Large-Scale Adversarial Attacks on Graph Neural Networks via Graph Coarsening

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) are fragile to adversarial attacks. However, existing state-of-the-art adversarial attack methods against GNNs are typically constrained by the graph's scale, failing to attack large graphs effectively. In this paper, we propose a novel attack method that attacks the graph in a divide-and-conquer manner to tackle large-scale adversarial attacks on GNNs. Specifically, the nodes are clustered based on node embeddings, coarsened graphs are constructed using the node clusters, and attacks are conducted on the coarsened graphs. Perturbations are selected starting with smaller coarsened graphs and progressing to larger detailed graphs while most of the irrelative nodes remain clustered, significantly reducing the complexity of generating adversarial graphs. Extensive empirical results show that the proposed method can greatly save the computational resources required to attack GNNs on large graphs while maintaining comparable performance on small graphs.

1 Introduction

Graphs are a versatile representation that can be utilized to model data from a wide range of disciplines, such as biology, chemistry, social networks, recommendation systems, and financial systems. Not surprisingly like Deep Neural Networks (DNNs), Graph neural networks (GNNs) on graph data act crucial roles in a number of emerging graph-based applications. GNNs exhibit their ability to extract valuable information from both the features (*i.e.*, information from individual node embedding) and the topology (*i.e.*, the relationship between nodes) in various applications.

Like DNNs, GNNs are also subject to adversarial attacks under both test-time (evasion) (Dai et al., 2018; Wu et al., 2019; Xu et al., 2019) and training-time (poisoning) (Zügner et al., 2018; Zügner & Günnemann, 2019) attacks. In this paper, we focus on a common node classification task (London & Getoor, 2014; Chapelle et al., 2006), which is of semi-supervise learning category: the whole graph, except for the labels for test nodes, is given during the training process. The semi-supervised learning setting contributes the attacker to generate enormous effects on GNNs. GNNs can improve classification by the message passing scheme (Kipf & Welling, 2017), exploiting the information in a node's neighborhood. This is a core strength but also a major vulnerability: due to the message propagation, an attacker can easily affect GNNs to change the prediction of a single node, even without changing any of its attributes or edges connected to it. Adversarial attacks on GNNs could result in real-world consequences; for example, unnoticed lending relationships can be forged to deceive the financial system, posing a serious systemic risk.

However, existing attack methods on GNNs suffer from a major complexity problem, making them difficult to use against large-scale graphs. The attack method must be memory- and time-efficient due to the large number of nodes in graph data. Existing work invariably falls short of at least one of the aforementioned concerns. Specifically, many existing methods are based on calculating gradients for all the edge entries. For example, PGD (Xu et al., 2019) maintains the large size of edge gradients in a dense matrix format, which results in a quadratic memory expense. For a graph with N nodes, a dense $N \times N$ gradient matrix is needed. A similar situation occurs to Mettack (Zügner & Günnemann, 2019), due to the implementation of meta-learning calculating meta-gradients for all $N \times N$ edge entries. Other methods, like RL-S2V (Dai et al., 2018) needs to train the attackers with reinforcement learning framework, which also consumes significant computational resources. Nettack (Zügner et al., 2018) suffers from the edge updates in an inefficient trial-and-error manner.

To overcome these limitations, it is crucial to simplify the graph while preserving the important properties of graph. One effective way is to coarsen the graph using a subset of the original node set, which involves contracting disjoint sets of connected vertices to reduce the number of nodes. Graph coarsening (Ruge & Stüben, 1987) has found various applications in graph partitioning (Hendrickson et al., 1995; Karypis & Kumar, 1998; Kushnir et al., 2006), visualization (Harel & Koren, 2000; Hu, 2005; Walshaw, 2000) and machine learning (Shuman et al., 2015). In this paper, we propose a novel graph coarsening method to achieve adversarial attack on GNNs with efficient computation. Our key insight is to divide a large graph into small graphs and attack those small graphs with a dividing-and-conquering strategy, as shown in Figure 1. For each perturbation, the proposed method will first cluster the nodes to build the coarsening graph, then select the most impacting edge entry based on the coarsening graph. Clusters related to the selected entry will be split and the coarsened graph will be rebuilt. The select-and-rebuild process will be repeated until the selected clusters are unsolvable, *i.e.*, the selected clusters only contain a single node, while most irrelative nodes remain clustered.

Our proposed method will reduce the complexity of GNNs attacks from $O(N^2)$ to $O\left(N\sqrt[l]{N}+M\right)$, where $l \geq 2$ is a hyperparameter, N and M are the number of nodes and edges. Empirical results on 8 datasets show the proposed method is able to attack large-scale graphs effectively and also perform compatibly to state-of-the-art methods on small-scale graphs.

2 Related Works

Adversarial attacks on machine learning models, unnoticeable perturbations deceives models, have been studied for a variety of model types. Unlike outliers, adversarial examples are purposefully constructed to be unnoticed in order to deceive machine learning models. Deep neural networks are also highly sensitive to small adversarial perturbations added to the data Szegedy et al. (2014); Goodfellow et al. (2015). The vast majority of attacks and defenses believe that data instances are independent and continuous, which is not the case for node classification different from many other graph-based tasks.

Recently, adversarial attacks for graphs start to attract attention from researchers. Torkamani & Lowd (2013) adopts associative Markov networks to improve robustness of collective classification with adversarial noise in the node features. In Chen et al. (2017), the modifications in the generated graph clustering are measured with injected noise to a bipartite graph representing DNS queries. Dai et al. (2018); Xu et al. (2019) take test-time (*i.e.,* evasion) attacks into consideration on node classification. However, poisoning (*i.e.,* training-time) attacks are more suitable for the semi-supervised setting of node classification and the transferability evaluation of their attacks is ignored. Also, the attacks are restricted to delete edge only and aiming at attacking the prediction of a single node. In nettack (Zügner et al., 2018), both evasion and poisoning attacks are designed as bilevel optimization problem on node classification models. The attacks are performed based on a surrogate model and the impact of those attacks are evaluated with a classifier trained on the modified data. This attack method can both add and remove edges, and manipulate node attributes using binary vectors, but only be suited to targeted attacks on single nodes. In Zügner & Günnemann (2019), meta-gradients are leveraged to attack the graph globally, making it possible to degrade the overall performance of models on node classification seriously.

