Attacking Graph-Based Classification without Changing Existing Connections

Xuening Xu
Temple University
Philadelphia, PA, USA
xuening.xu@temple.edu

Xiaojiang Du
Temple University
Philadelphia, PA, USA
dux@temple.edu

Qiang Zeng
University of South Carolina
Columbia, SC, USA
ZENG1@cse.sc.edu

ABSTRACT

In recent years, with the rapid development of machine learning in various domains, more and more studies have shown that machine learning models are vulnerable to adversarial attacks. However, most existing researches on adversarial machine learning study non-graph data, such as images and text. Though some previous works on graph data have shown that adversaries can make graph-based classification methods unreliable by adding perturbations to features or adjacency matrices of existing nodes, these kinds of attacks sometimes have limitations for real-world applications. For example, to launch such attacks in real social networks, the attacker cannot force two good users to change (e.g., remove) the connection between them, which means that the attacker can not launch such attacks. In this paper, we propose a novel attack on collective classification methods by adding fake nodes into existing graphs. Our attack is more realistic and practical than the attack mentioned above. For instance, in a real social network, an attacker only needs to create some fake accounts and connect them to existing users without modifying the connections among existing users. We formulate the new attack as an optimization problem and utilize a gradient-based method to generate edges of newly added fake nodes. Our extensive experiments show that the attack can not only make new fake nodes evade detection, but also make the detector misclassify most of the target nodes. The proposed new attack is very effective and can achieve up to 100% False Negative Rates (FNRs) for both the new node set and the target node set.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Adversarial machine learning, adversarial graph-based classification, AI security

ACM Reference Format:
Xuening Xu, Xiaojiang Du, and Qiang Zeng. 2020. Attacking Graph-Based Classification without Changing Existing Connections. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3427228.3427245

ACSAC 2020, December 7–11, 2020, Austin, USA

ACM ISBN 978-1-4503-8858-0/20/12…$15.00
https://doi.org/10.1145/3427228.3427245

1 INTRODUCTION

Graph data is ubiquitous for many real-world applications, such as biological networks, social networks and communication networks. One of the most important tasks on graph data is the classification of nodes, which is critical to security analysis, such as malware detection, Sybil detection and malicious website detection.

Given a graph and a subset of nodes associated with labels, which is called a training dataset, a node classification method is used to predict labels for the rest of the unlabeled nodes. The training dataset includes some labeled positive nodes and labeled negative nodes, serving as pre-knowledge for the classification method. A node with a positive label means the node is malicious, and a node with a negative label means benign. For instance, in social networks, a positive node represents a fraudulent user and a negative node represents a normal user.

Node classification methods can be divided into two categories: graph neural networks (GNNs) [4, 6, 13, 16, 24–26, 32, 40, 42] and collective classification [3, 5, 10, 12, 14, 19, 20, 22, 23, 28, 31, 38, 41]. Graph neural networks are deep learning based methods, tailored for graph data and proposed to collectively gather information from graph structure. GNNs extract features for nodes in the graph and utilize these features to do the classification task on nodes. In contrast, collective classification methods are based on belief propagation algorithms. At the very beginning, a collective classification method assigns a prior reputation score for each node according to the training dataset and also associates edges with some proper weights. Then it starts to propagate the prior reputation scores via weighted edges to get a posterior reputation score for each node in the graph. Eventually, the collective classification method classifies these nodes based on the calculated posterior reputation scores. Prior work [36] shows collective classification outperforms GNNs for some security and privacy problems. Besides, collective classification methods were deployed in industry for different security tasks, such as fraudulent user detection in social networks [3, 5] and malware detection [7, 31].

