EdgePruner: Poisoned Edge Pruning in Graph Contrastive Learning

1st Hiroya Kato
KDDI Research, Inc., Japan

2nd Kento Hasegawa
KDDI Research, Inc., Japan

3rd Seira Hidano
KDDI Research, Inc., Japan

4th Kazuhide Fukushima
KDDI Research, Inc., Japan

Abstract*—Graph contrastive learning (GCL) is unsupervised graph representation learning that can obtain useful representation of unknown nodes. The node representation can be utilized as features of downstream tasks. However, GCL is vulnerable to poisoning attacks as with existing learning models. A state-of-the-art defense cannot sufficiently negate adverse effects by poisoned graphs although such a defense introduces adversarial training in the GCL. To achieve further improvement, pruning adversarial edges, which can also be called poisoned edges is important. To the best of our knowledge, the feasibility remains unexplored in the GCL domain. In this paper, we propose a simple defense in GCL, *EdgePruner*. We focus on the fact that the state-of-the-art poisoning attack on GCL tends to mainly add adversarial edges to create poisoned graphs, which means that pruning edges is important to sanitize the graphs. Thus, EdgePruner prunes edges that contribute to minimizing the contrastive loss based on the node representation obtained after training on poisoned graphs by GCL. Furthermore, we focus on the fact that nodes with distinct features are connected by adversarial edges in poisoned graphs. Thus, we introduce feature similarity between neighboring nodes to help more appropriately determine adversarial edges. This similarity is helpful in further eliminating adverse effects from poisoned graphs on various datasets. Finally, EdgePruner outputs a graph that yields the minimum contrastive loss as the sanitized graph. Our results demonstrate that pruning adversarial edges is feasible on six datasets. EdgePruner can improve the accuracy of node classification under the attack by up to 5.55% compared with that of the state-of-the-art defense. Moreover, we show that EdgePruner is immune to an adaptive attack.

***Index Terms*—graph representation learning, contrastive learning, poisoning attack, edge pruning, graph sanitization**

I. Introduction

The graph representation learning plays an important role in utilizing information from graph structured data such as social networks [1], [2] and e-commercial networks [3], [4] for the various purposes including community detection [5], [6] and recommendation systems [7]–[10]. However, most existing graph representation learning methods are executed in a supervised or semi-supervised manner, which requires plentiful labeled data for training. Unfortunately, the node labels of graph structured data are insufficient in quantity [11]. This is because manually labeling nodes in large graphs is rarely realistic in practical situations. Thus, unsupervised learning methods for graph structured data are needed. In an early stage of research, traditional unsupervised methods such as DeepWalk [12] and node2vec [13] yielded relatively insufficient performance compared with that of supervised methods. However, recently, the classical information maximization (InfoMax) principle [14] has attracted renewed attention, and the contrastive learning has achieved great success in many fields such as computer vision [15]–[21] and natural language processing [22]–[27]. In particular, the breakthroughs of the contrastive learning in computer vision have motivated researchers to apply the similar techniques from visual representation learning to graph representation learning [28]. To learn node representation in a graph without labels, several contrastive learning methods for graphs have been proposed. In particular, such methods are called *graph contrastive learning (GCL)* in this paper. Recent GCL achieves comparable performance with supervised methods by introducing various techniques, including different view generation via stochastic augmentations and a well-considered contrastive loss. The main goal of GCL is to learn an encoder that maps nodes in a graph into low-dimensional numerical vectors (the vectors are called embeddings). Useful embeddings can be acquired by training the encoder so that only similar nodes in the input space are close in the embedding space. The embeddings can be utilized as features for many downstream tasks such as node classification. Thus, GCL is helpful in obtaining useful representation of unseen nodes.

However, a recent study [29] shows that the GCL models are vulnerable to poisoning attacks. To make matters worse, the state-of-the-art attack called CLGA [29] can degrade the performance of many downstream tasks by contaminating embeddings output from an encoder trained by GCL. Thus, CLGA may cause a severe obstacle that prevents GCL from being widely utilized. As a state-of-the-art defensive method in the GCL domain, an effective GCL method called ARIEL [28] has been proposed recently. In ARIEL, another view called the adversarial view is introduced so as to make the GCL model robust to poisoning attacks. As a result, ARIEL can improve the quality of the embeddings based on poisoned graphs. However, ARIEL only tries to learn a robust model when a poisoned graph is given. Accordingly, ARIEL achieves suboptimal performance on poisoned graphs, which means that there is still room for improvement to the quality of the embeddings. To achieve further improvement, we argue that sanitizing the poisoned graph is needed. Sanitizing means pruning adversarial edges, which can also be called poisoned edges in a graph in this paper. To the best of our knowledge, there has been no study that explores the feasibility of sanitizing poisoned graphs in the GCL domain.

In this paper, we propose a simple defense called *EdgePruner*. We focus on the fact that the state-of-the-art attack on GCL, namely CLGA tends to mainly add adversarial edges to create poisoned graphs. On this basis, we consider that the pruning edges is important to sanitize poisoned graphs. EdgePruner prunes edges that contribute to minimizing the contrastive loss on the basis of gradients of the loss. By doing this, sanitizing poisoned graphs can be expected. Furthermore, to help determine more appropriate edges to prune, we focus on the fact that adversarial edges tend to connect nodes for which the features are dissimilar in poisoned graphs, which is shown in other work [30], [31]. This is because connecting such nodes is effective in confusing the relationship between neighboring nodes in a graph from the perspective of attacks. Thus, in light of this tendency, we introduce feature similarity between neighboring nodes as additional information to determine adversarial edges connecting nodes whose features are distinct. This similarity is helpful in further eliminating adverse effects from poisoned graphs on various datasets, which further improves the quality of the embeddings. Finally, EdgePruner outputs a graph that yields the minimum contrastive loss as the sanitized graph. The sanitized graphs are fed into GCL methods for training the encoder.

Our contributions. The main contributions of this work are as follows:

1) We propose a simple and effective pruning method against poisoning attacks for the GCL. Our pruning method sanitizes poisoned graphs by pruning edges that contribute to minimizing the contrastive loss. We formulate our pruning method as the optimization problem.
2) We introduce feature similarity between neighboring nodes to prune adversarial edges connecting nodes with distinct features. We also formulate our pruning method with the feature similarity as the optimization problem. We show that the similarity is helpful in further eliminating adverse effects from poisoned graphs on various datasets.
3) We conduct extensive experiments to demonstrate the effects of EdgePruner for clean graphs and poisoned graphs created by the CLGA and an adaptive attack.
4) Our experimental results demonstrate that EdgePruner can eliminate the detrimental effects from the poisoned graphs on six datasets while maintaining acceptable accuracies on clean graphs. In particulr, EdgePruner can improve accuracy by up to 9.60% compared with that of a GCL method without defense. Moreover, we show that EdgePruner is immune to an adaptive attack. To the best of our knowledge, this work first shows the feasibility of pruning adversarial edges in poisoned graphs in the GCL domain.

II. Related Work

Most existing graph representation learning methods are supervised or semi-supervised ones [32]–[36]. However, the node labels of large graphs in the real world are difficult to obtain [11]. This is because manually labeling nodes in such graphs is rarely realistic in practical situations. Therefore, in the graph domain, several GCL methods, which can learn graph representation without labels have been proposed. The main goal of GCL is to learn an encoder that converts nodes in a graph into low-dimensional embeddings without labels. For example, as an encoder, a two-layer graph convolutional network (GCN) [35] is utilized. An encoder is trained so that only similar nodes in the input space are close in the embedding space. The embeddings produced by the trained encoder can be utilized as features for downstream tasks such as node classification and link prediction.

A. Graph Contrastive Learning

To learn node representation in a graph without labels, several GCL models have been proposed [37]–[42]. Velivckovic et al. [37] propose deep graph InfoMax (DGI), which is a general approach for unsupervised graph learning based on mutual information, rather than random walks. DGI maximizes mutual information between patch representations and corresponding high-level summaries of graphs derived by using graph convolutional network architectures. The patch representations are vectors that express local information about the graph centered around a node rather than just the node. According to the results in [37], DGI yields higher performance on downstream tasks compared with that of a traditional unsupervised method, DeepWalk [12].

To supplement the input graph with more global information, Hassani and Khasahmadi propose contrastive multi-view graph representation learning (called MVGRL in this paper) [38]. In MVGRL, graph diffusion is introduced into the GCL approach, and graph views are obtained by uniformly sampling subgraphs. Then, MVGRL contrasts node representations to global embeddings across the two views. As a result, MVGRL outperforms DGI. You et al. propose GraphCL [39] and design four types of graph augmentations, namely, node dropping, edge perturbation, attribute masking, and sampling subgraphs. The impact of various combinations of these graph augmentations on multiple datasets is systematically studied. Qiu et al. propose the graph contrastive coding (GCC) framework to learn structural representations across graphs [40]. GCC aims to distinguish between subgraphs sampled from a certain node and subgraphs sampled from other nodes. Since GCC does not assume that nodes and subgraphs come from the same graph, the graph encoder is designed to capture universal patterns across different input graphs. In other words, the pretrained GCC model can be applied to unseen graphs for downstream tasks such as node classification. Zhu et al. propose GRACE [41], which generates two correlated graph views by randomly performing corruption. The model is trained by using a contrastive loss to maximize the agreement between node embeddings in these two views. They also consider random augmentation at both topological and node attribute levels so as to accelerate optimization of the contrastive loss. The augmentation includes removing edges and masking features to provide diverse contexts for nodes in different views. Zhu et al. propose graph contrastive learning with adaptive augmentation (GCA) [42]. GCA introduces adaptive augmentation that perturbs both node features and edges in accordance with their importance. They argue that augmentation schemes used in the existing methods suffer from suboptimal performance because most existing methods adopt uniform data augmentation schemes, such as uniformly dropping edges and uniformly shuffling features. On the topological level, new augmentation schemes based on node centrality measures are designed to identify important edges. Furthermore, noise is added to node attributes by randomly masking some node features with zeros. As a result, GCA further improves GRACE.

B. *Poisoning Attack on GCL*

Although GCL is promising for learning useful representation of unseen nodes, a recent study [29] shows that the GCL models are also vulnerable to poisoning attacks as with existing learning methods. Almost all of the existing attacks are mainly intended for supervised learning methods such as GCN. In other words, they are supervised attacks and require labels to create poisoned graphs. PGD and MinMax [43] optimize the negative cross-entropy loss on the basis of gradients. Nettack [44] iteratively selects edges to alter by calculating the score of each possible alteration. Mettack [45] uses the gradient of the classification loss with respect to the adjacency matrix to select the edges to change.

However, in real-world scenarios, it is difficult to attack GCL with the abovementioned supervised attacks because the labels of large graphs are difficult to acquire. This motivated researchers to demonstrate whether unsupervised attacks[1] targeting at GCL are feasible without labels or not. Bojchevisk and Stephan propose a graph poisoning attack based on random walks [46]. They devise efficient adversarial perturbations that poison the network structure of graphs. Their poisoning attack has a negative effect on both the quality of the node embeddings and the downstream tasks. However, their attack cannot work well for GCA that is the latest GCL model because that attack relies only on random walks. Zhang et al. [29] propose an unsupervised attack based on gradient ascent on the GCL for node embeddings, which is called CLGA. CLGA is intended for GCL and computes the gradient of the contrastive loss w.r.t. the adjacency matrix. After that, the edges with the largest gradients are flipped (deletion or addition). According to the experiments in [29], CLGA outperforms the existing unsupervised attack [46]. In addition to that, CLGA has comparable performance with some of the existing supervised attacks. These results mean that the emergent poisoning attack can prevent GCL from being widely utilized from now on. Therefore, developing defense and robust GCL models that resist poisoning attacks is of significant importance.

[1]Unsupervised attacks mean that attackers succeed in creating poisoned graphs without labels of nodes.