There are only a few works developing efficient attack methods in terms of time complexity. Furthermore, given that most attack algorithms are gradient-based, how to reduce their memory complexity remains a challenge. The high complexity of attack methods has hindered their use in practical applications.

3 Approaches

3.1 Problem Formulation

We concentrate to attacking GNNs on the task of semi-supervised node classification. Given a single (attributed) graph and a set of labeled nodes, the goal of GNNs is to predict the labels of the unlabeled nodes. Let $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ be an attributed graph with adjacency matrix $\mathbf{A} \in \{0,1\}^{N\times N}$ and node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ where N is the number of nodes and D the dimension of the node feature vectors. Given the node IDs $\mathcal{V} = \{1, \ldots, N\}$ and the set of labeled nodes $\mathcal{V}_L \subseteq \mathcal{V}$, each node i is assigned to exactly one class c_i from K different classes. The goal of GNNs is to learn a function f_θ mapping each node $v \in \mathcal{V}$ to one of the K classes. Please note this is an instance of transductive learning that all the unlabeled nodes (*i.e.,* test samples) as well as their attributes and edges are known, except for the labels of the unlabeled nodes $\mathcal{V}_U = \mathcal{V}/\mathcal{V}_L$, and used during training (Chapelle et al., 2006). The parameters θ of the function f_θ are learned by minimizing a loss function $\mathcal{L}_{\text{train}}$ (*e.g.*, Cross-Entropy $\mathcal{L}_{\text{CE}}$) on $\mathcal{V}_L$ and corresponding labels $\mathbf{C}$:

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}\left(f_\theta\left(\mathcal{G}\right)\right) \quad \text{s.t.} \quad \mathcal{L}_{\text{train}}\left(f_\theta\left(\mathcal{G}\right)\right) = \mathcal{L}_{\text{CE}}\left(f_\theta\left(\mathcal{G}\right); \mathcal{V}_L, \mathbf{C}\right). \tag{1}$$

Attacker's Goal: Adversarial attacks are unnoticeable deliberate perturbations of data samples for desired outcome by the attacker applied to the machine learning models. Specifically in this work, the attacker's goal is to increase the misclassification rate of a node classification model achieved after training on the modified graph data $\hat{\mathcal{G}} = (\hat{\mathbf{A}}, \hat{\mathbf{X}})$. Following Zügner & Günnemann (2019) and in contrast to Zügner et al. (2018); Dai et al. (2018), the attacker is designed for *global* attacks reducing the overall classification performance of a model (*i.e.*, misclassify nodes in $\mathcal{V}_U$ as many as possible).

Attacker's Knowledge: In this work, we focus on gray-box attacks where the attacker has limited knowledge about the classification model. The attacker can access the graph data $\mathcal{G}$, labeled nodes $\mathcal{V}_L$ and the training labels $\mathbf{C}_{\mathcal{V}_L}$, but cannot access the model parameters θ and testing labels $\mathbf{C}_{\mathcal{V}_U}$. In other words, we are focusing on *poisoning* (training-time) attacks instead of *evasion* (test-time) attacks. To attack the victim model, the attacker can use a surrogate model to generate the modified graph data $\hat{\mathcal{G}}$. We use the popular GCN in Kipf & Welling (2017), the same as in Zügner et al. (2018); Zügner & Günnemann (2019), to be the surrogate model:

$$f_\theta(\mathbf{A}, \mathbf{X}) = \operatorname{softmax}\left(\tilde{\mathbf{A}}^2 \mathbf{X} \mathbf{W}_1 \mathbf{W}_2\right), \tag{2}$$

where $\tilde{\mathbf{A}} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}$, $\mathbf{D}$ the diagonal matrix of the node degrees and $(\mathbf{W}_1, \mathbf{W}_2)$ are weight parameters of the two-layer GCN.

Attacker's Capability: We focus on changing the graph structure only, because our algorithm can be easily adopted to attack the node features. Adversarial attacks should be unnoticeable to the original data. We follow Zügner et al. (2018)'s attacker capabilities that a budget constraint Δ is set to limit the number of changes $||\mathbf{A} - \hat{\mathbf{A}}||_0 \leq 2\Delta$, assuming the graph to be symmetric. We use $\Phi(\mathcal{G})$ to denote the set of all possible unnoticeable graph data.

Objective Function: We formulate the goal of our algorithm as an optimization problem:

$$\min_{\hat{\mathcal{G}} \in \Phi(\mathcal{G})} \quad \mathcal{L}_{\text{atk}}\left(f_{\theta^*}(\hat{\mathcal{G}})\right) \quad \text{s.t.} \quad \theta^* = \arg\min_\theta \quad \mathcal{L}_{\text{train}}\left(f_\theta(\hat{\mathcal{G}})\right). \tag{3}$$

$\mathcal{L}_{\text{atk}}$ is the loss function to decrease the generalization performance of the model on the unlabeled nodes, which is not able to be directly optimized without $\mathbf{C}_{\mathcal{V}_U}$. There are three options:

- *Train*: One option is to maximize the loss on labeled nodes $\mathcal{V}_L$, which means $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}}$. The reason for this option is, a model with a high training error is very likely to also generalize poorly on test set;
- *Self*: Another option is to use estimated labels. Recalling the semi-supervised setting of the task, $\mathcal{G}$ is known at training time. We can obtain estimated labels of $\hat{\mathbf{C}}$ with the surrogate model to compute the loss on the unlabeled nodes $\mathcal{V}_U$, which indicates $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{self}} = -\mathcal{L}_{\text{CE}}\left(f_\theta\left(\hat{\mathcal{G}}\right); \mathcal{V}_U, \hat{\mathbf{C}}\right)$;
- *Both*: It is also a good option to combine both *Train* and *Self*, which means $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}} - \mathcal{L}_{\text{self}}$.

3.2 Graph Structure Poisoning via Meta-Gradients

As formulated in Eqn. (3), the attacker aims to maximize the classification loss obtained after training the model parameters on the modified (poisoned) graph $\hat{\mathcal{G}}$, which is a highly challenging bi-level optimizing problem. Furthermore, graphs are discrete data, that using gradient-based methods such as gradient descent to make small perturbations on the graph data is not applicable. The possible decisions for the attacker include edge insertions and deletions and the attackers can make decisions on individual edge entry, which means N^2 possible decision for each perturbation. Hence, given a budget of Δ, the number of possible attacks is about $O\left(N^{2\Delta}\right)$, which indicates searching the best attack is impossible.