While most existing researches on adversarial machine learning study non-graph data, such as images and audios [27, 43, 46, 47], some recent works show that node classification methods are also vulnerable to adversarial attacks, which implies these classification methods become unreliable and could misclassify nodes under some intentional and designed attacks. These recent works [2, 9, 29, 34, 44, 45] mainly focused on manipulating the graph structure by inserting new edges or deleting existing edges. However, such attacks have limitations, e.g., deleting an existing edge

between two nodes requires at least one of nodes to be a malicious node. That is, the attacker cannot force two good nodes (not being compromised) to delete the edge between them. In [39], the authors considered an attack on graph convolutional networks (GCNs) by adding fake nodes to the graph. Targeting GNNs, [30] formulated the fake node injection problem as a Markov decision process and utilized Q-learning algorithm to address this problem. However, they did not explore such an attack (adding new nodes) on collective classification methods.

In this paper, we propose a new attack targeted at a collective classification method named Linearized Loopy Belief Propagation (LinLBP), which was also explored in [15, 34, 35, 37, 38]. LinLBP addresses the limitations on convergence and scalability that LBP suffers. Besides, it is more accurate and more robust than other state-of-art collective classification methods, such as Random Walk based methods and LBP-based methods[37]. We assume that an attacker has the ability to create/insert a number of new (fake) nodes and then link them to existing nodes, without modifying the existing edges in the original graph. This assumption is reasonable. For example, in social networks such as Facebook and Twitter, an attacker can create multiple new (fake) accounts, and then connect them with existing accounts. Actually, these kinds of attacks happen every day. The attacker chooses a group of target nodes in the original graph, and the goal of the attacker is to make LinLBP misclassify the new fake nodes, as well as the chosen target nodes (e.g., existing malicious nodes), i.e., label them as negative rather than positive. To be more formal, the attacker aims to achieve a high False Negative Rate (FNR) on these nodes to evade detection. Note that the new attack we propose here does not need to modify any existing edges among existing nodes in the original graph, which means that it may be easier and more feasible for an attacker to implement this attack in real-world applications.

In this paper, we formulate the new attack as a constrained optimization problem. The objective function is the total cost of creating fake nodes and connecting them to existing nodes. The constraints are FNR=1 for the new fake nodes and the target nodes. Note that the solution for each fake node is to decide whether it should link to an existing node or not, so the variables in this optimization problem are all binary. Besides, because the constraints are highly nonlinear, we use a similar approach as in [34] to relax the constraints and approximately solve the proposed optimization problem. Specifically, we temporarily relax binary variables to continuous variables and add the highly nonlinear constraints into the objective function using Lagrangian multipliers. Our experiments on real-world graphs show that our attack can make all new nodes evade detection and also make LinLBP misclassify most of the target nodes. Moreover, our attack performs well even if the attacker lacks full knowledge of LinLBP or the training dataset.

Our contributions are summarized as follows:

- We design an attack aimed at the collective classification method by adding new fake nodes to a graph without modifying any existing edges in the original graph.
- We propose a threat model and formulate our attack as an optimization problem and use an effective solution to solve the problem.
- We evaluate our attack on real-world graphs, and the new attack turns out to be effective even if the attacker only has partial knowledge.

The rest of the paper is organized as follows. Section 2 includes a brief literature review of related work and the background about Linearized Loopy Belief Propagation. In Section 3, we introduce the threat model and describe the attack scenario. In Section 4, we formulate the problem as an optimization problem and show the detail of our attack design. Section 5 presents the evaluation of our attack on real-world graphs. We conclude in Section 6.

2 RELATED WORK AND BACKGROUND

2.1 Node Classification Methods

2.1.1 Graph Neural Network. Graph neural networks (GNNs) are deep learning based methods that have been widely applied to the graph domain. GNNs are motivated by convolutional neural networks (CNNs) [17], which can extract spatial features and utilize them to construct a highly expressive form of representations. GNNs learn the representation of graph nodes as feature vectors and use them to operate node classification.

Some graph neural network methods [6, 13, 25, 32, 42] first do the graph embedding, which aims to represent graph nodes as low-dimensional vectors while maintaining graph topology structure and node information. Then, some simple off-the-shelf machine learning algorithms can be applied to perform node classification, i.e., predict labels for the unlabeled nodes. For example, DeepWalk [25] obtains local information from truncated random walks and then takes sequences of walks as the equivalent of sentences in natural language to learn representations of nodes, utilizing the Skip-gram model [21].