C. *Defenses against Poisoning Attack on GCL*

Recently, Feng et al. have proposed a GCL method called ARIEL [28] that is robust to poisoning attacks. In ARIEL, an adversarial view created by an existing attack [47] is introduced so as to make the GCL model robust to poisoning attacks. In other words, the technique that is equivalent to adversarial training is utilized in ARIEL. The adversarial view is treated as another view to assimilate adversarial training to the GCL. The adversarial contrastive loss is newly defined as the contrastive loss between one of the two views and the adversarial view. Although adversarial training in ARIEL may effectively improve the robustness of the model to poisoned graphs, the progress in such hard training may be stagnant in bad parameter area at an early stage. To realize stable training, ARIEL introduces one additional constraint called information regularization. Furthermore, ARIEL adopts two additional techniques, namely subgraph sampling and curriculum learning. The subgraph sampling can reduce the computational cost because the gradient derivation on the entire graph is avoided. On the other hand, the curriculum learning contributes to making GCL become harder gradually by increasing the portion of the adversarial contrastive loss as the training progresses. Accordingly, ARIEL outperforms the existing GCL methods in the node classification task on poisoned graphs, which is further improvement in the robustness of the GCL. Strictly speaking, ARIEL is not against a GCL poisoning attack. However, compared with GCA, ARIEL can counter the GCL poisoning attack, namely CLGA to some extent according to our experiments. Furthermore, ARIEL can mitigate bad effects of poisoned graphs without utilizing labels. This corresponds to our assumed situation that is defending GCL from poisoning attacks without depending on labels, which is described in Section IV. Thus, we regard ARIEL as a state-of-the-art defense in the GCL domain.

ARIEL is a promising defense against CLGA. However, ARIEL only tries to learn a robust model by introducing the adversarial view when a poisoned graph is given. Thus, as shown in Section VI, ARIEL cannot sufficiently eliminate adverse effects from poisoned graphs, which means that there is still room for improvement to the quality of the embeddings.

On the other hand, EdgePruner can further eliminate adverse effects from poisoned graphs. Our method focuses on eliminating adverse effects by pruning adversarial edges rather than learning robust parameters of the encoder by the GCL, which is the different point from ARIEL.

III. Preliminaries

A. *GCL for Node Embeddings*

In this work, we focus on node-level GCL. Let $G = (\boldsymbol{A}, \boldsymbol{X})$ denote a graph, where $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ is the node feature matrix. N is the number of nodes, and d is the number of features. The objective of GCL is to learn an encoder $f(\boldsymbol{A}, \boldsymbol{X})$ that outputs node embeddings.

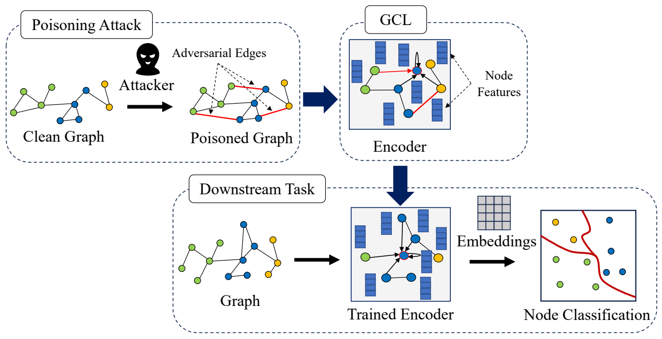


Fig. 1. Threat model.

In many cases [28], [29], [42], GCN [35] is utilized as the encoder f in GCL due to its simplicity and the stable performance. In this work, we also employ a two-layer GCN as with the other work. The overview of GCN is explained in Appendix A1. As for GCL method, we utilize GCA [42], which is the state-of-the-art GCL in our experiments. The details of GCA are described in Appendix A2.

B. Unsupervised poisoning attack on GCL

We focus on the state-of-the-art attack in the GCL domain, namely CLGA in this work. CLGA is the untargeted poisoning attack that deteriorates the overall classification performance for samples in every class. The purpose of CLGA is to poison graphs so that the quality of the embeddings learned by GCL is degraded. As a result, the poisoned embeddings cause worse performance in downstream tasks. In what follows, we explain the overview of CLGA.

Overview of CLGA. CLGA only modifies edges and does not change node features. In other words, CLGA creates only the poisoned adjacency matrix $\boldsymbol{A}^{'}$. CLGA creates $\boldsymbol{A}^{'}$ in the iterative manner so as to locate more informative edges. Once an edge is modified at each iteration, the modified adjacency matrix is used to retrain the encoder to obtain gradients in the next iteration. In an attempt to create $\boldsymbol{A}^{'}$, CLGA tries to solve the following optimization problem

$$\begin{gathered}\max_{\boldsymbol{A}^{'}} \mathcal{L}(f_{\theta'}(\boldsymbol{A}_1,\boldsymbol{X}_1), f_{\theta'}(\boldsymbol{A}_2,\boldsymbol{X}_2)),\\ \text{s.t. } \theta' = \operatorname*{argmin}_{\theta} \mathcal{L}(f_{\theta}(\boldsymbol{A}_1,\boldsymbol{X}_1), f_{\theta}(\boldsymbol{A}_2,\boldsymbol{X}_2)),\\ (\boldsymbol{A}_1,\boldsymbol{X}_1) = t_1(\boldsymbol{A}^{'},\boldsymbol{X}), (\boldsymbol{A}_2,\boldsymbol{X}_2) = t_2(\boldsymbol{A}^{'},\boldsymbol{X}), \|\boldsymbol{A}-\boldsymbol{A}^{'}\|_F^2 = \sigma.\end{gathered} \tag{1}$$

The Frobenius norm of $\boldsymbol{A}$ is defined by $\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} \boldsymbol{A}[i,j]}$. In addition to that, t_1 and t_2 are two stochastic augmentation procedures. When it is assumed that GCA is attacked, $\mathcal{L}$ is equivalent to the contrastive loss in Eq. (13) in Appendix A2. θ is parameters of f. The number of pruned edges is bounded by a given threshold σ.

C. Threat Model

Fig. 1 shows the threat model in this work. The encoder is trained by GCL on unlabeled graphs in order to facilitate utilizing them in downstream tasks such as node classification. The encoder may be trained on graphs published by a third party. Downstream tasks utilize the embeddings obtained through inputting nodes in a graph into the trained encoder as features.

Attacker's goal. Attacker's goal is to extensively launch attack on many downstream tasks. Considering the efficiency of attacks, it is desirable for the attacker to directly contaminate the encoder trained GCL rather than downstream tasks. This is why the attacker conducts the poisoning attack on GCL.

Attacker's capability. The attacker has no access to both the classification models and graph datasets utilized in the downstream tasks. Meanwhile, the attacker knows the model parameters of target GCL for some reasons such as information leakage, which means the worst case for defenders. Thus, the attacker can reproduce the target GCL model. To create the poisoned graph, the attacker modifies only edges in a graph so that the loss of the reproduced model is maximized by CLGA. The poisoned graphs are published by the attacker so as to contaminate encoders. If the attacker successfully poisons the target encoder, the resulting embeddings are degraded. Accordingly, the attacker can indirectly have detrimental effects on myriads of downstream tasks, which means the attack success.

IV. Problem Statement and Motivation

As mentioned in Section II-B, we consider that the poisoning attack on GCL becomes a severe obstacle. However, the studies regarding defensive methods for GCL are not adequately explored. To defend GCL, it is preferable that defenders counter the attack without depending on labels as much as possible. Thus, in this work, we assume the situation where defenders cannot utilize abundant labels of graphs. Although ARIEL can mitigate the bad effect of poisoned graphs, there is still room for improvement. To achieve further improvement, we argue that sanitizing the poisoned graph is needed. Sanitizing means pruning adversarial edges from graphs in this work. Furthermore, we assume that edge pruning is conducted just before training an encoder rather than training classification models in downstream tasks. This is because defending training encoders is more efficient in light of myriads of downstream tasks. To the best of our knowledge, there has been no study that focuses on adversarial edge pruning in the GCL domain. Therefore, since the feasibility still remains largely unexplored, we decided to work on that.

V. Proposed Method

A. Overview

In this paper, we propose EdgePruner, which is poisoned edge pruning in GCL. We focus on the fact that the poisoning attack on GCL tends to mainly add adversarial edges to create poisoned graphs. On the basis of this fact, we consider that the deletion of edges is important to sanitize poisoned graphs. One promising way for determining which edges should be deleted is to utilize gradients of the contrastive loss based on embeddings from the encoder that trains on a poisoned

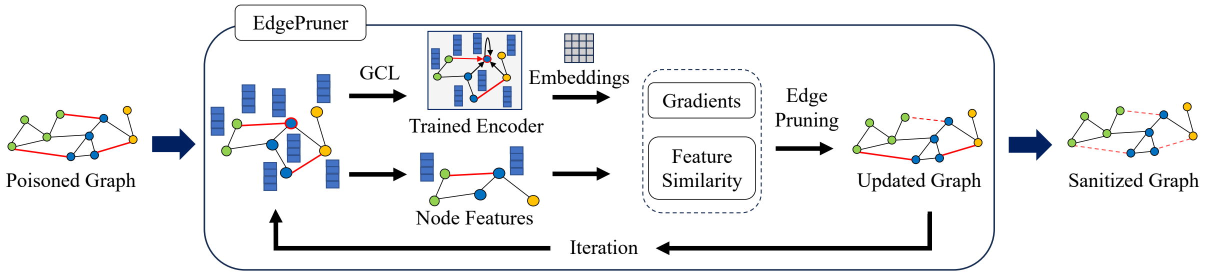


Fig. 2. Overview of EdgePruner.

graph. By pruning edges that contribute to minimizing the contrastive loss, sanitizing poisoned graphs can be expected. However, there are cases where it is difficult to decide edges to prune because the trained encoder is influenced by adversarial edges in the poisoned graph. As a result, the contrastive loss based on the embeddings from the encoder may not be totally reliable. Furthermore, since the node labels are not available in GCL, counting only on the encoder trained on poisoned graphs is not necessarily desirable. Thus, additional information that is not influenced by adversarial edges is needed. In order to help determine the appropriate edges, we focus on the fact that adversarial edges tend to connect nodes for which the features are dissimilar. This is because connecting such nodes is effective in confusing the relationship between neighboring nodes in a graph. As a result, embeddings are poisoned, which results in degrading the performance of the downstream tasks. In fact, recently, it has been demonstrated that adversarial attacks on graphs tend to connect nodes with distinct features [30], [31]. Thus, in light of this tendency, we utilize feature similarity between neighboring nodes connected to each other in order to prune adversarial edges connecting nodes with distinct features. This similarity is helpful in further eliminating adverse effects from poisoned graphs. Finally, EdgePruner outputs a graph that yields the minimum contrastive loss as the sanitized graph. Fig. 2 shows the overview of EdgePruner.

In what follows, we first validate how edges are changed when poisoned graphs are created and formulate an optimization problem that EdgePruner solves. After that, we elaborate on the usefulness of feature similarity between neighboring nodes. Finally, the algorithm of EdgePruner is explained.

B. Edge Modification on Poisoned Graphs

To validate the tendency that adversarial edges are principally added for creating poisoned graphs, we inspect the number of edges that are modified when poisoned graphs are created. TABLE I shows the number of modified edges when poisoned graphs are created by CLGA. In this inspection, we utilize six datasets, namely Cora, CiteSeer [48], Amazon-Computers, Amazon-Photo [3], Coauthor-CS, and Coauthor-Physics [3]. Note that we utilize poisoned graphs created from subgraphs of 5,000 nodes on Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics as with the experimental setup in [28] because we could not create the poisoned graphs based on their entire graphs due to memory limitations. Detailed information about these datasets is described in Section VI. The poisoned graphs in TABLE I are the ones created by maximizing the loss of GCA. As for ARIEL, we observe a similar result, which is shown in Appendix B. As shown in TABLE I, it is obvious that poisoned graphs are created by adding edges to clean graphs in most cases. In particular, as for Cora and CiteSeer, poisoned graphs are created only by adding edges. Therefore, we consider that the deletion of edges is more important than the addition so as to sanitize poisoned graphs by unsupervised attacks on GCL.

EdgePruner deletes edges on the basis of gradient descent on the adjacency matrix. In other words, EdgePruner converts $\boldsymbol{A}^{'}$ to the sanitized adjacency matrix $\boldsymbol{S}$ through edge pruning so that a contrastive loss $\mathcal{L}_s$ of an encoder f that we defend is minimized. Thus, EdgePruner solves the following optimization problem formulated as

$$\begin{aligned} &\min_{\boldsymbol{S}} \mathcal{L}_s(\theta', \boldsymbol{S}, \boldsymbol{X}),\\ &\text{s.t. } \theta' = \operatorname*{argmin}_{\theta} \mathcal{L}_s(\theta, \boldsymbol{S}, \boldsymbol{X}),\ \|\boldsymbol{A}^{'} - \boldsymbol{S}\|_F^2 \le \sigma^{'}. \end{aligned} \tag{2}$$

When GCA and ARIEL are assumed, $\mathcal{L}_s$ is defined as

$$\begin{aligned} \mathcal{L}_s(\theta', \boldsymbol{S}, \boldsymbol{X}) = &\mathcal{L}_{GCL}(\boldsymbol{H}^1_{\theta'}, \boldsymbol{H}^2_{\theta'})\\ &+ \epsilon_1 \cdot \mathcal{L}_{GCL}(\boldsymbol{H}^1_{\theta'}, \boldsymbol{H}^3_{\theta'}) + \epsilon_2 \cdot \mathcal{L}_{\text{IR}}. \end{aligned} \tag{3}$$

Note that $\boldsymbol{H}^m_{\theta'}$ is defined in Appendix A2. $\mathcal{L}_{\text{IR}}$ is information regularization introduced in ARIEL. Also, ϵ_1 and ϵ_2 are predefined parameters to control adversarial contrastive loss and information regularization used in ARIEL. When $\epsilon_1 = 0$ and $\epsilon_2 = 0$, $\mathcal{L}_s$ is equivalent to the contrastive loss of GCA. The number of modified edges is bounded by a threshold $\sigma^{'}$.