To tackle this problem, Zügner & Günnemann (2019) designed a greedy sequential decision making algorithm utilizing meta-gradients, which are widely used in meta-learning works (Thrun & Pratt, 1998; Naik & Mammone, 1992; Bengio, 2000; Bengio et al., 2013; Schmidhuber, 1992; Agostinelli et al., 2014; Finn et al., 2017). It treats the graph structure matrix as a hyperparameter and compute the gradient of $\mathcal{L}_{\text{atk}}$ after training the surrogate model with $\mathcal{L}_{\text{train}}$:

$$\nabla_{\mathcal{G}}^{\text{meta}} := \nabla_{\mathcal{G}} \mathcal{L}_{\text{atk}}\left(f_{\theta^*}(G)\right) \quad \text{s.t.} \quad \theta^* = \text{opt}_\theta\left(\mathcal{L}_{\text{train}}\left(f_\theta\left(\mathcal{G}\right)\right)\right). \tag{4}$$

$\text{opt}_\theta(\cdot)$ is a differentiable optimization procedure, like vanilla gradient descent with learning rate α starting from some initial parameters θ_0 and $\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}\left(f_{\theta_t}(\mathcal{G})\right)$ for T steps.

If we only perform changes to the graph structure $\mathbf{A}$, the node attributes $\mathbf{X}$ can be treated as a constant. To perform discrete updates, a score matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ is defined to assign each possible action a value indicating its estimated impact on the attacker objective $\mathcal{L}_{\text{atk}}$. Given the meta-gradient for a node pair $\{u, v\}$, we have:

$$\mathbf{S}(u, v) = \nabla_{\mathbf{A}(u,v)}^{\text{meta}} \cdot (-2 \cdot \mathbf{A}(u, v) + 1). \tag{5}$$

With $\mathbf{S}$, we can pick the most impacting edge to update the graph sequentially. For the K-th update, the entry with highest score is picked and the graph is modified by:

$$\mathcal{G}^{K+1} = \left(\text{flip}\left(\mathbf{A}^K, e'\right), \mathbf{X}\right) \quad \text{s.t.} \quad e' = \underset{e=(u,v):(\text{flip}(\mathbf{A}^K, e), \mathbf{X}) \in \Phi(\mathcal{G})}{\arg\max} \mathbf{S}^K(u, v). \tag{6}$$

flip$(\mathbf{A}^K, e)$ denoting function flipping the edge entry (*i.e.*, $\mathbf{A}^K(u, v) \leftarrow 1 - \mathbf{A}^K(u, v)$). $\mathcal{G}^K$ is the graph after K updates. $\mathcal{G}^0 = \mathcal{G}$. and the final modified graph $\hat{\mathcal{G}} = \mathcal{G}^\Delta$.

The computational complexity of this algorithm is $O(\Delta N^2)$, which is significantly improved from naïve searching algorithm. However, it costs more computational resources compared with downstream GNNs requiring $O(M)$ complexity, where M is the number of edges that most graphs are sparse so $M << N^2$.

3.3 Graph Structure Poisoning via Graph Coarsening

The main drawback of existing adversarial attack methods for graphs is the enormous decision space: $O(N^2)$ edges to be added or deleted, which is still not affordable for attacking large-scale graphs. To tackle this problem, we simplify the graph to small scales to reduce decision space. To this end, we contract the nodes into disjoint sets, which is similar to construct a "coarser" graph. We want to search the most impacting edge entry on the coarsen graph to reduce the decision space.

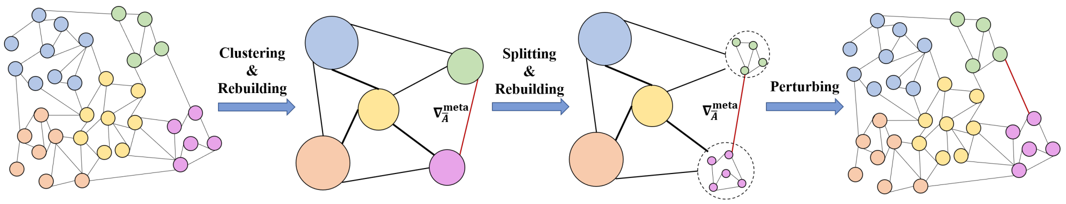
Clustering & Rebuilding
$\nabla_{\bar{A}}^{\text{meta}}$
Splitting & Rebuilding
$\nabla_{\bar{A}}^{\text{meta}}$
Perturbing

Figure 1: **Algorithm Overview:** Given the graph, our proposed method first *cluster* the nodes based on node embeddings. Then *rebuild* the coarsened graph with the weighted edges, averaged features and summed labels. After that the most impacting edge entry in the coarsened graph is *selected* calculated by meta-gradients. The selected clusters are *split* into new clusters, and the new coarsened graph is rebuilt. The *clustering*, *rebuilding*, *selecting* and *splitting* procedures are repeated until the selected edge entry is connecting two single nodes. Finally, the perturbation is added to the graph.

Illustration for the whole proposed algorithm is shown in Algorithm 1 and for a single perturbation step is shown in Figure 1. The proposed methods contains four basic procedures: *clustering*, *rebuilding*, *selecting* and *splitting*. We will introduce these four procedures in the following paragraphs.