Other graph neural network methods [4, 16, 26] solve graph tasks in the manner of end-to-end. The architecture of the neural network varies according to the graph structure. A neuron connecting to the neurons in the previous layer simulates a node linking its neighbors. These neurons in the hidden layers stand for feature vectors of nodes, which are iteratively computed by aggregating feature vectors of their neighbors. Finally, feature vectors are used to classify nodes in the last layer of the neural network. For instance, Graph Convolutional Network (GCN) [16] utilizes a first-order approximation of spectral convolutions on graphs to learn feature vectors in a neural network, and uses logistic regression classifiers to perform classification.

2.1.2 Collective Classification. Collective classification makes joint classifications of connected nodes to address node classification tasks. Given a graph dataset, collective classification methods define a prior reputation score for each node and then assign weights for edges in the graph. After that, the prior reputation scores are propagated along the edges to compute posterior reputation scores for nodes. Finally, the posterior reputation scores are used to classify unlabeled nodes. Different strategies can be applied for defining prior reputation scores, assigning weights, or propagating prior reputation scores. For instance, some methods use loopy belief propagation (LBP) to propagate prior reputation scores among the graphs, while others utilize random walks for propagation.

LBP-based methods [10, 12, 19, 23, 31, 38] utilize both labeled positive and labeled negative nodes in the training dataset to define prior reputation scores. In terms of weight, LBP-based methods usually set the same weight for all edges. For the propagation phase, a standard LBP or a variant version of LBP is used for computing posterior reputation scores. Nevertheless, LBP suffers limitations on convergence and scalability. Recently, some works [15, 37, 38] address these limitations by linearizing LBP to obtain a Linearized LBP (LinLBP).

Random Walk based methods [3, 5, 14, 20, 41] assign prior reputation scores to labeled nodes, which are used in the training dataset. Similar to LBP-based methods, most random walk based methods assign a constant weight for all edges in the graph. Besides, weights also can be learned by using features of nodes [3]. Random walks distribute the reputation score of a node to neighbors based on the weight distribution of its edges, and each of the neighbors sums the received reputation scores as the new reputation score.

2.2 Linearized Loopy Belief Propagation

Here, we introduce Linearized Loopy Belief Propagation [38] and describe how this classification method works. Given a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. We denote $|V|$ and $|E|$ as the number of nodes and edges, respectively. A training dataset D contains labeled positive nodes and labeled negative nodes. For each node $u \in V$ in graph G, LinLBP assigns a prior reputation score according to the following:

$$q_u = \begin{cases} \theta, & u \text{ is positive} \\ -\theta, & u \text{ is negative} \\ 0, & \text{Otherwise} \end{cases} \quad (1)$$

where $0 < \theta < 0.5$ [38]. LinLBP calculates the posterior scores in the following way:

$$\mathbf{p} = \mathbf{q} + \mathbf{A} \odot \mathbf{W}\mathbf{p} \quad (2)$$

where $\mathbf{p}$ and $\mathbf{q}$ are $|V| \times 1$ column vectors of posterior reputation scores and prior reputation scores, respectively. $\mathbf{W}$ is a $|V| \times |V|$ matrix and each entry is the weight w (between $(0, 0.5]$) for the corresponding edge. The larger the value of w, the higher the possibility that the two connected nodes have the same label. We denote $\mathbf{A}$ as the adjacency matrix of graph G. Calculating posterior reputation scores is an iterative process until the posterior reputation scores converge. After convergence, a node with posterior reputation score $p_u > 0$ is positive, i.e., it's a malicious node. The larger the value, the higher confidence that it is a malicious node.