In order to select informative edges, we execute K times calculations of $\mathcal{L}_s$. On the basis of $\boldsymbol{S}$ and $\boldsymbol{X}$, multiple views $G^k_m = (\boldsymbol{S}^k_m, \boldsymbol{X}^k_m)$ are obtained via the stochastic augmentations, which are repeated K times. The value of m is an index for identifying M views generated in GCL methods. For example, since GCA generates two views, $M = 2$. On the other hand, since ARIEL requires the adversarial view in addition to the two views, $M = 3$. At the k-th augmentation, the contrastive loss $\mathcal{L}^k_s$ based on G^k_m is computed. With respect to $\boldsymbol{S}^k_m$, gradient matrices of $\mathcal{L}^k_s$, namely $\nabla^k_m \in \mathbb{R}^{N\times N}$ are

TABLE I
THE NUMBER OF EDGES THAT ARE ADDED AND DELETED WHEN POISONED GRAPHS ARE CREATED BY CLGA.

	Cora			CiteSeer			Amazon-Computers			Amazon-Photo			Coauthor-CS			Coauthor-Physics		
	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
Added edges	52	263	528	45	227	445	355	1698	3235	496	2477	4927	61	309	618	52	260	523
Deleted edges	0	0	0	0	0	0	1	85	333	0	3	33	0	0	1	0	2	2

calculated. When $f(\boldsymbol{S},\boldsymbol{X})$ is a differentiable encoder such as GCN, we can calculate ∇_m^k as

$$\nabla_m^k = \frac{\partial \mathcal{L}_s^k}{\partial \boldsymbol{S}_m^k} = \frac{\partial \mathcal{L}_s^k}{\partial f(\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)} \cdot \frac{\partial f(\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)}{\partial \boldsymbol{S}_m^k}. \quad (4)$$

To alleviate the bias caused by the stochastic augmentation, we adopt two techniques that are introduced in CLGA [29]. The first one is adding up ∇_m^k. The gradient matrix at the k-th augmentation is regarded as

$$\nabla^k = \sum_m^M \nabla_m^k, \quad (5)$$

where M is the number of augmented views used in an assumed GCL. The other is adding up ∇^k over K times iteration, which means $\nabla_{\text{total}} = \sum_{k=1}^K \nabla^k$. This total gradient matrix ∇_{total} is utilized to select pruned edges.

EdgePruner prunes edges that meet a condition regarding the presence of edges and the correct direction of gradients. Pruning an edge means changing a value from 1 to 0 in the adjacency matrix. Thus, the condition is represented as

$$\boldsymbol{S}[i,j] = 1 \land \nabla_{\text{total}}[i,j] > 0. \quad (6)$$

If an existent edge (which means the corresponding value is 1 in the adjacency matrix) has a positive gradient, changing the value from 1 to 0 would decrease the loss, which means deleting an edge. Our edge pruning is mainly conducted by deleting an edge out of edges that meet Eq. (6) one by one. After an edge is deleted, $\boldsymbol{S}$ is checked to see whether the loss $\mathcal{L}_s$ is minimum at that time. Let $\mathcal{L}_{\min}$ denote the minimum loss during our pruning procedures. If $\mathcal{L}_s$ is less than $\mathcal{L}_{\min}$, $\boldsymbol{S}$ is saved as the optimal adjacency matrix $\boldsymbol{S}_{\text{opt}}$, which should be returned as output. The updated graph $G_{\text{u}} = (\boldsymbol{S},\boldsymbol{X})$ is fed into f to retrain. Once the encoder is retrained, ∇_{total} based on node embeddings from the trained encoder is recalculated. Then, another edge is pruned in the next iteration. By iterating the above procedures, the poisoned graph is gradually sanitized. Finally, the sanitized graph $G_s = (\boldsymbol{S}_{\text{opt}},\boldsymbol{X})$ is outputted.

C. Features Similarity of Neighboring nodes

Pruning adversarial edges is not always an easy task if we rely only on gradients calculated on poisoned graphs. This is because training the encoder is influenced by adversarial edges in poisoned graphs at each iteration, which interrupts selecting edges to prune appropriately. To help determine the appropriate edges, we focus on the feature similarity between neighboring nodes connected to each other. As mentioned in Section V-A, nodes that have distinct features are connected when the poisoned graph is created. We inspect the feature similarity of neighboring nodes connected by clean edges or adversarial ones. TABLE II shows the average cosine similarity of neighboring nodes in poisoned graphs based on the loss of GCA. It is observed that the poisoned graphs based on the loss of ARIEL also have similar tendency in our preliminary experiment. In TABLE II, we also utilize the poisoned graphs created from subgraphs of Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics. As shown in TABLE II, the average cosine similarities of nodes connected by adversarial edges are small compared with those of nodes connected by clean edges. Furthermore, distribution of the cosine similarity on the six datasets are shown in Appendix J. These inspection results demonstrate that poisoned graphs tend to be created by connecting nodes whose features are distinct. Hence, we introduce the cosine similarity of features of neighboring nodes as additional information for appropriately determining edges to prune. If node features are available, EdgePruner deletes an edge that meets Eq. (6) and connects two nodes whose features are distinct to some extent at each iteration. In other words, the condition regarding feature similarity is represented as

$$\mathrm{s}(\boldsymbol{X}[i,:],\boldsymbol{X}[j,:]) < T, \quad (7)$$

where i and j are indices of nodes. Also, $\mathrm{s}(\cdot,\cdot)$ and T are the cosine similarity and a threshold, respectively. In practical situations, T should be tuned depending on parameters of GCL methods and datasets. Thus, when node features are available, instead of Eq. (2), EdgePruner solves the following optimization problem

$$\begin{gathered}\min_{\boldsymbol{S}} \mathcal{L}_s(\theta', \boldsymbol{S}, \boldsymbol{X}), \text{ s.t. } \theta' = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_s(\theta, \boldsymbol{S}, \boldsymbol{X}), \\ \|\boldsymbol{A}^{'} - \boldsymbol{S}\|_F^2 \le \sigma^{'}, \bar{\mathrm{s}}(\boldsymbol{S}) > \bar{\mathrm{s}}(\boldsymbol{A}^{'}),\end{gathered} \quad (8)$$

where $\bar{\mathrm{s}}$ is defined as

$$\bar{\mathrm{s}}(\boldsymbol{A}) = \frac{\sum_{i,j} \mathrm{s}(\boldsymbol{X}[i,:],\boldsymbol{X}[j,:]) \cdot \boldsymbol{A}[i,j]}{\|\boldsymbol{A}\|_F^2}. \quad (9)$$

D. Algorithm

In this subsection, we concisely describe the algorithm of EdgePruner. Algorithm 1 shows the algorithm of EdgePruner. The notation table for Algorithm 1 is shown as TABLE IX in Appendix C. EdgePruner takes a poisoned graph $G' = (\boldsymbol{A}^{'}, \boldsymbol{X})$ as input. First of all, $\boldsymbol{S}$ is initialized as the poisoned one $\boldsymbol{A}^{'}$ (Line 2). Similarly, the optimal adjacency matrix $\boldsymbol{S}_{\text{opt}}$is initialized as $\boldsymbol{S}$ (Line 3). Note that $\boldsymbol{S}_{\text{opt}}$ is the output of Algorithm 1. Furthermore, the minimum loss $\mathcal{L}_{\min}$ is initialized (Line 4).

TABLE II
THE AVERAGE COSINE SIMILARITY OF NODES CONNECTED BY CLEAN EDGES OR ADVERSARIAL ONES IN THE POISONED GRAPHS WITH STANDARD DEVIATION.

	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
Clean edges	0.167 ± 0.127	0.191 ± 0.140	0.495 ± 0.209	0.489 ± 0.202	0.297 ± 0.133	0.352 ± 0.154
Adversarial edges	0.029 ± 0.048	0.021 ± 0.029	0.297 ± 0.087	0.297 ± 0.098	0.011 ± 0.018	0.025 ± 0.032

Algorithm 1 EdgePruner

```
Input: Poisoned Graph $G' = (\boldsymbol{A}^{'}, \boldsymbol{X})$
 1: $n \leftarrow 0$
 2: Initialize the sanitized adjacency matrix $\boldsymbol{S} \leftarrow \boldsymbol{A}^{'}$
 3: Initialize the optimal matrix $\boldsymbol{S}_{\mathrm{opt}} \leftarrow \boldsymbol{S}$
 4: Initialize the minimum loss $\mathcal{L}_{\min} \leftarrow \infty$
 5: while $n < N$ do
 6:     Train $f$ with $\boldsymbol{S}$ and $\boldsymbol{X}$ from scratch
 7:     Initialize gradients $\nabla_{\mathrm{total}} \leftarrow 0$
 8:     for $k = 1$ to $K$ do
 9:         Obtain $m$ views $G_m^k = (\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)$
10:         Compute $\mathcal{L}_s^k$ based on the trained $f$ via Eq. (3)
11:         $\nabla_m^k \leftarrow \frac{\partial \mathcal{L}_s^k}{\partial \boldsymbol{S}_m^k}$    ▷ Compute the gradient.
12:         $\nabla^k \leftarrow \sum_m^M \nabla_m^k$
13:         $\nabla_{\mathrm{total}} \leftarrow \nabla_{\mathrm{total}} + \nabla^k$
14:     end for
15:     Create a set of candidates $\boldsymbol{C}$ based on Eq. (6) and Eq. (7)
16:     if $\boldsymbol{C} \neq \varnothing$ then
17:         Delete $\boldsymbol{C}[i, j]$ that has the largest gradient
18:         if $\mathcal{L}_s < \mathcal{L}_{\min}$ then
19:             $\mathcal{L}_{\min} \leftarrow \mathcal{L}_s$
20:             $\boldsymbol{S}_{\mathrm{opt}} \leftarrow \boldsymbol{S}$    ▷ Save the optimal adjacency matrix.
21:         end if
22:     else
23:         break    ▷ There is no edge to prune.
24:     end if
25:     $n \leftarrow n + 1$
26: end while
27: return Sanitized Graph $G_s = (\boldsymbol{S}_{\mathrm{opt}}, \boldsymbol{X})$
```

The encoder f is trained with $\boldsymbol{S}$ and $\boldsymbol{X}$ (Line 6). After that, depending on an assumed GCL method, $G_m^k = (\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)$ are obtained via the k-th stochastic augmentation, which is repeated K times (Line 8). By inputting $G_m^k = (\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)$ into the trained $f_{\theta'}$, ∇_{total} is calculated (Line 13). The candidates of pruned edges are selected from $\boldsymbol{S}$ depending on whether they meet the two conditions in Eq. (6) and Eq. (7) (Line 15). Let $\boldsymbol{C}$ the candidate edges to prune. Then, out of edges in $\boldsymbol{C}$, EdgePruner prunes one edge that has the largest gradient (Line 17). Note that EdgePruner executes only deleting edges although it is desirable that sanitizing poisoned graphs be conducted by edge modification including both deletion and addition of edges. This is because EdgePruner simply tries to solve optimization problems in Eq. (2) or Eq. (8) in a greedy manner at this stage. If there is no edge in $\boldsymbol{C}$, our edge pruning is stopped at that point. Once an edge is deleted, $\mathcal{L}_s$ is less than $\mathcal{L}_{\min}$, $\boldsymbol{S}$ is saved as $\boldsymbol{S}_{\mathrm{opt}}$ (Line 20). After that, EdgePruner tries to prune another edge in the next iteration by repeating the above mentioned procedures (Lines 5-26). Finally, after up to N edges are pruned, the sanitized graph $G_s = (\boldsymbol{S}_{\mathrm{opt}}, \boldsymbol{X})$ is returned (Line 27).