Algorithm 1 Poisoning Attack on GNNs via Graph Coarsening

```
Require: Graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, Labeled nodes and corresponding labels $\mathcal{V}_L$, $\mathbf{C}_{\mathcal{V}_L}$. perturbation
    budget $\Delta$, meta gradients iterations $T$, number of clusters $K = \lfloor \sqrt[l]{|\mathcal{V}|} \rfloor$.
Ensure: Modified graph $\hat{\mathcal{G}} = (\hat{\mathbf{A}}, \mathbf{X})$.
 1: Train surrogate model with $\mathcal{G}$, $\mathbf{C}_{\mathcal{V}_L}$ and get $\hat{\theta}$. Use $\hat{\theta}$ to predict labels of unlabeled nodes $\hat{\mathbf{C}}_{\mathcal{V}_U}$.
 2: $\hat{\mathbf{A}} \leftarrow \mathbf{A}, \hat{\mathcal{G}} \leftarrow (\hat{\mathbf{A}}, \mathbf{X})$.
 3: while $||\hat{\mathbf{A}} - \mathbf{A}|| < 2\Delta$. do
 4:     Randomly initialize the surrogate model with $\theta_0$.
 5:     for $t$ in $0, \dots, T-1$ do
 6:         $\theta_{t+1} \leftarrow \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}} \left( f_{\theta_t}(\hat{\mathcal{G}}) \right)$        // Update surrogate model by gradient descent.
 7:     end for
 8:     Calculate node embeddings $\mathbf{E}$. Initialize coarsened node sets $\bar{\mathcal{V}}, i, j \leftarrow \{\mathcal{V}\}, 0, 0$
 9:     while $|\bar{v}_i| > 1$ or $|\bar{v}_j| > 1$ do
10:         $\bar{\mathcal{V}} \leftarrow (\bar{\mathcal{V}}/\{\bar{v}_i\}) \cup \text{kmeans}(\bar{v}_i, \mathbf{E}, K)$.        // *Splitting* and *Clustering*.
11:         $\bar{\mathcal{V}} \leftarrow (\bar{\mathcal{V}}/\{\bar{v}_j\}) \cup \text{kmeans}(\bar{v}_j, \mathbf{E}, K)$.        // Omit if $i = j$.
12:         *Rebuild* the graph $\bar{\mathcal{G}} = (\bar{\mathbf{A}}, \bar{\mathbf{X}})$ and labels $\bar{\mathbf{C}}_L$, $\bar{\mathbf{C}}_U$.
13:         Compute meta-gradients $\nabla^{\text{meta}}_{\bar{\mathbf{A}}}$ based on Eqn (10).
14:         *Select* the most impacting entry $(i, j)$ based on $\bar{\mathcal{G}}$ and $\nabla^{\text{meta}}_{\bar{\mathbf{A}}}$.
15:     end while
16:     Update $\hat{\mathcal{G}} \leftarrow \left( \text{flip} \left( \hat{\mathbf{A}}, (\bar{v}_i, \bar{v}_j) \right), \mathbf{X} \right)$.
17: end while
18: return $\hat{\mathcal{G}}$
```

Clustering: We will map the nodes $\mathcal{V}$ to a smaller set of nodes $\bar{\mathcal{V}}$ with a surjective map $\pi : \mathcal{V} \mapsto \bar{\mathcal{V}}$, where $|\bar{\mathcal{V}}| << |\mathcal{V}|$. Previous works on graph coarsening assume the graph is not attributed (*i.e.*, without features $\mathbf{X}$) and focus on preserving different properties. They are usually related to the spectrum of the original graph and coarse graph Loukas & Vandergheynst (2018); Loukas (2019); Bravo-Hermsdorff & Gunderson (2019); Cai et al. (2021). Since we have the node features $\mathbf{X}$, we can utilize it to construct the node map. K-Means method is applied to construct the node map π with the output embedding $\mathbf{E}$ from the first layer of the surrogate GCN (*i.e.*, $\mathbf{E} = \tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_1$). We define $\text{kmeans}(\mathcal{V}, \mathbf{E}, K)$ as a function that cluster the nodes in $\mathcal{V}$ into a set $\bar{\mathcal{V}}$ containing K clusters (*i.e.*, a set of nodes) $\{\bar{v}_1, \dots, \bar{v}_K\}$ based on embedding matrix $\mathbf{E}$, where $K < |\mathcal{V}|$. If $K >= |\mathcal{V}|$, $\text{kmeans}(\cdot)$ will return $\mathcal{V}$ itself.

Rebuilding: After constructing the node maps, we need to calculate the edge maps and features for the coarsened graph $\bar{\mathcal{G}} = (\bar{\mathbf{A}}, \bar{\mathbf{X}})$, and the labels $\bar{\mathbf{C}}$ of each cluster.

The feature vector $\bar{\mathbf{X}}(i, :)$ for cluster $\bar{v}_i$ equals the average features of all the nodes in the cluster:

$$\bar{\mathbf{X}}(i, :) = \frac{1}{|\bar{v}_i|} \sum_{j \in \bar{v}_i} \mathbf{X}(j, :); \tag{7}$$

The adjacency matrix $\bar{\mathbf{A}}$ is weighted and the weight for each entry:

$$\bar{\mathbf{A}}(i, j) = \sum_{p \in \bar{v}_i, q \in \bar{v}_j} \mathbf{A}(i, j); \tag{8}$$

The labels are weighted sum of the one-hot labels of the nodes, but are calculated separately for labeled nodes and unlabeled nodes, where:

$$\bar{\mathbf{C}}_L(i, :) = \sum_{j \in \bar{v}_i \cap \mathcal{V}_L} \mathbf{C}(j, :), \text{ and } \bar{\mathbf{C}}_U(i, :) = \sum_{j \in \bar{v}_i \cap \mathcal{V}_U} \mathbf{C}(j, :). \tag{9}$$

Selecting: With the coarsened graph $\bar{\mathcal{G}} = (\bar{\mathbf{A}}, \bar{\mathbf{X}})$, we reform Eqn. (3), by changing $\mathcal{L}_{\text{atk}}$ to soft label cross-entropy loss. *e.g.*, for the *Both* setting:

$$\mathcal{L}_{\text{atk}} = -\frac{1}{|\mathcal{V}_L|} \sum_{i \in \bar{\mathcal{V}}} \log \left(f_\theta \left(\bar{\mathcal{G}} \right) \right)(i, :) \cdot \bar{\mathbf{C}}_L(i, :) - \frac{1}{|\mathcal{V}_U|} \sum_{i \in \bar{\mathcal{V}}} \log \left(f_\theta \left(\bar{\mathcal{G}} \right) \right)(i, :) \cdot \bar{\mathbf{C}}_U(i, :). \tag{10}$$

Table 1: Misclassification rate (in %) of GCN with 1%, 5%, and 10% perturbed edges.

