SYSU), China. His research areas include data mining, graph neural network, adversarial learning, and services computing. In the recent five years, he has published over 70 papers in sev- eral top conferences/journals, including SIGIR, KDD, ICDE, WWW, ICML, IJCAI, ICSOC, WSDM, TKDE, TSC, TOIT, and TII. His work on service recommendation has received the Best Paper Award Nomination in ICSOC 2016. Moreover, he has served as PC member of several top conferences including SIGIR, WWW, IJCAI, WSDM etc., and the regular reviewer for journals including TKDE, TNNLS, TSC, etc.



Fenfang Xie received the PhD degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2019. She is currently a postdoctoral fellow at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. Her research interests include recommender system, services computing, software engineering, machine learning and deep neural network.

Xiangnan He received the Ph.D. in Computer Science from the National University of Singapore (NUS) in 2016. His research interests span information retrieval, data mining, and multimedia analytics. He has over 70 publications that appeared in several top conferences such as SIGIR, WWW, and MM, and journals including TKDE, TOIS, and TMM. His work on recommender systems has received the Best Paper Award Honorable Mention in WWW 2018 and ACM SIGIR 2016. Moreover, he has served as the PC chair of CCIS 2019, area chair of MM (2019, 2020) ECML-PKDD 2020, and PC member for several top conferences including SIGIR, WWW, KDD, WSDM etc., and the regular reviewer for journals including TKDE, TOIS, TMM, etc.

Zibin Zheng received the Ph.D. degree from The Chinese University of Hong Kong, in 2011. He is currently a Professor with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. His research interests include services computing, software engineering, and blockchain. He received the ACM SIGSOFT Distinguished Paper Award at the ICSE'10, the Best Student Paper Award at the ICWS'10, and the IBM Ph.D. Fellowship Award.