2.3 Adversarial Machine Learning on Graphs

The past few years have seen rapid progress in adversarial machine learning, but existing studies on this field mainly focus on non-graph data, such as image and text. Only a few works [2, 8, 9, 29, 33, 34, 39, 44, 45] considered the adversarial scenario over graph-based machine learning. Some of them proposed attacks via manipulating the graph structure, e.g., inserting new edges or deleting existing edges, targeting either collective classification methods [34] or graph neural network methods [2, 9, 29, 44, 45]. For instance, Wang et al. [34] proposed an attack to change connection status among nodes. To this end, they formulated their attack as an optimization problem and solved the problem by minimizing the total cost of an attacker's operations. In [39], aiming at GCNs, Wang et al. used a greedy algorithm to add fake nodes into a graph to conduct fake node attacks. Another recent work [30] proposed node injection attacks to poison the training graph used by GNNs, in order to reduce classification accuracy of GNNs on the unlabeled nodes. To achieve that, they used a deep hierarchical reinforcement learning based method to launch these attacks.

In this paper, we target LinLBP, a collective classification method, to launch attacks by adding fake nodes without modifying existing edges. Our attack is designed for collective classification and can be seen as a complement to previous works, which aimed at graph neural networks.

3 MODEL SETUP

In this section, we present the threat model, which includes the attacker's knowledge, capability, and goal. Also, we will describe our new attack.

3.1 Threat Model

For a given graph, a LinLBP system is defined by a chosen training dataset and weights of edges among nodes. The training dataset acts as prior knowledge for LinLBP and it has a potential impact on the posterior reputation scores after the propagation process. The value of edge weight indicates the degree of influence of neighbors, and it affects the posterior reputation scores during each iteration until convergence. The attacker's knowledge may be classified based on the following two things: *training dataset* and *weight*. An attacker with full knowledge means that the attacker knows both the training dataset and the weight of LinLBP. Sometimes the attacker does not have full access to the training dataset, i.e., the attacker only has partial training dataset to launch the attack, which may still be effective. If an attacker has no idea of the LinLBP weight, a substitute weight may be used to launch an attack.

The attacker has the capability to create/insert new nodes and link them to existing nodes in the graph, as well as to some of the new nodes. The attacker does *not* need to modify existing edges in the original graph.

This is easy for the attacker to do. For example, in a real social network, the attacker only needs to register a group of fake accounts, which act like normal users, and then connect them to existing users by sending friend requests. The friend requests will be easily accepted for those who would like to expand their social networks. The attacker will determine the number of new nodes in order to perform a successful attack. Of course, the attacker will have costs due to creating new nodes and connecting them to existing nodes. Hence, we assign costs for creating new nodes and connecting to existing nodes.

If new nodes try to connect with many existing nodes, this becomes suspicious and can be easily detected (e.g., by the social network administrator). To be realistic, we use K to denote the maximum number of neighbors that each new node can have.

In order to hide its malicious identity, the new fake nodes may prefer to connect with a well-known benign node. However, for the popular benign node, it is highly suspicious to have many new connection requests in such a short time period. Therefore, to be more concealed, the attacker may set a constraint on the number

of new node connections to an existing node. We use l to denote this constraint. Specifically, if an existing node has already been connected by l new nodes, the attacker will not connect any more new nodes to that existing node.

Like prior work [34], we assume an attacker has knowledge of the graph and knows which nodes are positive nodes and negative nodes. For example, in social network, a well-developed web crawler can be used by an attacker to collect the graph information. The attacker selects a set of positive nodes as target nodes, denoted by T, and the goal is to make LinLBP misclassify both the new nodes and target nodes as negative, i.e., achieve high False Negative Rates (FNRs) for both the new node set and target node set.

Note that there need be some new edges between new nodes and target nodes in order to achieve high FNR for the target nodes. That is, each new node connects to two sets of nodes: one is the target node set and the other is the remaining nodes in the graph. For each new node, the number of edges connecting target nodes is denoted by r. The value of r could be different for different graph datasets, and it is determined by the attacker based on the properties of the graph, such as size and density.

3.2 The New Attack

Based on the threat model above, we describe the new attack as follows. Given a graph G, the attacker's knowledge, the number of new nodes n, the maximum number of neighbors K that each new node can have, constraint l, the chosen target node set T, the number of edges to target nodes r for each new node, and the costs of creating new nodes and connecting to existing nodes, our attack aims at LinLBP to make it achieve high FNRs on both the new node set and the target node set at the same time. This is done by connecting the new nodes to existing nodes without modifying the existing edges in the original graph.