Attack	Cora			Citeseer			Polblogs		
Ptb Rate	1%	5%	10%	1%	5%	10%	1%	5%	10%
Clean	16.6 ± 0.3	16.6 ± 0.3	16.6 ± 0.3	28.5 ± 0.8	28.5 ± 0.9	28.5 ± 0.8	6.4 ± 0.5	6.4 ± 0.6	6.4 ± 0.5
DICE	16.6 ± 0.3	18.0 ± 0.4	19.5 ± 0.5	28.4 ± 0.3	28.9 ± 0.3	29.7 ± 0.3	7.7 ± 0.9	11.2 ± 1.1	14.4 ± 0.8
First-order	16.6 ± 0.3	17.2 ± 0.3	17.6 ± 0.5	28.2 ± 0.3	28.3 ± 0.3	28.2 ± 0.3	7.0 ± 0.8	7.8 ± 0.9	7.7 ± 0.6
Nettack	-	-	-	29.0 ± 0.4	31.9 ± 0.3	-	-	-	-
A-Meta	17.4 ± 0.4	21.8 ± 0.9	28.1 ± 1.1	29.1 ± 0.5	31.9 ± 0.7	34.3 ± 1.1	**13.7 ± 1.6**	19.8 ± 0.8	22.7 ± 0.7
Meta	17.0 ± 0.4	24.5 ± 1.0	**37.3 ± 1.4**	29.2 ± 0.5	34.6 ± 0.7	38.6 ± 1.0	11.4 ± 0.4	22.5 ± 0.8	**28.7 ± 3.6**
Ours	**18.6 ± 0.5**	**24.6 ± 0.8**	36.9 ± 1.1	**29.4 ± 0.4**	**34.9 ± 0.6**	**39.8 ± 0.9**	11.8 ± 0.8	**24.0 ± 1.1**	27.3 ± 0.9

For *Train* and *Self* settings, $\mathcal{L}_{\text{atk}}$ only contains the first or the second term in the last equation. With the new $\mathcal{L}_{\text{atk}}$, we follow the process in Eqn. (5, 6) to select the most impacting edge entry $(\bar{v}_i, \bar{v}_j)$.

Different from picking the entry from the original graph which is binary, edges in $\bar{\mathbf{A}}$ are weighted. We need to judge whether to add or delete edges for the picked entry and determine whether it is able to add or delete an edge between or inside corresponding clusters.

Similar to Eqn. (5), the choice to add or delete is determined by the maximum value of the score matrix calculated with meta-gradients $\nabla^{\text{meta}}_{\bar{\mathbf{A}}}$. If we want to add (*resp.*, delete) an edge for $\bar{A}(i,j)$, the score is $\nabla^{\text{meta}}_{\bar{\mathbf{A}}}(i,j)$ (*resp.*, $-\nabla^{\text{meta}}_{\bar{\mathbf{A}}}(i,j)$).

For an edge to be deleted, we only need to check if $\bar{\mathbf{A}}(i,j) > 0$. When we hope to add an edge to $\bar{\mathbf{A}}(i,j)$, it is more complicated: if $i \neq j$, $\bar{\mathbf{A}}(i,j) < |\bar{v}_i||\bar{v}_j|$ must be satisfied; If $i = j$, $\bar{\mathbf{A}}(i,j) < |\bar{v}_i|(|\bar{v}_i| - 1)/2$ is the condition before adding the edge.

Splitting: After selecting the edge entry, we will first check whether the selected entry is connecting two single nodes. If it is true, the perturbation will be updated to $\hat{\mathcal{G}}$ and the algorithm will start searching for another perturbation. If the edge entry is connecting two clusters with more than one nodes, we will split the selected clusters. Specifically for $\bar{v}_i$, we will remove it from $\bar{\mathcal{V}}$ and merge $\bar{\mathcal{V}}$ with $\text{kmeans}(\bar{v}_i, \mathbf{E}, K)$. After that the proposed method will *rebuild* the coarsened graph and *select* another edge entry.

Please note the proposed method will only select edge entries in the new graph related to the clusters added to the graph by the splitting procedure. e.g., if the selected entry is $\bar{v}_i$ and $\bar{v}_j$, the new selected entry must satisfy $\bar{v}'_i \in \text{kmeans}(\bar{v}_i, \mathbf{E}, K)$ and $\bar{v}'_j \in \text{kmeans}(\bar{v}_j, \mathbf{E}, K)$.

Complexity Analyses: Let the number of repeated times for *clustering*, *rebuilding*, *selecting* and *splitting* be l. The complexity of GCN is $O(Mdh)$, where M is the number of edges, d is the number of features and h is the size of hidden layer.

- *Clustering* procedure only contains a K-Means function, so the complexity of it is $O(NKd)$.
- *Rebuilding* procedure, calculating averaged features requires $O(Nd)$ complexity, while for calculating the weighted adjacency we can add the weights edge by edge requiring $O(M)$ complexity.
- *Selecting* procedure is done on the coarsened graph which contains no more than $2Kl$ clusters, hence, it requires $O(K^2 l^2)$ complexity.
- *Splitting* is a set operation requiring $O(N)$ complexity.

Overall, after omitting h, d and l which are relatively small to N and M, complexity of the whole algorithm is $O(NK + M)$. In practice we found K-Means always tends to cluster the nodes averagely without any regularization, so that we can roughly take $K \approx \sqrt[l]{N}$. Hence, taking l as the hyperparameter of the proposed method, we set $K = \lfloor \sqrt[l]{N} \rfloor$. The complexity of the proposed method is $O\left(N\sqrt[l]{N} + M\right)$.

Table 2: Misclassification rate (in %) / running time (in minutes) of the proposed method on large scale datasets.

Ptb.Rate (%)	Pubmed	Amazon	CS	Physics	Arxiv
0	13.8±0.3 / -	12.0±0.2 / -	7.3±0.3 / -	4.4±0.1 / -	34.6±1.1 / -
1	14.4±0.3 / 43.4	15.3±0.3 / 143.1	9.4±0.4 / 68.3	5.0±0.2 / 182.3	49.2±0.9 / 473.2
5	16.7±0.2 / 198.9	17.4±0.3 / 673.2	12.1±0.2 / 277.5	5.4±0.2 / 820.1	52.8±1.0 / 1823.8
10	19.2 ± 0.4 / 402.0	21.8±0.4 / 1347.1	15.1±0.3 / 562.9	6.9±0.3 / 1621.5	57.9±1.2 / 3712.9

4 Experiments

4.1 Settings

Datasets: we evaluate our method on different scale datasets from the well-known small-scale datasets including Cora (McCallum et al., 2000), Citeseer (Giles et al., 1998; Sen et al., 2008), Polblogs (Adamic & Glance, 2005) datasets to large-scale datasets including Pubmed (Sen et al., 2008), Amazon (Shchur et al., 2018), CS (Shchur et al., 2018), Physics (Shchur et al., 2018), and Arxiv (Wang et al., 2020) datasets. Each dataset is split into labeled (10%) and unlabeled (90%) nodes. The labels of the unlabeled nodes are only used for the evaluation of generalization performance among different models. We report the statistics summary of those datasets in Appendix A.2.