VI. EXPERIMENTS

In this section, we evaluate the effectiveness of EdgePruner. In particular, we mainly evaluate the performance of the node classification on both the graphs sanitized by the EdgePruner and the poisoned ones to reveal the following questions:

1) Can EdgePruner eliminate the adverse effects from the poisoned graphs by pruning edges?
2) Is the feature similarity effective in selecting adversarial edges to prune on various datasets?
3) How does EdgePruner affect the quality of the embeddings on clean graphs?
4) How does the number of pruned edges affect the quality of the embeddings on poisoned and clean graphs?
5) Is EdgePruner effective against an adaptive attack?

Evaluation metric is the accuracy that is defined as the percentage of correctly predicted nodes to all the testing nodes. The higher the accuracy is, the better the quality of embeddings is. All our experiments are conducted on the NVIDIA GeForce RTX 3090 GPU with a 24GB memory.

A. Experimental Setups

Dataset. We utilize the six datasets including Cora, CiteSeer [48], Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics [3] following [28], [29]. The details about these datasets are shown in Appendix.D

GCL. As for GCL methods, GCA [42] and ARIEL [28] are utilized for training the encoder that outputs embeddings. The details about GCL methods are in Appendix E.

Poisoning attack. We utilize CLGA [29] as the attack method in our experiments. The details about the setup of CLGA are in Appendix F.

Downstream task. The downstream task in our experiments is node classification. We follow the evaluation scheme conducted in the other studies [28], [29], [42], where a single graph of each dataset is firstly trained by the GCL method, then the resulting embeddings are utilized to train and test a classifier for the node classification. The details about the setup of the downstream task are in Appendix G.

Baseline. We compare EdgePruner with baselines to clarify the effectiveness of EdgePruner. We compare EdgePruner to two baselines for supervised GNN, namely GNNGuard [49] and GNNJaccard [30]. Although these baselines are not for GCL methods, this comparison clarifies to what extent our method can eliminate adverse effects from poisoned graphs compared to them. Detailed hyperparameters are described in Appendix H. Furthermore, we consider a method that prunes multiple edges once at all as the baseline in this experiment. As with EdgePruner, the baseline selects edges to prune on the basis of the gradients of the contrastive losses. The difference from EdgePruner is that the baseline always prunes a designated number of edges once at all. In this experiment, we set the number of pruned edges at 10% of all the edges in a graph. There are the four baselines depending on the combinations of baseline and GCL methods, namely Baseline$^{\mathrm{NF}}$(G), Baseline$^{\mathrm{NF}}$(A), Baseline(A), and Baseline(G). Note that G and A are the abbreviation of GCA and ARIEL, respectively. As with EdgePruner, Baseline$^{\mathrm{NF}}$(G), Baseline$^{\mathrm{NF}}$(A) do not utilize the feature similarity of neighboring nodes. On the other hand, Baseline(A) and Baseline(G) select the edge to prune with the feature similarity.

EdgePruner. In EdgePruner, the encoder is retrained on updated graphs by a supposed GCL method during the pruning procedure for one epoch. This is because it is possible that training poisoned graphs for a large number of epochs makes the encoder poisoned, which causes inappropriate selection of pruned edges. The number of stochastic augmentations K shown in Line 8 in Algorithm 1 is set at 10 for calculating ∇_{total} in all experiments. Since EdgePruner is designed for eliminating adverse effects from poisoned graphs, EdgePruner can be with any GCL methods including GCA and ARIEL. We define the proposed methods as the GCL methods that take sanitized graphs instead of poisoned ones. There are the four proposed methods, specifically EdgePruner$^{\mathrm{NF}}$ (G), EdgePruner$^{\mathrm{NF}}$ (A), EdgePruner (G), and EdgePruner (A). All the four proposals are allowed to prune up to 10% of edges in a graph so that the contrastive losses of supposed GCL methods are minimized. The details about EdgePruner are shown in Appendix I.

B. Effectiveness of Edge Pruning on Poisoned Graphs

We evaluate the effectiveness of the proposed variants under CLGA by comparing the node classification accuracy on sanitized graphs with that on poisoned ones. TABLE III shows the node classification accuracy under CLGA. As shown in TABLE III, the variants of EdgePruner achieve both the best and the second best accuracies on Cora, Amazon-Computers, Coauthor-CS, and Coauthor-Physics. In the following discussions, we compare EdgePruner with the other methods while referring to the results in TABLE III.

Comparison with baselines. Compared with the our Baseline methods, the variants of EdgePruner attains the best accuracies on the five datasets except for Amazon-Photo. EdgePruner (A) and EdgePruner$^{\mathrm{NF}}$ (A) achieve the best accuracies, namely 71.83% and 84.92% on CiteSeer and Amazon-Computers, respectively. EdgePruner (G) also yields the best accuracies on Cora, Coauthor-CS, and Coauthor-Physics. On the other hand, Baseline (A) attains the best accuracy, 88.60% only on Amazon-Photo. As for other cases, the baselines are outperformed by EdgePruner. In particular, the accuracy of Baseline (G) on Coauthor-CS is 82.32%, which is 8.58% lower than that of EdgePruner (G).

Additionally, we consider that EdgePruner is more competent than the Baseline variants from the point of view of robustness against an adaptive attack. Since Baseline variants always prune a designated number of edges, they are vulnerable to an adaptive attack. In other words, it is possible for the adaptive attack to perturb edges so that the number of perturbed edges is larger than the number of pruned edges if the number of edges pruned by the Baseline method is leaked to the attacker. On the other hand, EdgePruner is immune to the adaptive attack. We will show the details of the results on the adaptive attack in Section VI-E. As for existing baselines for GNN, GNNGuard only achieves the second best accuracy on Coauthor-Physics. Overall, GNNGuard and GNNJaccard are inferior to EdgePruner although they counter poisoned graphs by using labels. For these reasons, EdgePruner is superior to the baselines on poisoned graphs.

Comparison with existing GCL methods. Compared with GCA, EdgePruner (G) outperforms GCA on all the datasets. In particular, EdgePruner (G) achieves 90.90% accuracy on Coauthor-CS, which is 9.60% higher than that of GCA. Additionally, EdgePruner (G) achieves 93.04% accuracy on Coauthor-Physics, which is 4.38% higher than that of GCA. On the other hand, EdgePruner (A) and EdgePruner$^{\mathrm{NF}}$ (A) also improve the accuracies in most cases compared with ARIEL. In particular, EdgePruner (A) yields 71.83% accuracy on CiteSeer, which is the best accuracy. In addition to that, EdgePruner (A) achieves the second best accuracies, namely 83.30% and 89.13% accuracies on Amazon-Computers and Coauthor-CS, respectively. Similarly, EdgePruner$^{\mathrm{NF}}$ also yields the best accuracy on Amazon-Computers. The overall results demonstrate that EdgePruner is effective in improving the quality of the embeddings. Additionally, we consider that EdgePruner may also be effective in sanitizing poisoned graphs created by other attacks on GCL as long as poisoned graphs are created by mainly adding adversarial edges. Thus, we conclude that EdgePruner can eliminate adverse effects from the poisoned graphs, which answers the research question (1).

Impact of similarity of node features. We compare EdgePruner with EdgePruner$^{\mathrm{NF}}$ to evaluate the impact of feature similarity of neighboring nodes. As shown in TABLE III, EdgePruner (G) and EdgePruner (A) do not always outperform EdgePruner$^{\mathrm{NF}}$ (G) and EdgePruner$^{\mathrm{NF}}$ (A). In particular, EdgePruner$^{\mathrm{NF}}$ (G) achieves higher accuracies on Amazon-Computers and Amazon-Photo datasets compared with EdgePruner (G). Similarly, EdgePruner$^{\mathrm{NF}}$ (A) also outperforms EdgePruner (A) on the two datasets. Furthermore, EdgePruner$^{\mathrm{NF}}$ (A) yields 84.92% accuracy on Amazon-

TABLE III
ACCURACY $\pm$ STANDARD DEVIATION (%) OF NODE CLASSIFICATION UNDER CLGA. THE BOLD AND UNDERLINED ACCURACIES ARE THE BEST AND THE SECOND BEST ONES, RESPECTIVELY.

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	80.91 $\pm$ 1.29	**71.83 $\pm$ 0.69**	83.30 $\pm$ 0.74	88.37 $\pm$ 0.32	89.13 $\pm$ 0.69	92.34 $\pm$ 0.51
EdgePruner (G)	**81.70 $\pm$ 0.99**	69.82 $\pm$ 1.83	81.17 $\pm$ 0.60	83.63 $\pm$ 0.42	**90.90 $\pm$ 0.67**	**93.04 $\pm$ 0.38**
EdgePruner$^{\mathrm{NF}}$ (A)	81.44 $\pm$ 1.10	69.80 $\pm$ 0.95	**84.92 $\pm$ 0.63**	88.58 $\pm$ 0.33	88.14 $\pm$ 0.61	91.76 $\pm$ 0.56
EdgePruner$^{\mathrm{NF}}$ (G)	79.83 $\pm$ 0.96	63.26 $\pm$ 1.27	82.63 $\pm$ 0.69	83.88 $\pm$ 0.55	85.35 $\pm$ 0.81	90.51 $\pm$ 0.31
Baseline (A)	81.08 $\pm$ 0.67	71.18 $\pm$ 0.75	83.19 $\pm$ 0.41	**88.60 $\pm$ 0.39**	86.73 $\pm$ 0.59	90.20 $\pm$ 0.44
Baseline (G)	80.89 $\pm$ 1.01	63.17 $\pm$ 1.48	80.53 $\pm$ 0.58	82.53 $\pm$ 0.67	82.32 $\pm$ 0.50	89.48 $\pm$ 0.41
Baseline$^{\mathrm{NF}}$ (A)	80.23 $\pm$ 0.99	70.24 $\pm$ 0.93	82.49 $\pm$ 1.01	88.36 $\pm$ 0.38	86.24 $\pm$ 0.52	89.50 $\pm$ 0.50
Baseline$^{\mathrm{NF}}$ (G)	79.16 $\pm$ 1.09	61.56 $\pm$ 1.35	80.64 $\pm$ 0.69	85.71 $\pm$ 0.54	81.40 $\pm$ 0.44	88.90 $\pm$ 0.38
GNNGuard	79.53 $\pm$ 1.49	69.93 $\pm$ 1.42	81.18 $\pm$ 0.75	87.57 $\pm$ 0.66	88.16 $\pm$ 0.69	92.81 $\pm$ 0.32
GNNJaccard	79.95 $\pm$ 0.73	67.46 $\pm$ 1.85	81.13 $\pm$ 0.97	87.99 $\pm$ 0.65	83.67 $\pm$ 0.86	89.93 $\pm$ 0.57
ARIEL	80.77 $\pm$ 1.18	69.96 $\pm$ 0.94	82.51 $\pm$ 0.70	87.26 $\pm$ 0.37	85.35 $\pm$ 0.79	87.70 $\pm$ 0.38
GCA	80.37 $\pm$ 0.79	61.73 $\pm$ 1.47	79.42 $\pm$ 0.77	81.10 $\pm$ 0.47	81.30 $\pm$ 0.68	88.66 $\pm$ 0.35

Computers and 88.58% accuracy on Amazon-Photo, which are the best and the second best ones, respectively. These results mean that utilizing the feature similarity is not always required for sanitizing poisoned graphs. The reason for this could be that the distributions of the cosine similarity of nodes connected by adversarial edges on Amazon-Photo and Amazon-Computers are different from those on the other datasets. As shown in Fig. 5 in Appendix J, the cosine similarity scores on the Amazon datasets are widely distributed. This is why it may be difficult to identify adversarial edges on the two datasets compared with adversarial edges on other datasets. However, compared with ARIEL, EdgePruner$^{\mathrm{NF}}$ (A) cannot improve the accuracies on CiteSeer. EdgePruner$^{\mathrm{NF}}$ (G) also degrades the accuracy on Cora compared with GCA. On the other hand, EdgePruner (G) and EdgePruner (A) can improve the accuracies of node classification compared with those of GCA and ARIEL, respectively, on all the datasets. According to these results, we conclude that feature similarity is basically effective in selecting pruned edges on various datasets, which is the answer to the research question (2).

C. Effect of Edge Pruning on Clean Graphs

In this subsection, we evaluate the effect of our edge pruning on clean graphs. It is difficult for defenders to judge whether an input graph is clean or poisoned every time. Thus, it is desirable that edge pruning is applicable to input graphs regardless of whether they are poisoned. TABLE IV shows the accuracy of node classification on clean graphs. In TABLE IV, we report the results of node classification evaluated on clean subgraphs of 5,000 nodes on Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics. As shown in TABLE IV, ARIEL basically outperforms GCA on all datasets except for Coauthor-Physics.