4 ATTACK DESIGNS

In this section, we present the attack design with the full knowledge of LinLBP, that is, the attacker knows the entire training dataset and the weight used in LinLBP. In Section 5.2.2, we show that the assumption can be relaxed and the attack success rate is still high.

We formulate the attack as an optimization problem. The objective function is the total cost of creating new nodes and connecting the new nodes to existing nodes. The constraints are: a) FNR = 1 for the new nodes, b) FNR = 1 for the target nodes, and c) the maximum number of each new node's neighbors is bounded by K.

4.1 Notations

Table 1 lists some of the important notations used in this paper, which will be detailed in the rest of this section.

Given an undirected graph $G = (V, E)$, where $u \in V$ is a node and $(u, v) \in E$ is an edge. $|V|$ is the number of nodes in the original graph. $\mathbf{W}$ is a $|V| \times |V|$ matrix, and each entry is the weight of the edge between two nodes. S is the set of n new nodes. $G' = (V', E')$ denotes the graph after adding new nodes and new edges. Here $V' = V \cup S$ denotes the set of all nodes in graph G', and E' denotes the set of edges in the new graph G'. C_{node} is the cost of creating a new node, and C_{uv} is the cost of inserting an edge between the new node u and the other node v. We use a binary variable B_{uv} for a new node u, and a node v (v can be either an existing node or a new node) to indicate whether adding a edge between them or not. Specifically, $B_{uv} = 1$ if a new edge is added between node u and v. Note that $B_{uv} = 0$ for $u, v \in V$, since no edge is inserted between two existing nodes in the original graph. Then we define the adversarial matrix $\mathbf{B}$ as follows:

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{|V|\times|V|} & \mathbf{B}_{SV}^T \\ \mathbf{B}_{SV} & \mathbf{B}^S \end{bmatrix} \tag{3}$$

Each entry (s^{th} row and v^{th} column) of $\mathbf{B}_{SV}$ is B_{sv} and it indicates whether an edge is added between node $s \in S$ and node $v \in V$ or not.

The superscript T represents the matrix transpose operator. Each entry (i^{th} row and j^{th} column) of $\mathbf{B}^S$ is B_{ij}^S and it indicates whether an edge is added between two nodes in S. Specifically, $B_{ii}^S = 0$ for all nodes in S since no edge is added to a node itself; $B_{ij}^S = 1$ means that an edge is added between two nodes $i, j \in S$ and $B_{ij}^S = 0$ otherwise.

Recall that $\mathbf{A}$ is the adjacency matrix of the original graph G. Now we can define the adjacency matrix $\mathbf{A}_{G'}$ of the new graph G',

Table 1: Explanation of notations.

Notations	Descriptions
n	Number of new fake nodes
K	The maximum number of neighbors of each new node
l	The maximum number of new edges that each existing node can have
T	The set of target nodes
r	Number of edges connecting target nodes for each new node
w	Weight of edges
S	The set of new nodes
G	The original graph
G'	The new graph
$\mathbf{A}$	The adjacency matrix of the original graph
$\mathbf{A}_{G'}$	The adjacency matrix of the new graph
$\mathbf{W}$	The matrix of weights of the original graph
$\mathbf{W}_{G'}$	The matrix of weights of the new graph
$\mathbf{w}_u$	The uth row of $\mathbf{W}_{G'}$
$\mathbf{C}$	The cost matrix
$\mathbf{c}_u$	The uth row of the cost matrix $\mathbf{C}$
$\mathbf{B}$	The adversarial matrix with relaxed continuous variables
B_{uv}	An entry of $\mathbf{B}$ that indicates whether insert an edge between node u and node v
$\bar{B}_{uv}$	Binarized B_{uv}
$\mathbf{b}_u$	The uth row of adversarial matrix $\mathbf{B}$
$\bar{\mathbf{b}}_u$	Binarized $\mathbf{b}_u$
q_u	The prior reputation score of node u
p_u	The posterior reputation score of node u
α, β	Lagrangian multipliers
$\mathbf{d}$	The column vector for Lagrangian multipliers
$*^{(t)}$	A value in the tth iteration