Implementation details: Following Zügner & Günnemann (2019), we train node classification models on the modified (poisoned) data to evaluate the effectiveness of adversarial attacks. In our experiments, we adopt Graph Convolutional Networks (GCN) (Kipf & Welling, 2017) as the node classification model, which utilizes the message passing framework (*a.k.a.* graph convolution) and is trained in a semi-supervised way. We repeat all of our attacks on five different splits of labeled/unlabeled nodes and train all target classifiers ten times per attack (using the split that was used to create the attack). Uncertainty indicates 95 % confidence intervals of the mean obtained via bootstrapping is reported with every classification accuracy result. The detailed choice of α, l, T and the settings of $\mathcal{L}_{\text{atk}}$ are shown in Appendix A.3.

Baselines: We report the results of various state-of-the-art poisoning attack methods on graphs for comparison:

- DICE ("delete internally, connect externally") $O(M)$: DICE is a baseline that has all true class labels (train and test) available and thus more knowledge than all competing methods. For each perturbation, DICE randomly chooses whether to insert or remove an edge. Edges are only removed between nodes from the same class, and only inserted between nodes from different classes.
- First-order $O(N^2)$: this baseline is proposed in Finn et al. (2017), which ignores all second-order derivatives to attack the graph.
- Nettack $O(N^3)$: Nettack (Zügner et al., 2018) is a targeted attack method. In our experiments, one target node is randomly selected from the unlabeled nodes for each perturbation and the graph is attacked considering all nodes in the network.
- Meta $O(N^2)$: In Zügner & Günnemann (2019) proposed a meta-gradient method based on greedy meta-gradients selection, which is described in Section 3.2.
- A-Meta $O(N^2)$: A-Meta is a variation approach to Meta proposed in Zügner & Günnemann (2019), which approximately calculates the meta-gradients similar to the method proposed in Nichol et al. (2018).

4.2 Comparison to Other Attack Methods

Evaluation on Small-Scale Datasets: Let's look at how the proposed method performs on small-scale datasets before evaluating it on large-scale datasets. In Table 1, we exhibit the misclassification rates of all state-of-the-art attack methods and our proposed method on three datasets Cora, Citeseer and Polblogs. The largest dataset among the selected datasets is Cora, which has 2485 nodes in total. Perturbation rate varies in $[1\%, 5\%, 10\%]$.

Table 3: Running details of the proposed method with $l = 3$.

Dataset	Ptb.Rate (%)	Add	Delete	Loop	$\lvert\bar{\mathcal{V}}\rvert$	Time (min.)	Mis.Rate(%)
Citeseer	0	0	0	-	-	-	28.5 ± 0.8
	1	32.3	3.7	3.19	31.7	0.6	29.4 ± 0.4
	5	171.8	11.2	3.22	31.8	2.7	34.9 ± 0.6
	10	334.4	31.6	3.22	31.8	5.5	39.8 ± 0.9
Pubmed	0	0	0	-	-	-	13.8 ± 0.3
	1	351.5	91.5	3.36	75.0	9.1	14.4 ± 0.3
	5	1463.9	752.1	3.39	75.7	48.8	16.7 ± 0.2
	10	2667.2	1764.8	3.36	75.4	94.5	19.2 ± 0.4

Table 4: Ablative studies for l. Perturbation rate is 5%.

Dataset	l	Add	Delete	Loop	$\lvert\bar{\mathcal{V}}\rvert$	Time (min.)	Mis.Rate(%)
Citeseer	1	180.8	2.2	1.00	2110	10.9	32.3 ± 0.6
	2	141.2	41.8	2.04	69.0	3.7	34.4 ± 0.7
	3	171.8	11.2	3.22	31.8	2.7	34.9 ± 0.6
	4	172.7	10.3	4.36	21.0	2.8	33.5 ± 0.8
Pubmed	1	-	-	-	-	-	-
	2	1636.8	579.2	2.25	241.6	168.9	16.0 ± 0.3
	3	1463.9	752.1	3.39	75.7	48.8	16.7 ± 0.2
	4	1408.4	807.6	4.49	41.6	47.3	16.6 ± 0.2

Nettack failed to output valid results in three days for some cases due to its $O(N^3)$ complexity. DICE, First-Order, and Nettack, among all the state-of-the-art approaches, are too weak to deceive GNNs into misclassifying more nodes, as seen in the table. Meta and A-Meta are dominating among these methods. However, our proposed method offers competitive results. With most of the cases, our proposed method performs slightly better while with some of the cases it performs slightly weaker compared to Meta and A-Meta. These findings show that the proposed strategy is capable of attacking GNNs on small-scale graphs.

Evaluation on Large-Scale Datasets: Table 2 shows the performance and running time of our proposed method on five large-scale datasets. The size of these datasets varies from 13752 to 169343 (details can be found in Appendix A.2). Perturbation rate varies in $[1\%, 5\%, 10\%]$.

The proposed approach can effectively trick GNNs to misclassify nodes on large graphs, as shown in table. *Please note that state-of-the-art methods First-Order, Nettack, Meta and A-Meta all fail to attack these large-scale graphs due to out-of-memory or out-of-time in three days.* Because these methods require complexity $O(N^2)$ or higher. DICE can finish the attack on time with little consumed memory, but it needs more knowledge (the labels for unseen nodes) for attacking and the performance is relatively poorer compared to our proposed method (please refer to Table 8 in Appendix).