Comparison with baselines. As we can see from TABLE IV, the best and the second best accuracies appear in different methods depending on the datasets. Baseline (A) and Baseline$^{\mathrm{NF}}$ (A) achieve the best accuracies on Amazon-Computers and Amazon-Photo, respectively. On the other hand, EdgePruner (A) and EdgePruner (G) yield the best accuracies on Coauthor-CS and Coauthor-Physics, respectively. Considering the best accuracies, the proposed methods are comparable to the our Baseline variants. However, EdgePruner (G) EdgePruner$^{\mathrm{NF}}$ (A) and EdgePruner$^{\mathrm{NF}}$ (G) yield the second best accuracies on Coauthor-CS, CiteSeer, and Coauthor-Physics, respectively. On the other hand, there is no the second best accuracy in Baseline variants. As for existing baselines for GNN, GNNJaccard only achieves the second best accuracy on Amazon-Photo. Although GNNGuard and GNNJaccard achieve relatively better accuracies on Amazon datasets compared with EdgePruner variants. However, they are inferior to EdgePruner on the other datasets. From these results, the variants of EdgePruner slightly outperforms the baselines on clean graphs.

Comparison with existing GCL methods. In the case of Coauthor-CS and Coauthor-Physics, it is observed that EdgePruner (A) and EdgePruner (G) can yield better accuracies than ARIEL and GCA, respectively. These results mean that our pruning is also capable of eliminating adverse edges in clean graphs. To be precise, EdgePruner (A), and EdgePruner (G) achieve the best accuracies, namely 90.13%, and 93.52% accuracies on Coauthor-CS and Coauthor-Physics, respectively. The detailed interpretation is shown in Appendix K. In addition to that, EdgePruner (G), EdgePruner$^{\mathrm{NF}}$ (A), and EdgePruner$^{\mathrm{NF}}$ (G) also yield the second best accuracies on Coauthor-CS, CiteSeer, and Coautor-Physics, respectively.

On the other hand, both EdgePruner (G) and EdgePruner$^{\mathrm{NF}}$ (G) are inferior to GCA on Cora, CiteSeer, Amazon-Computers, and Amazon-Photo. Similarly, both EdgePruner (A) and EdgePruner$^{\mathrm{NF}}$ (A) are inferior to ARIEL on these four datasets. In the worst case, compared with GCA, EdgePruner$^{\mathrm{NF}}$ (G) degrades the accuracies on clean graphs by 2.54%, 0.76%, 3.47%, and 0.97% on Cora, CiteSeer, Amazon-Computers, and Amazon-Photo, respectively. In other words, the accuracy drops are within approximately 3.5%. According to these results, it seems that our pruning may degrade the quality of embeddings on clean graphs depending on datasets,

TABLE IV

ACCURACY $\pm$ STANDARD DEVIATION (%) OF NODE CLASSIFICATION ON CLEAN GRAPHS. THE BOLD AND UNDERLINED ACCURACIES ARE THE BEST AND THE SECOND BEST ONES, RESPECTIVELY.

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	83.09 $\pm$ 0.95	72.16 $\pm$ 0.87	85.92 $\pm$ 0.73	91.02 $\pm$ 0.28	**90.13 $\pm$ 0.55**	93.10 $\pm$ 0.39
EdgePruner (G)	81.96 $\pm$ 1.18	70.29 $\pm$ 1.62	83.92 $\pm$ 0.63	90.39 $\pm$ 0.44	90.12 $\pm$ 0.60	**93.52 $\pm$ 0.33**
EdgePruner$^{\mathrm{NF}}$ (A)	83.40 $\pm$ 0.57	72.55 $\pm$ 0.72	85.77 $\pm$ 0.64	91.03 $\pm$ 0.47	89.87 $\pm$ 0.55	92.74 $\pm$ 0.59
EdgePruner$^{\mathrm{NF}}$ (G)	81.58 $\pm$ 0.98	69.64 $\pm$ 1.74	82.80 $\pm$ 0.74	89.81 $\pm$ 0.52	89.39 $\pm$ 0.43	93.37 $\pm$ 0.27
Baseline (A)	82.65 $\pm$ 0.75	72.25 $\pm$ 0.91	**87.95 $\pm$ 0.58**	91.93 $\pm$ 0.41	89.80 $\pm$ 0.63	92.83 $\pm$ 0.41
Baseline (G)	82.73 $\pm$ 1.21	70.46 $\pm$ 1.32	85.61 $\pm$ 0.72	88.79 $\pm$ 0.43	89.15 $\pm$ 0.71	93.11 $\pm$ 0.30
Baseline$^{\mathrm{NF}}$ (A)	82.73 $\pm$ 1.02	71.99 $\pm$ 0.67	87.20 $\pm$ 0.78	**92.34 $\pm$ 0.36**	89.39 $\pm$ 0.75	92.64 $\pm$ 0.30
Baseline$^{\mathrm{NF}}$ (G)	82.99 $\pm$ 1.03	70.28 $\pm$ 1.64	86.20 $\pm$ 0.65	88.96 $\pm$ 0.38	88.85 $\pm$ 0.58	92.92 $\pm$ 0.32
GNNGuard	79.69 $\pm$ 1.56	69.85 $\pm$ 1.36	85.96 $\pm$ 0.56	91.88 $\pm$ 0.54	88.69 $\pm$ 0.71	92.81 $\pm$ 0.44
GNNJaccard	82.18 $\pm$ 0.62	70.41 $\pm$ 1.01	86.15 $\pm$ 0.43	92.22 $\pm$ 0.31	87.47 $\pm$ 0.98	92.68 $\pm$ 0.46
ARIEL	83.97 $\pm$ 0.83	**72.69 $\pm$ 0.74**	87.38 $\pm$ 0.70	92.08 $\pm$ 0.41	89.50 $\pm$ 0.57	92.52 $\pm$ 0.35
GCA	**84.12 $\pm$ 1.24**	70.40 $\pm$ 1.38	86.27 $\pm$ 0.51	90.78 $\pm$ 0.45	88.87 $\pm$ 0.49	93.07 $\pm$ 0.39

which is a limitation of EdgePruner at this stage. However, most importantly, when the same GCL method is utilized, the accuracies of the proposed methods on clean graphs are always high compared with the accuracies of GCA and ARIEL on poisoned graphs as shown in TABLE III, which means that applying EdgePruner to clean graphs yields better situations than being attacked. For example, EdgePruner (G) achieves 90.39% accuracy on clean graphs of Amazon-Photos, which is 9.29% higher than the accuracy of GCA on the poisoned graph of Amazon-Photo. For the other five datasets, EdgePruner (G) on clean graphs outperforms GCA on poisoned graphs. These results also apply to EdgePruner (A). The reason for this could be that deleting edges is less detrimental to the graph structure unlike attacks.

These results mean that EdgePruner can raise lower bounds of accuracies on poisoned graphs while maintaining acceptable accuracies on clean graphs, which answers the research question (3). Hence, EdgePruner is worth utilizing regardless of whether input graphs are clean or poisoned because it prevents GCL methods from getting trapped into the worst case.

D. Effect of Pruning Rates

In this subsection, we evaluate the relationship between the number of pruned edges and the node classification accuracy. Fig. 3 shows the accuracy of node classification on poisoned graphs as a function of the pruning rate. The pruning rate means the rate of deleted edges to all the edges in a graph. In this experiment, we report the accuracies when pruning rates are 1%, 3%, 5%, 7%, and 10%. As shown in Fig. 3, as the pruning rate is increased, the accuracies are also increased in most cases. EdgePruner (A) tends to be always more competent than EdgePruner$^{\mathrm{NF}}$ (A) at most pruning rates except for Amazon-Photo and Amazon-Computers (comparing red lines with purple dotted ones). EdgePruner (G) also has a similar tendency compared with EdgePruner$^{\mathrm{NF}}$ (G) (comparing blue lines with green dotted ones) as with EdgePruner (A). For example, as we can see from red and blue lines in Fig. 3(f), the accuracies of EdgePruner (A) and EdgePruner (G) on Coauthor-Physics are considerably improved as the pruning rates are increased. In particular, the accuracy of EdgePruner (A) on Coauthor-Physics is increased by approximately 5% when the pruning rates change from 1% to 10%. Furthermore, EdgePruner (A) achieves the similar performance on Coauthor-CS as shown in Fig. 3(e). On the other hand, as we can see from the green dotted line and the purple dotted one in Fig. 3(a), the accuracies of EdgePruner$^{\mathrm{NF}}$ (A) and EdgePruner$^{\mathrm{NF}}$ (G) on Cora decrease gradually as the pruning rate increases whereas EdgePruner (A) and EdgePruner (G) finally improve accuracies when the pruning rate is 10%.

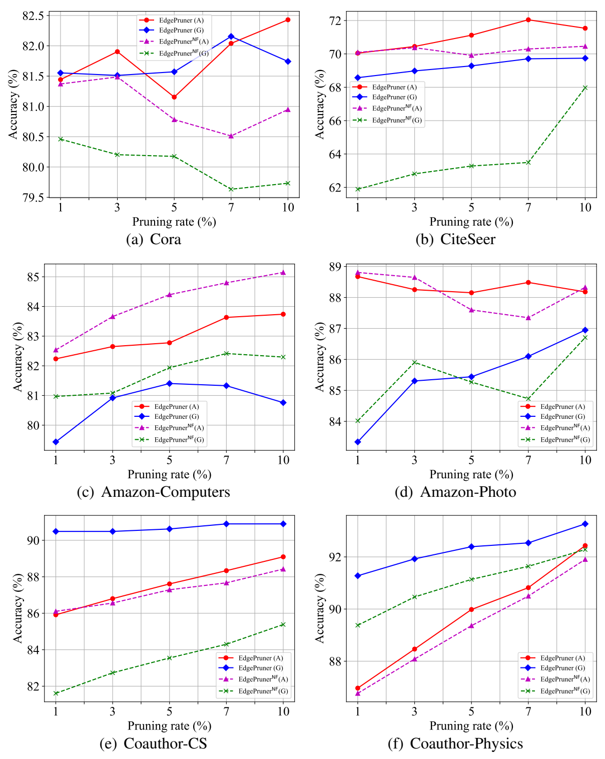


Fig. 3. Accuracies on poisoned graphs as a function of the pruning rate.

In the following, we discuss the meaning of our loss based graph selection utilized in EdgePruner on the basis of the

TABLE V

PRUNING RATES (%) WHEN THE FINAL S_{opt} IS OBTAINED. (EDGEPRUNER IS APPLIED TO POISONED GRAPHS)

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	1.89	7.26	9.87	3.23	9.69	9.85
EdgePruner (G)	10.00	7.03	9.78	0.53	8.68	9.53
EdgePruner$^{\text{NF}}$ (A)	7.77	0.73	9.26	3.23	8.31	9.85
EdgePruner$^{\text{NF}}$ (G)	7.35	0.05	8.05	0.02	0.10	3.35

pruning rate. TABLE V shows the pruning rates when the final S_{opt} is obtained in EdgePruner on poisoned graphs. As we can see from TABLE V, pruning rates are different depending on the variants and datasets. In comparison with the pruning rates in Fig. 3 and those in TABLE V, it is observed that the best variant on each dataset obtains the final S_{opt} around the pruning rates when better accuracies are achieved in Fig. 3. For example, EdgePruner (A) obtains the final S_{opt} on CiteSeer when the pruning rate is 7.26% as shown in TABLE V, which is almost consistent with the result that EdgePruner (A) attains approximately 72% accuracy at 7% pruning rate as shown in Fig. 3(b). Similarly, in the case of Coauthor-Physics, EdgePruner (G) gets the final S_{opt} when the pruning rate is 9.53%, and the accuracy is the highest at 10% pruning rate as shown in Fig. 3(f). On the other hand, there are cases where the pruning rates are inconsistent. For example, EdgePruner (A) obtains the final S_{opt} on Cora when the pruning rate is 1.89% although the highest accuracy is yielded when the pruning rate is 10% as shown in Fig. 3(a). This means that adopting graphs when the pruning rate is 10% is better.

However, we consider that utilizing the minimum losses to select the final graphs is more effective. This is because this loss based graph selection is helpful in alleviating the bad influence on clean graphs. We also evaluate the relationship between the number of pruned edges and the accuracies of node classification on clean graph. Fig. 4 shows the accuracy of node classification on clean graphs as a function of the pruning rate. As shown in Fig. 4, the accuracies tend to be decreased as the pruning rate is increased. In particular, the tendency is clearly observed on Cora, Amazon-Computers, and Amzoon-Photo as shown in Fig. 4(a), Fig. 4(c), and Fig. 4(d), respectively. TABLE VI shows the pruning rates when the final S_{opt} is obtained in EdgePruner on clean graphs. As shown in TABLE VI, on Amazon-Photo, EdgePruner (A) gets the final S_{opt} when the pruning rate is 3.54%. Similarly, EdgePruner (A) obtains the final S_{opt} when the pruning rate is 6.56% on Cora. The accuracies around them are higher compared with those when the pruning rate is 10% as shown in Fig. 4(d). Meanwhile, as shown in Fig. 4(e), the accuracies of EdgePruner (A) and EdgePruner (G) on the clean graph of Coauthor-CS are increased as the pruning rate is increased. These accuracies are the highest when the pruning rate is 10%. In such a dataset, EdgePruner (A) and EdgePruner (G) adopt the final S_{opt} at the pruning rates that are close to 10% as shown in TABLE VI. These results mean that EdgePruner can select better graphs as much as possible on the basis of the

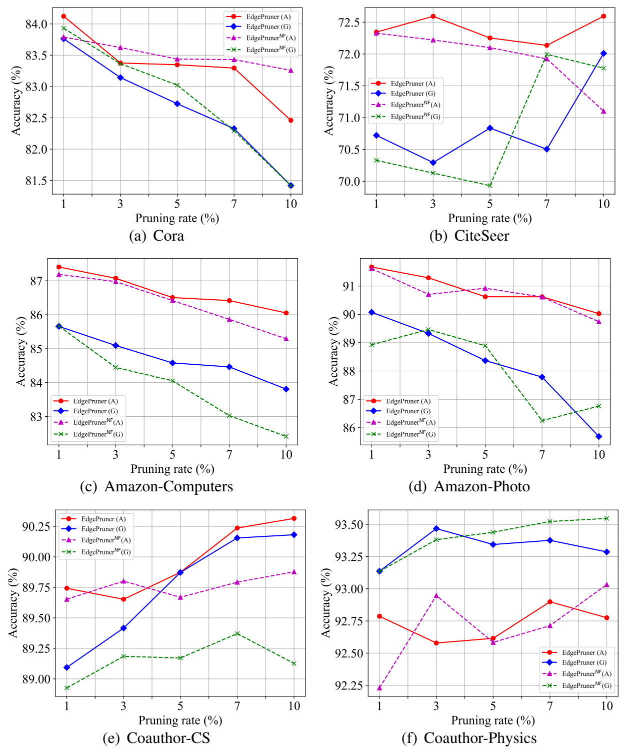


Fig. 4. Accuracies on clean graphs as a function of the pruning rate.

losses on clean graphs. Thus, our loss based graph selection in EdgePruner is more effective than adopting graphs after a designated number of edges are always pruned.

From these results, we conclude that pruning more edges in poisoned graphs basically contributes to improvement to the quality of the embeddings although pruning too many edges in clean graphs may degrade the quality on clean graphs, which answers the research question (4).

E. Adaptive Attack

In this subsection, we evaluate the node classification accuracies under an adaptive attack against EdgePruner. We assume that an attacker knows that the pruning rate utilized in EdgePruner is 10%. In the adaptive attack, poisoned graphs are created by modifying 15% of edges in clean graphs in order to make it difficult to eliminate adverse effect by the poisoning attack. We evaluate the node classification accuracies in the case where EdgePruner(G) is in place. As with other experiments, the proposed methods are allowed to prune up to 10% of edges in the poisoned graphs. TABLE VII shows the

TABLE VI
PRUNING RATES (%) WHEN THE FINAL S_{opt} IS OBTAINED. (EDGEPRUNER IS APPLIED TO CLEAN GRAPHS)

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	6.56	3.71	8.99	3.54	8.53	4.21
EdgePruner (G)	8.09	5.91	8.71	0.07	8.16	3.68
EdgePruner$^{\text{NF}}$ (A)	6.56	0.09	7.39	3.54	8.14	9.89
EdgePruner$^{\text{NF}}$ (G)	9.45	2.77	7.94	0.07	8.16	3.68

TABLE VII
ACCURACY $\pm$ STANDARD DEVIATION (%) OF NODE CLASSIFICATION UNDER ADAPTIVE CLGA. THE BOLD IS THE BEST ACCURACY.

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (G)	**78.89** $\pm$ 0.78	**63.92** $\pm$ **2.08**	**79.41** $\pm$ **0.62**	**83.45** $\pm$ **0.40**	**85.15** $\pm$ **0.60**	**88.54** $\pm$ **0.24**
GCA	77.22 $\pm$ 1.19	59.61 $\pm$ 1.85	75.68 $\pm$ 0.89	82.02 $\pm$ 0.68	78.58 $\pm$ 0.75	86.60 $\pm$ 0.47

node classification accuracy under adaptive CLGA. As shown in TABLE VII, EdgePruner (G) improves node classification accuracy on all the datasets. However, the accuracies are less improved compared with the situation of normal CLGA. This is because pruning edges are not sufficient to eliminate the adverse effects of adaptive CLGA. This result demonstrates that EdgePruner is effective in sanitizing poisoned graphs created by the adaptive attack to some extent.

However, we believe that the adaptive attack against our defense is infeasible in the case where EdgePruner is allowed to prune all the edges in a graph. In other words, this case means that the pruning rate is set at 100%. This case is possible because EdgePruner does not necessarily prune all the edges even in such a case. EdgePruner can flexibly select a sanitized graph on the basis of the minimum loss. Furthermore, defenders who utilize EdgePruner can also adopt another graph on the basis of the validation accuracy if they have a small amount of validation data consisting of labeled graphs. Thus, in this case, an attacker cannot set a poisoning rate so that it is larger than the pruning rate utilized in EdgePruner. For these reasons, we conclude that EdgePruner is effective against the adaptive attack from the perspectives of both our empirical results and the logic, which answers the research question (5).

F. Discussion

Analysis of Pruned Edges in Poisoned Graph Our method can improve the the node classification accuracies on poisoned graphs. However, the accuracies are not completely improved to the level comparable to the ones on clean graphs. We analyze the reason, and the results are in Appendix L.

Performance of Link Prediction We evaluate the performance of link prediction of proposed variants and the existing GCL methods on poisoned graphs. The detailed results are in Appendix M.

The Effect of Edge Addition for Sanitizing Poisoned Graph In our experiments, it is observed that poisoned graphs against GCL are mainly created by adding poisoned edges in a graph. This is why our experimental results show that our concept of pruning edges is effective. However, edge deletion is also conducted when poisoned graphs on Amazon datasets are created. Thus, we conduct some experiments in order to confirm the effects of edge addition in the defense. We evaluate the effect of edge addition for sanitizing poisoned graphs in the case where our method conducts both addition and deletion. The detailed results are in Appendix N.

VII. CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

In this paper, we have proposed EdgePruner, poisoned edge pruning in GCL. EdgePruner prunes edges that contribute to minimizing the contrastive loss. EdgePruner can achieve more improvement to the quality of the embeddings under the poisoning attack. Our experimental results demonstrate that pruning adversarial edges in poisoned graphs is feasible on the six datasets. We demonstrate that the feature similarity of neighboring nodes is basically effective in determining pruned edges on various datasets. Furthermore, EdgePruner can also improve the quality of embedding even on the adaptive attack.

However, our method have a main limitation, that is the fact that EdgePruner cannot completely deal with the addition of edges, which may contribute to further improvement. At this stage, EdgePruner cannot sufficiently restore edges deleted by attacks. Simply executing both the addition and the deletion of edges depending on situations is not so effective in our additional experiment. However, we found that controlling edge addition is relatively promising way for defense. We leave devising solutions for that to our future work.

Given the circumstances that there is no study that focuses on edge pruning in the GCL domain, we have shown its feasibility. As a first step of the sanitization strategy against poisoning attacks on GCL, our work and experimental results can motivate researchers to design more effective defense methods for GCL. To devise more practical countermeasures in the GCL domain, we plan to study how to more effectively restore clean graphs in the future.

REFERENCES

[1] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, 2019, pp. 417–426.

[2] S. Zhang, H. Chen, X. Ming, L. Cui, H. Yin, and G. Xu, "Where are we in embedding spaces? a comprehensive analysis on network embedding approaches for recommender systems," *arXiv preprint arXiv:2105.08908*, 2021.

[3] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[4] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[5] C. Tu, X. Zeng, H. Wang, Z. Zhang, Z. Liu, M. Sun, B. Zhang, and L. Lin, "A unified framework for community detection and network representation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 6, pp. 1051–1065, 2018.

[6] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," in *International Conference on Learning Representations*, 2018.

[7] Q. Wu, H. Zhang, X. Gao, P. He, P. Weng, H. Gao, and G. Chen, "Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems," in *The world wide web conference*, 2019, pp. 2091–2102.

[8] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.

[9] X. Wang, H. Jin, A. Zhang, X. He, T. Xu, and T.-S. Chua, "Disentangled graph collaborative filtering," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 2020, pp. 1001–1010.

[10] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph neural networks in recommender systems: a survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[11] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax." *ICLR (Poster)*, vol. 2, no. 3, p. 4, 2019.

[12] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[14] R. Linsker, "Self-organization in a perceptual network," *Computer*, vol. 21, no. 3, pp. 105–117, 1988.

[15] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *arXiv preprint arXiv:1803.07728*, 2018.

[16] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," *Advances in neural information processing systems*, vol. 32, 2019.

[17] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[18] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*. Springer, 2020, pp. 776–794.

[19] J. Li, P. Zhou, C. Xiong, and S. C. Hoi, "Prototypical contrastive learning of unsupervised representations," *arXiv preprint arXiv:2005.04966*, 2020.

[20] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *Advances in neural information processing systems*, vol. 33, pp. 9912–9924, 2020.

[21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[22] A. Mnih and K. Kavukcuoglu, "Learning word embeddings efficiently with noise-contrastive estimation," *Advances in neural information processing systems*, vol. 26, 2013.

[23] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[24] L. Kong, C. d. M. d'Autume, W. Ling, L. Yu, Z. Dai, and D. Yogatama, "A mutual information maximization perspective of language representation learning," *arXiv preprint arXiv:1910.08350*, 2019.

[25] H. Fang, S. Wang, M. Zhou, J. Ding, and P. Xie, "Cert: Contrastive self-supervised learning for language understanding," *arXiv preprint arXiv:2005.12766*, 2020.

[26] J. Giorgi, O. Nitski, B. Wang, and G. Bader, "Declutr: Deep contrastive learning for unsupervised textual representations," *arXiv preprint arXiv:2006.03659*, 2020.

[27] Z. Chi, L. Dong, F. Wei, N. Yang, S. Singhal, W. Wang, X. Song, X.-L. Mao, H. Huang, and M. Zhou, "Infoxlm: An information-theoretic framework for cross-lingual language model pre-training," 2021.

[28] S. Feng, B. Jing, Y. Zhu, and H. Tong, "Adversarial graph contrastive learning with information regularization," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1362–1371.

[29] S. Zhang, H. Chen, X. Sun, Y. Li, and G. Xu, "Unsupervised graph poisoning attack via contrastive loss back-propagation," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1322–1330.

[30] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples on graph data: Deep insights into attack and defense," *arXiv preprint arXiv:1903.01610*, 2019.

[31] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 66–74.

[32] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[34] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[36] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[37] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[38] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International conference on machine learning*. PMLR, 2020, pp. 4116–4126.

[39] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *Advances in neural information processing systems*, vol. 33, pp. 5812–5823, 2020.

[40] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, "Gcc: Graph contrastive coding for graph neural network pre-training," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 1150–1160.

[41] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[42] ——, "Graph contrastive learning with adaptive augmentation," in *Proceedings of the Web Conference 2021*, 2021, pp. 2069–2080.

[43] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin, "Topology attack and defense for graph neural networks: An optimization perspective," *arXiv preprint arXiv:1906.04214*, 2019.

[44] 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*, 2018, pp. 2847–2856.

[45] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *International Conference on Learning Representations (ICLR)*, 2019.

[46] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 695–704.

[47] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[48] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on machine learning*. PMLR, 2016, pp. 40–48.

[49] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," *Advances in neural information processing systems*, vol. 33, pp. 9263–9275, 2020.