4.3 Ablative Studies

Statistics of the Proposed method: In Table 3, we exhibit the critical statistics for the proposed method. We select Citeseer and Pubmed dataset and vary the perturbation rate in $[1\%, 5\%, 10\%]$, then show the average number of edges added ("add"), deleted ("delete"), the average executed times ("Loop") of the while loops (in Algorithm 1 from line 9 to 15) per perturbation, the average size of clusters in the coarsened graph $\bar{\mathcal{V}}$ in Algorithm 1 ("$|\bar{\mathcal{V}}|$"), the running time per attack ("Time") and the corresponding misclassification rates. l is set to be 3 for Algorithm 1.

The proposed method adds more edges than it removes (Meta (Zügner & Günnemann, 2019) encounters similar situations). K-Means always clusters the nodes averagely, as we said at the end of Section 3.3. This statement is supported by the fact that the number of loops is extremely close to l, in the range of $1.2l$. In addition, the coarsened graph's size is consistent to varied perturbation rates.

Table 5: Ablative studies for different settings of our proposed method.

Dataset	Ptb.Rate (%)	Clean	$\mathbf{E} = \mathbf{X}$	Logit	*Train*	*Self*	*Both*
Citeseer	1	28.5 ± 0.8	28.7 ± 0.4	28.6 ± 0.5	28.9 ± 0.3	29.2 ± 0.3	**29.4 ± 0.4**
	5	28.5 ± 0.8	32.9 ± 0.5	34.2 ± 0.7	34.8 ± 0.6	33.9 ± 0.9	**34.9 ± 0.6**
	10	28.5 ± 0.8	35.8 ± 1.2	37.3 ± 0.9	39.5 ± 0.6	37.8 ± 1.3	**39.8 ± 0.9**
Pubmed	1	13.8 ± 0.3	14.0 ± 0.4	14.3 ± 0.3	14.1 ± 0.4	**14.4 ± 0.3**	14.3 ± 0.3
	5	13.8 ± 0.3	16.0 ± 0.2	16.6 ± 0.2	16.4 ± 0.3	**16.7 ± 0.2**	16.5 ± 0.3
	10	13.8 ± 0.3	19.0 ± 0.4	19.2 ± 0.5	18.8 ± 0.4	**19.2 ± 0.4**	18.9 ± 0.5

Ablative Studies for l: l in Algorithm1 is a critical hyperparameter that it determines the complexity of the proposed method. In Table 4, we show the statistics for the proposed method with different choices of l. We select Citeseer and Pubmed dataset and the perturbation rate is 5%. l varies form 1 to 4. When $l = 1$, the proposed method is the same as Meta.

From Table 4 we can infer that when l becomes larger, the size of coarsened graph reduces, the number of loops increases. The running time also reduces when l becomes larger, but when $l > 3$ the reduction is not significant. Hence, we set $l = 3$ for most of the situations, except for some small graphs or larger graphs like Arxiv (please refer to Appendix A.3 for more details).

When $l = 1$, the proposed method fails to finish the attack due to out-of-memory. Although when $l > 3$ the benefit of larger l on time saving is not significant, it helps to reduce memory resources consumed by the size of coarsened graphs, making the proposed method possible to attack huge graphs avoiding out-of-memory issues.

Choices of the Embeddings for K-Means: As we mentioned in Section 3.3, we use the output from the first layer of the surrogate GCN as the node embeddings for clustering nodes with K-Means approach. The reason for this choice is, the surrogate model only contains two layers, size of the output of the last layer (*i.e.*, logits of the model) may be small due to the number of classes for each graph. While if we use the input node feature $\mathbf{X}$ as node embeddings to build the coarsened graph, the adjacency information $\mathbf{A}$ is ignored.

In Table 5, we show the results of using $\mathbf{X}$ ("$\mathbf{E} = \mathbf{X}$") or the logits of the surrogate model ("Logit") for clustering procedure. We select Citeseer and Pubmed dataset and vary the perturbation rate in $[1\%, 5\%, 10\%]$. Misclassification rates are reported. Compared with the best results among *Train*, *Self* and *Both* which use the output of the first layer to cluster the nodes, performance for these settings is worse.

Choices of $\mathcal{L}_{\text{atk}}$: The choice of $\mathcal{L}_{\text{atk}}$ for the proposed method is also important. If we select *Train* setting, the attacker will update the graph to increase the loss of the surrogate model on the training set, hoping the surrogate model generalizes poorly on the unlabeled nodes. While if we select the *Self* setting, the attacker will directly concentrate on changing the predictions of the surrogate model on the unlabeled nodes.

In Table 5, we show the results of different choices of $\mathcal{L}_{\text{atk}}$. We select Citeseer and Pubmed dataset and vary the perturbation rate in $[1\%, 5\%, 10\%]$. We find that *Both* is the best choice for Citeseer while for Pubmed the best one is *Self.* Although there are differences in the results for different choices, the gaps are not big.

5 Conclusion

In this paper, we propose a novel attack method that attacks the graph in a divide-and-conquer manner. The proposed method clusters nodes with node embeddings, constructs coarsened graphs using the node clusters, and searches attacks based on the coarsened graphs. The proposed method selects perturbations from smaller coarsened graphs to larger detailed graphs with most of the irrelative nodes clustered, significantly reducing the complexity of attacking graphs. Extensive empirical results show that the proposed method can greatly save the computational resources required to attack GNNs on large graphs while maintaining comparable performance on small graphs.

6 Ethics Statement

GNNs can be utilized to a wide range of applications, such as biology, chemistry, social networks, recommendation systems, and financial systems. Adversarial attacks on GNNs could result in real-world consequences; for example, unnoticed lending relationships can be forged to deceive the financial system, posing a serious systemic risk. One potential negative impact is to apply the proposed method to real-world applications. But on the other hand, positive impacts of the proposed method is we can train new GNNs which immune the proposed attack method to improve robustness.

All the datasets used in this paper are public for academic use.

7 Reproducibility Statement

We provide the source code of the proposed method in the supplementary materials. Detailed settings and hyperparameters of the proposed method are listed in Appendix A.3.

References

Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Link Discovery (workshop)*, 2005.

Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gescei. On the optimization of a synaptic learning rule. In *Optimality in Biological and Artificial Networks?* 2013.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8): 1889–1900, 2000.

Gecia Bravo-Hermsdorff and Lee M Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. In *NeurIPS*, 2019.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *ICLR*, 2021.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. Semi-supervised learning. adaptive computation and machine learning series, 2006.