[50] "sklearn.linear_model.LogisticRegression," https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

APPENDIX

A. Overview of Graph Representation Learning

1) Overview of GCN: In the GCN, to alleviate problems such as numerical instabilities and exploding or vanishing gradients during training, $\boldsymbol{A}$ is transformed into the symmetrically normalized adjacency matrix

$$\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}}, \tag{10}$$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$ is the adjacency matrix where self-connections are added to by the identity matrix $\boldsymbol{I}_n$. $\tilde{\boldsymbol{D}}$ is the diagonal degree matrix of $\tilde{\boldsymbol{A}}$, which means $\boldsymbol{D}[i,j] = \sum_j \hat{\boldsymbol{A}}[i,j]$. The encoder of the two-layer GCN is represented as

$$f(\boldsymbol{A}, \boldsymbol{X}) = \sigma(\hat{\boldsymbol{A}} \sigma(\hat{\boldsymbol{A}} \boldsymbol{X} \boldsymbol{W}^{(1)}) \boldsymbol{W}^{(2)}), \tag{11}$$

where $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ are the trainable weights of the first and second layers, respectively. Additionally, σ is an activation function.

2) Overview of GCA: We introduce how to learn node embeddings following GCA model [42]. In a typical GCL, there are three steps to learn $f(\boldsymbol{A}, \boldsymbol{X})$, namely (1) obtaining two views, (2) obtaining node embeddings, and (3) optimizing the parameters of $f(\boldsymbol{A}, \boldsymbol{X})$ on the basis of contrastive loss. First of all, two augmented views $G_1 = (\boldsymbol{A}_1, \boldsymbol{X}_1)$ and $G_2 = (\boldsymbol{A}_2, \boldsymbol{X}_2)$ are generated from G through two stochastic augmentations t_1 and t_2, respectively. Typical augmentation strategies include feature masking, edge dropping, and subgraph extraction, to name a few. After obtaining the two views, they are fed into a shared encoder $f(\boldsymbol{A}, \boldsymbol{X})$ to obtain node embeddings. Let $\boldsymbol{H}_\theta^m = f_\theta(\boldsymbol{A}_m, \boldsymbol{X}_m)$ denote the node embedding matrix transformed from G_m by f with parameters θ. A contrastive loss is calculated for each node by using node embeddings in $\boldsymbol{H}_\theta^m$. The contrastive loss is designed to gather similar nodes and to push dissimilar nodes away from each other. In particular, the contrastive loss for the i-th node in the first view is calculated as

$$l(h_i^1, h_i^2) = -\log \frac{e^{\beta(h_i^1, h_i^2)/\tau}}{e^{\beta(h_i^1, h_i^2)/\tau} + \sum_{j \neq i} e^{\beta(h_i^1, h_j^1)/\tau} + e^{\beta(h_i^1, h_j^2)/\tau}}, \tag{12}$$

where $h_i^1 = \boldsymbol{H}_\theta^1[i,:]$ and $h_i^2 = \boldsymbol{H}_\theta^2[i,:]$ denote the embeddings of the i-th node in $G_1 = (\boldsymbol{A}_1, \boldsymbol{X}_1)$ and $G_2 = (\boldsymbol{A}_2, \boldsymbol{X}_2)$, respectively. $\beta(\cdot)$ is a similarity function such as cosine similarity. τ is a hyperparameter called temperature parameter. Note that $l(h_i^2, h_i^1)$ must also be calculated because the above loss is nonsymmetric for h_i^1 and h_i^2. Therefore, the final loss $\mathcal{L}_{GCL}$ is calculated as

$$\mathcal{L}_{GCL}(\boldsymbol{H}_\theta^1, \boldsymbol{H}_\theta^2) = \frac{1}{2N} \sum_{i=1}^{N} [l(h_i^1, h_i^2) + l(h_i^2, h_i^1)]. \tag{13}$$

Finally, θ are updated so that $\mathcal{L}_{GCL}(\cdot,\cdot)$ is minimized, which makes $f_\theta(\boldsymbol{A}, \boldsymbol{X})$ output similar node embeddings for similar nodes.

B. Modified Edges in Poisoned Graphs for ARIEL

TABLE VIII shows the number of edges that modified when poisoned graphs are created on the basis of the loss of ARIEL. The results are similar to the ones in the case where poisoned graphs are created on the basis of the loss of GCA. The different tendency is observed on CiteSeer. As we can see from TABLE VIII, the number of deleted edges are increased and larger than the number of added edges on CiteSeer although there is no deleted edge on CiteSeer in TABLE I. We consider deleting edges is less effective than adding edges according to our experience. This is why the attack performance for ARIEL on CiteSeer is not high in our experiment compared with the attack for GCA.

C. Notation and Terminology for Our Algorithm

TABLE IX shows notations and their descriptions in Algorithm 1 for helping readers understand our method.

D. Dataset

TABLE X shows the dataset summary. These datasets are from PyTorch Geometric[2]. Cora and CiteSeer [48] are citation networks where nodes and edges correspond to documents and their citations, respectively.

Amazon-Computers and Amazon-Photo [3] are extracted from the copurchase graph in Amazon. In these graphs, nodes represent goods, and edges mean that two goods are frequently bought together.

Coauthor-CS and Coauthor-Physics [3] are the coauthorship graphs. Nodes mean authors, and they are connected by an edge if there are the coauthorship between them on a paper.

TABLE X
SUMMARY OF DATASET USED IN OUR EXPERIMENTS. THE NUMBER OF NODES, EDGE, FEATURES, AND CLASSES ARE LISTED.

Dataset name	**Nodes**	**Edges**	**Features**	**Classes**
Cora	2,708	5,429	1,433	7
CiteSeer	3,327	4,732	3,703	6
Amazon-Computers	13,752	245,861	767	10
Amazon-Photo	7,650	119,081	745	8
Coauthor-CS	18,333	81,894	6,805	15
Coauthor-Physics	34,493	247,962	8,415	5

E. Setup of GCL

In both GCA and ARIEL, two-layer GCN is employed as the encoder as with the settings in GCA and ARIEL. The implementation of GCA is based on the public implementation shared by the authors of CLGA because codes of GCA is also provided for evaluating the quality of the embeddings. We also utilize the public implementations of ARIEL. We adopt the hyperparameters used in the implementation of ARIEL, including

[2]All the datasets are from PyTorch Geometric 1.13.1 (https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html)

TABLE VIII
THE NUMBER OF EDGES THAT ARE ADDED AND DELETED WHEN POISONED GRAPHS ARE CREATED ON THE BASIS OF THE LOSS OF ARIEL.

	Cora			CiteSeer			Amazon-Computers			Amazon-Photo			Coauthor-CS			Coauthor-Physics		
	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
Added edges	52	263	528	29	106	216	338	1530	2869	496	2471	4898	61	309	619	52	258	519
Deleted edges	0	0	0	16	121	239	18	253	699	0	9	62	0	0	0	0	4	6

TABLE IX
NOTATIONS AND DESCRIPTION IN ALGORITHM 1.

Notation	Description
$\boldsymbol{A}^{'}$	A poisoned adjacency matrix
$\boldsymbol{X}$	A node feature matrix
$G' = (\boldsymbol{A}^{'}, \boldsymbol{X})$	A poisoned graph inputted into a GCL method
$\boldsymbol{S}$	A sanitized adjacency updated by pruning procedures
f	An encoder trained by a GCL method
$\boldsymbol{S}_m^k$	A m-th augmented sanitized adjacency matrix at the k-th augmentation out of M views created by a GCL method
$\boldsymbol{X}_m^k$	A m-th augmented node feature matrix at the k-th augmentation out of M views created by a GCL method
$G_m^k = (\boldsymbol{S}_m^k, \boldsymbol{X}_m^k)$	A m-th graph view obtained through k-th augmentation out of M views created by a GCL method
$\theta^{'}$	Parameters of f trained in a GCL method
$\nabla_m^k = \frac{\partial \mathcal{L}_s^k}{\partial \boldsymbol{S}_m^k}$	A gradient matrix with regard to $\boldsymbol{S}_m^k$ at the k-th augmentation
$\nabla^k = \sum_m^M \nabla_m^k$	A total gradient matrix obtained by adding up ∇_m^k at the k-th augmentation
∇_{total}	A total gradient matrix obtained by adding up ∇^k over K times augmentations
$\boldsymbol{C}$	Candidate edges to prune in each iteration
$\mathcal{L}_s$	A contrastive loss of f that trains $\boldsymbol{S}$
$\mathcal{L}_{\text{min}}$	The minimum contrastive loss during our pruning procedures
$\boldsymbol{S}_{\text{opt}}$	A optimal adjacency matrix that produces the minimum contrastive loss
$G_s = (\boldsymbol{S}_{\text{opt}}, \boldsymbol{X})$	A sanitized graph consisting of $\boldsymbol{S}_{\text{opt}}$ and $\boldsymbol{X}$ after pruning up to N edges in a graph

the number of training epochs, the temperature parameter, edge dropping rates, and feature dropping rates in both GCA and ARIEL. This is because utilizing the hyperparameters of ARIEL yields better node classification accuracy on clean graphs in our preliminary experiments.

F. Setup of CLGA

In terms of four large datasets, namely Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics, we could not create poisoned graphs from their entire graphs by CLGA due to the memory limitations. This is why we randomly sample a subgraph of 5,000 nodes for each of the above large graphs in accordance with the experimental setup in ARIEL. The poisoned graphs are created on the basis of the subgraph. We utilize the public implementation shared by the authors of CLGA in order to create poisoned graphs. Poisoned graphs are created by modifying edges so that the contrastive loss of GCL methods is maximized. In our experiments, it is assumed that GCA and ARIEL are attacked. As for hyperparameters of GCA, except for the number of training epochs, we utilize the same hyperparameters as the ones used in the implementation of ARIEL when poisoned graphs are created. We set the number of epochs for retraining at each iteration at one in our experiments because we can considerably reduce the computational cost while obtaining comparable attack performance to the results reported in [29] in our preliminary experiments. To create the poisoned graphs, 10% percent of edges in clean graphs are modified.

G. Setup of Downstream Task

We evaluate the quality of node embeddings produced by GCL methods. Embeddings produced by GCA and ARIEL are utilized as features of the node classification. The node classification is conducted by a logistic regression classifier that is implemented in scikit-learn [50]. When the node classification is evaluated, we split the nodes into 10%, 10%, and 80% for training, validating, and testing, respectively. As with the evaluation in [28], the node classification is evaluated on 20 random dataset splits, which means that node classification is conducted 20 times on different sets. Thus, we report the mean testing accuracy of the node classification.

H. Hyperparameters of Existing Defense Methods for GNN

We utilize the codes shared by [49] to implement GNN-Guard and GNNJaccard. We use two-layers GCN in GNN-Guard and GNNJaccard as with other GCL methods in our experiments. Basically, we utilize the same hyperparameters used in the shared codes except for the dimension of the hidden layer. We set the dimension of the hidden layer to 256 because the node classification accuracy is relatively low with a default value on some datasets. As for other hyperparameters, dropout rate, learning rate, and the number of epochs are 0.5, 0.01, and 200, respectively. In all experiments, we conduct training models with Adam optimizer. We repeat every experiment 10 times and report average results.

I. Setup of EdgePruner

EdgePruner (G) and EdgePruner (A) conduct edge pruning with the feature similarity of neighboring nodes. We tentatively utilize values of T that yield the best accuracy on the validation datasets in our preliminary experiments. As a result, we set T on Cora, CiteSeer, Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics at 0.10, 0.10, 0.30, 0.40, 0.25, and 0.25, respectively. On the other hand, EdgePruner$^{\mathrm{NF}}$ (G) and EdgePruner$^{\mathrm{NF}}$ (A) do not utilize the feature similarity. In the following experiments, we evaluate the effectiveness of the proposed methods by comparing them with the baselines and existing GCL methods, namely ARIEL and GCA.

J. Distribution of Feature Similarity

Fig. 5 shows the distribution of the cosine similarity of neighboring nodes connected by clean edges or adversarial ones. This distribution is based on the poisoned graphs created by maxmizing the loss of GCA. We confirm that the poisoned graphs for ARIEL also have similar distribution in our preliminary experiment. As shown in Fig. 5, adversarial edges on Cora, CiteSeer, Coauthor-CS, and Coauthor-Physics concentrate in the areas of lower values. On the other hand, it is observed that adversarial edges on Amazon-Computers and Amazon-Photo are relatively widely distributed compared to other datasets. According to these results, we consider that adversarial edges on the two Amazon datasets are more difficult to distinguish from clean edges even if the feature similarity is utilized.

K. Interpretation of Performance Improved by Pruning Edges on Clean Graph

There are cases where the accuracies are improved when edges in clean graphs are pruned by EdgePruner. In particular, EdgePruner (A) and EdgePruner (G) improve accuracies on both Coauthor-CS and Coauthor-Physics compared with ARIEL and GCA, respectively. The reason for this could be that some noise edges included in clean graphs are removed. Since our method prunes edges so that contrastive losses are minimized, such noise edges can be removed, which results in improving the performance of the node classification. However, on the other datasets, accuracies are degraded compared to existing GCL methods in almost all of the cases. This is because importance of edge information is different depending on the datasets. According to experimental results in [37], when an unsupervised linear model is trained only with node features, node classification accuracies on Cora and CiteSeer are 47.9% and 49.3%, respectively. When both edge information and node features are leveraged, the accuracy on Cora and CiteSeer are improved by approximately 35%, and 20%, respectively. On the other hand, according to [42], node classification accuracies on Coauthor-CS and Coauthor-Physics are 90.3% and 93.5%, respectively even when only node features are utilized. These results mean that edge information is less important on Coauthor datasets. Thus, noise edges are relatively easily pruned on Coauthor datasets because useful node representation can be learned with node features, which results in improving the performance. To the contrary, the more edge information is lost, the more difficult it is for an encoder to obtain useful node representation on other datasets. This is because the encoder has to learn complex information of nodes with node features and less edges. As a result, since the quality of node embedding is easy to degrade, accuracy of node classification in downstream tasks may be degraded.

L. Analysis of Pruned Edges in Poisoned Graph

We analyze the reasons why the node classification accuracies on poisoned graphs are not completely improved to the level comparable to the ones on clean graphs even when our methods are applied to them. We evaluate the ratio of adversarial edges that are successfully pruned to all the edges pruned by our methods. TABLE XI shows the ratio of the adversarial edges to all the pruned edges. It is observed that the ratios are different depending on datasets. According to TABLE III and TABLE XI, the higher the ratio is, the more the performance tends to be improved. For example, EdgePruner (A) can prune 81.48% of adversarial edges on CiteSeer and achieves the best accuracy in TABLE III. These results mean that pruning more adversarial edges is important to restore the accuracy comparable to the accuracy on clean graphs. In order to further ameliorate the performance, other techniques may be needed, which is interesting and important future work.

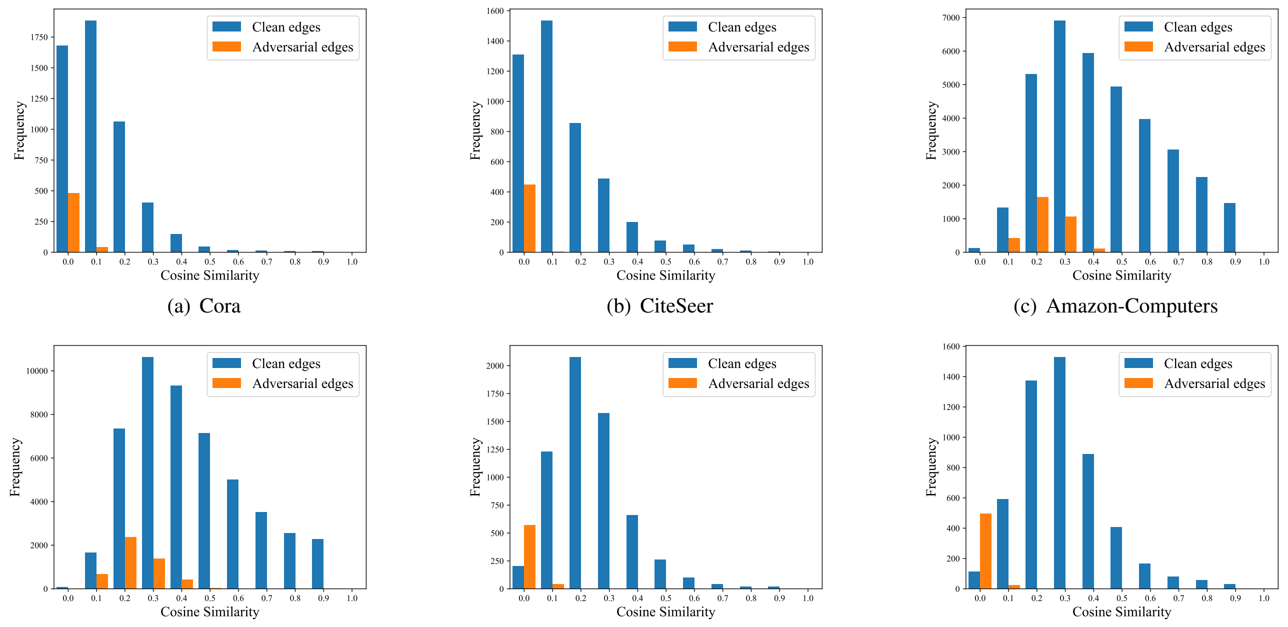
M. Performance of Link Prediction

We evaluate link prediction performance of proposed variants on poisoned graphs. We use a two-layer MLP as the projection head so as to project the embeddings into a new latent space following the setup in [29]. For all the datasets, we split the edges into 70%, 20%, and 10% for training, testing, and validating set, respectively. We report the area under curve (AUC) score. We repeat each experiment 5 times and report the average AUC and standard deviation. TABLE XII shows the results of link prediction. EdgePruner (A) and EdgePruner (G) achieve the best accuracies on Amazon-Computers and Cora, respectively. On the other hand, baseline variants yield the best accuracies on CiteSeer, Amazon-Photo, Coauthor-CS, and Couthor-Physics. From these results, baseline variants outperform EdgePruner variants in terms of the link prediction task. However, the difference is relatively marginal. Furthermore, as mentioned in Section VI-B, baselines are vulnerable to the adaptive attack described in Section VI-E. Therefore, we consider that EdgePruner is more competent overall.

N. Effects of Edge Addition on Poisoned Graph

Although our method is dedicated to edge pruning, we evaluate the effects of adding edges on poisoned graphs of Amazon datasets. We call the method that conducts both edge addition and deletion "EdgeModifier". EdgeModifier mainly conducts edge addition in accordance with the conditions opposite to the ones for edge deletion. The condition regarding gradients for selecting edges to add is represented as

$$\boldsymbol{S}[i,j] = 0 \wedge \nabla_{\mathrm{total}}[i,j] < 0. \tag{14}$$



(a) Cora (b) CiteSeer (c) Amazon-Computers

(d) Amazon-Photo (e) Coauthor-CS (f) Coauthor-Physics

Fig. 5. Distribution of the cosine similarity of nodes connected by clean edges or adversarial ones.

TABLE XI
THE RATIO OF ADVERSARIAL EDGES TO ALL THE PRUNED EDGES (%). THE BOLD RATIO IS THE HIGHEST ONE.

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	11.93	**81.48**	67.06	24.11	78.35	**91.52**
EdgePruner (G)	**50.00**	63.08	63.89	0.87	**80.10**	31.93
EdgePruner$^{\mathrm{NF}}$ (A)	29.17	3.24	**82.99**	**25.79**	57.84	85.74
EdgePruner$^{\mathrm{NF}}$ (G)	26.33	18.46	67.36	0.22	53.56	28.87

TABLE XII
AVERAGE AUC ± STANDARD DEVIATION (%) OF LINK PREDICTION UNDER CLGA. THE BOLD AND UNDERLINED ACCURACIES ARE THE BEST AND THE SECOND BEST ONES, RESPECTIVELY.

Method	Cora	CiteSeer	Amazon-Computers	Amazon-Photo	Coauthor-CS	Coauthor-Physics
EdgePruner (A)	98.02 ± 0.13	99.13 ± 0.09	**94.93 ± 0.15**	95.12 ± 0.13	99.08 ± 0.08	98.88 ± 0.13
EdgePruner (G)	**98.49 ± 0.11**	99.72 ± 0.05	94.25 ± 0.09	93.74 ± 0.13	99.53 ± 0.05	98.97 ± 0.06
EdgePruner$^{\mathrm{NF}}$ (A)	98.04 ± 0.12	99.06 ± 0.08	94.85 ± 0.11	95.18 ± 0.05	99.14 ± 0.05	98.78 ± 0.06
EdgePruner$^{\mathrm{NF}}$ (G)	97.94 ± 0.10	99.70 ± 0.02	94.50 ± 0.08	93.53 ± 0.10	99.38 ± 0.08	98.95 ± 0.11
Baseline (A)	98.36 ± 0.10	99.34 ± 0.09	$\underline{94.86 \pm 0.08}$	$\underline{95.56 \pm 0.05}$	99.22 ± 0.03	99.15 ± 0.07
Baseline (G)	$\underline{98.41 \pm 0.13}$	$\underline{99.77 \pm 0.03}$	94.37 ± 0.10	94.85 ± 0.09	**99.55 ± 0.05**	**99.24 ± 0.07**
Baseline$^{\mathrm{NF}}$ (A)	98.36 ± 0.10	99.34 ± 0.09	94.77 ± 0.08	**95.68 ± 0.04**	99.22 ± 0.03	99.15 ± 0.07
Baseline$^{\mathrm{NF}}$ (G)	98.41 ± 0.13	**99.78 ± 0.03**	94.45 ± 0.11	94.78 ± 0.11	**99.55 ± 0.05**	$\underline{99.23 \pm 0.07}$
ARIEL	97.85 ± 0.06	98.95 ± 0.08	94.46 ± 0.18	94.16 ± 0.03	98.97 ± 0.06	98.75 ± 0.16
GCA	97.88 ± 0.10	99.65 ± 0.03	93.56 ± 0.09	92.23 ± 0.12	99.43 ± 0.04	98.90 ± 0.15

The condition regarding feature similarity for selecting edges to add is represented as

$$\mathrm{s}(\boldsymbol{X}[i,:], \boldsymbol{X}[j,:]) > T. \tag{15}$$

On the other hand, the conditions for selecting edges to delete are Eq (6) and Eq (7) as with EdgePruner. We mainly compare two types of EdgeModifier in this experiment. The first method simply modifies an edge that has the largest gradient out of edges satisfying the conditions. We simply call this method EdgeModifier. The other method is EdgeModifier$^{\mathrm{b}}$ that conducts edge modification on the basis of bernoulli sampling in order to avoid adding too many edges. This is because deleting edges is more important from the perspective of defense. To the contrary, deleting edges is less important from the perspective of the poisoning attack. An adjacency matrix is basically sparse, which means that there are many zero values. In other words, modifying values from 1 to 0

TABLE XIII
ACCURACY OF NODE CLASSIFICATION ON CLGA WHEN BOTH EDGE ADDITION AND DELETION ARE ALLOWED IN EDGEMODIFIER. THE BOLD AND UNDERLINED ACCURACIES ARE THE BEST AND THE SECOND BEST ONES, RESPECTIVELY.

Method	**Amazon-Computers**	**Amazon-Photo**
EdgeModifier (A)	82.39 ± 0.91	87.60 ± 0.37
EdgeModifier (G)	81.02 ± 0.58	82.16 ± 0.71
EdgeModifier$^{\mathrm{NF}}$ (A)	82.53 ± 0.71	87.87 ± 0.43
EdgeModifier$^{\mathrm{NF}}$ (G)	78.39 ± 0.79	83.90 ± 0.47
EdgeModifier$^{\mathrm{b}}$ (A)	$\underline{83.16 \pm 0.75}$	$\underline{88.77 \pm 0.51}$
EdgeModifier$^{\mathrm{b}}$ (G)	81.16 ± 0.59	84.44 ± 0.44
EdgeModifier$^{\mathrm{b,NF}}$ (A)	**85.20 ± 0.50**	**88.78 ± 0.30**
EdgeModifier$^{\mathrm{b,NF}}$ (G)	82.82 ± 0.79	82.78 ± 0.52

cannot effectively affect gradients of encoders because it is a trivial change in a clean graph. As a result, edge addition tends to be conducted more frequently when poisoned graphs are created. We tentatively leverage the ratio of existing edges (the value is 1 in a adjacency matrix) to all edges as the probability of selecting addition. This is because the number of existing edges is less compared with non-existing edges in a sparse adjacency matrix, which is useful in controlling addition. To be specific, the probability that edge addition is conducted is set to 0.0015 on Amazon-Computers. On Amazon-Photo, we set the probability to 0.0022. As with EdgePruner, we evaluate four variants for each method depending on assumed GCL methods and using the feature similarity. In other words, we run experiments on eight methods. TABLE XIII shows the results of average accuracy of node classification. EdgeModifier$^{\mathrm{b,NF}}$ (A) achieves the best accuracies on both the datasets. The second best accuracies are produced by EdgeModifier$^{\mathrm{b}}$ (A). The results demonstrate that controlling edge addition is more effective on Amazon datasets. In particular, EdgeModifier$^{\mathrm{b,NF}}$ outperforms EdgePruner$^{\mathrm{NF}}$ (A), which attains the highest accuracies, 84.92% and 88.58% on Amazon-Computers and Amazon-Photo among EdgePruner variants. Thus, we consider that our method has potential for dealing with edges deleted by attacks although there is room for improvement.