Yizheng Chen, Yacin Nadji, Athanasios Kountouras, Fabian Monrose, Roberto Perdisci, Manos Antonakakis, and Nikolaos Vasiloglou. Practical attacks against graph-based clustering. In *CCS*, 2017.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *ICML*, 2018.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

C Lee Giles, Kurt Bollacker, and Steve Lawrence CiteSeer. An automatic citation indexing system. In *Digital Libraries*, 1998.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *stat*, 1050:20, 2015.

David Harel and Yehuda Koren. A fast multi-scale method for drawing large graphs. In *GD*, 2000.

Bruce Hendrickson, Robert W Leland, et al. A multi-level algorithm for partitioning graphs. *SC*, 95 (28):1–14, 1995.

Yifan Hu. Efficient, high-quality force-directed graph drawing. *Manuscr Math*, 10(1):37–71, 2005.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput*, 20(1):359–392, 1998.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Dan Kushnir, Meirav Galun, and Achi Brandt. Fast multiscale clustering and manifold identification. *R*, 39(10):1876–1891, 2006.

Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. DeepRobust: A PyTorch library for adversarial attacks and defenses. *CoRR*, abs/2005.06149, 2020.

Ben London and Lise Getoor. Collective classification of network data. *Data Classification: Algorithms and Applications*, 399, 2014.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, 20(116): 1–42, 2019.

Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *ICML*, 2018.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Devang K Naik and Richard J Mammone. Meta-neural networks that learn by learning. In *IJCNN*, 1992.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *NeurIPS*, 2019.

John W Ruge and Klaus Stüben. Algebraic multigrid. In *Multigrid methods*. 1987.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Trans. Signal Process*, 64(8):2119–2134, 2015.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. 1998.

MohamadAli Torkamani and Daniel Lowd. Convex adversarial collective classification. In *ICML*, 2013.

Chris Walshaw. A multilevel algorithm for force-directed graph drawing. In *GD*, 2000.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1): 396–413, 2020.

Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In *IJCAI*, 2019.

Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: An optimization perspective. *arXiv preprint arXiv:1906.04214*, 2019.

Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *ICLR*, 2019.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *SIGKDD*, 2018.

A Appendix

A.1 Implementation Environments

We use PyTorch (Paszke et al., 2019) to implement the proposed method with PyTorch-Geometric (Fey & Lenssen, 2019) and DeepRobust (Li et al., 2020) packages. The experiments are run on NVidia TITAN RTX GPU, Intel(R) Xeon(R) Silver 4116 CPU 2.10GHz, 192G RAM.

Table 6: Dataset Statistics.

Datasets	Nodes	Edges	Classes	Features	Feature Type
Cora	2485	5069	7	1433	Binary
Citeseer	2110	3668	6	3703	Binary
Polblogs	1222	16714	2	-	-
Amazon	13752	491722	10	767	Binary
CS	18333	163788	15	6805	Binary
Pubmed	19717	44338	3	500	Continuous
Physics	34493	495924	5	8415	Binary
Arxiv	169343	1166243	10	767	Continuous

A.2 Dataset Statistics

In this paper, we evaluate the performance of our method on eight datasets, which are summarized in Table 6.

- Cora (McCallum et al., 2000) and Citeseer (Giles et al., 1998; Sen et al., 2008) are two real-world bibliographic datasets. The Cora (*resp.,* CiteSeer) dataset contains a number of machine-learning papers divided into one of 7 (*resp.,* 6) classes. The final corpus of Cora (*resp.,* CiteSeer) dataset has 2708 (*resp.,* 3312) documents, 1433 (*resp.,* 3703) distinct words in the vocabulary, and 5429 (*resp.,* 4732) links. We choose largest connected component of Cora (*resp.,* CiteSeer) dataset in our experiments.

- Polblogs (Adamic & Glance, 2005) dataset is collected from the top 20 conservative leaning blogs and the top 20 liberal leaning blogs and contains 12470 posts from the left leaning set of blogs and 10414 posts from the right leaning set.

- Pubmed (Sen et al., 2008) dataset consists of 19717 scientific publications from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44338 links. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.

- Amazon (Shchur et al., 2018) dataset is based on the Amazon co-purchase graph, where nodes represent goods, edges indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category. We select the "Computers" collection of this dataset.

- CS and Physics datasets (Shchur et al., 2018) are based on the Microsoft Academic Graph from the KDD Cup 2016 challenge, nodes are authors, that are connected by an edge if they co-authored a paper; node features represent paper keywords for each author's papers, and class labels indicate most active fields of study for each author.

- Arxiv (Wang et al., 2020) dataset is a directed graph, representing the citation network between all Computer Science (CS) arXiv papers. Each node is an arXiv paper and each directed edge indicates that one paper cites another one. Each paper comes with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract.

Table 7: Hyperparameters for each dataset.

Datasets	l	α	T	$\mathcal{L}_{atk}$
Citeseer	3	0.1	10	Both
Cora	2	0.1	20	Train
Polblogs	2	0.1	100	Self
Amazon	3	0.1	5	Both
CS	3	0.1	5	Both
Pubmed	3	0.1	10	Self
Physics	3	0.3	1	Both
Arxiv	4	0.3	1	Both

Table 8: Misclassification rate (in $\%$) for DICE on large scale datasets.

Ptb.Rate ($\%$)	Pubmed	Amazon	CS	Physics	Arxiv
0	13.8±0.3	12.0±0.2	7.3±0.3	4.4±0.1	34.6±1.1
1	14.0±0.2	15.2±0.4	7.4±0.4	4.5±0.4	38.1±0.8
5	15.7±0.2	17.2±0.2	8.0±0.3	4.8±0.2	41.7±1.1
10	17.3±0.4	20.7±0.3	8.5±0.5	5.8±0.2	43.2±1.2

A.3 Hyperparameter Settings for Different Datasets

In this section, we exhibit the hyperparameters l, α, T and the choice of $\mathcal{L}_{\text{atk}}$ in Algorithm 1 for each dataset. The details can be found in Table 7.

A.4 Results of DICE on Large-Scale Datasets

In Table 2, we show the performance of our proposed method on large-scale datasets. In Table 8, we show the corresponding results of the baseline method DICE. DICE can finish the attack on time consuming little memory, but it needs more knowledge (the labels for unseen nodes) for attacking and the performance is relatively poorer compared to our proposed method.