which is

$$\begin{aligned}\mathbf{A}_{G'} &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_{|V|\times|S|} \\ \mathbf{0}_{|S|\times|V|} & \mathbf{0}_{|S|\times|S|} \end{bmatrix} + \mathbf{B} \\ &= \begin{bmatrix} \mathbf{A} & \mathbf{B}_{SV}^T \\ \mathbf{B}_{SV} & \mathbf{B}^S \end{bmatrix}\end{aligned} \tag{4}$$

Thus, we can rewrite the matrix of all weights for the new graph G' in a similar way:

$$\mathbf{W}_{G'} = \begin{bmatrix} \mathbf{W} & \mathbf{W}_{SV}^T \\ \mathbf{W}_{SV} & \mathbf{W}^S \end{bmatrix} \tag{5}$$

where $\mathbf{W}_{SV}$ is the weight between S and V, and $\mathbf{W}^S$ are the weights among the new nodes in the new node set S.

4.2 Formulation

Using the notations above, the formulation of the optimization problem is given as follows:

$$\begin{aligned}\min_{\mathbf{B}} \quad & \sum_{u\in S;\ v\in V'} B_{uv}C_{uv} + n \times C_{node} \\ s.t. \quad & FNR = 1, \text{ for the new nodes} \\ & FNR = 1, \text{ for the target nodes} \\ & B_{uv} \in \{0,1\}, \text{ for } u \in S, v \in V' \\ & \textstyle\sum_v B_{uv} \le K, \text{ for } u \in S\end{aligned} \tag{6}$$

We aim to find the adversarial matrix $\mathbf{B}$ that minimizes the objective function in (6). The objective function is the total cost of creating new nodes and inserting edges. The first and the second constraints mean that LinLBP mis-classifies all the new nodes and target nodes as negative. The third constraint means that the variables are binary. The last constraint indicates that the maximum number of neighbors that each new node can have is no more than K.

Since the variables are binary and some constraints are highly nonlinear, it is very difficult to solve this optimization problem directly. Instead, we approximately solve this problem in the following.

For each binary variable B_{uv}, we relax it to a continuous variable, and convert it back to a binary after solving the related optimization problem. In the rest of paper, B_{uv} is a continuous variable unless stated otherwise.

For the first and the second constraints, which are highly nonlinear, we convert FNR = 1 to $p_u < 0$ for each u in the new node set and target node set, where p_u is the posterior reputation score of u. This is reasonable because LinLBP predicts a node as negative when its posterior reputation score is negative. Then we add converted constraints to the objective function using Lagrangian multipliers, and we rewrite the optimization problem as below:

$$\begin{aligned}\min_{\mathbf{B}} \quad & \sum_{u\in S;\ v\in V'} B_{uv}C_{uv} + n \times C_{node} + \alpha \sum_{u\in S} p_u + \beta \sum_{u\in T} p_u \\ s.t. \quad & \textstyle\sum_v \bar{B}_{uv} \le K, \text{ for } u \in S\end{aligned} \tag{7}$$

where $\alpha, \beta > 0$ are Lagrangian multipliers, and $\bar{B}_{uv}$ is a binary converted from the continuous variable B_{uv}.

Since the posterior reputation scores are iteratively computed in LinLBP, without loss of generality, here we consider the tth iteration, and the posterior reputation scores are computed as follows:

$$\mathbf{p}_{G'}^{(t)} = \mathbf{q}_{G'} + \mathbf{A}_{G'}^{(t-1)} \odot \mathbf{W}_{G'}\mathbf{p}_{G'}^{(t-1)} \tag{8}$$

where $\mathbf{q}_{G'}$ is the prior reputation scores of the new graph G'. $\mathbf{A}_{G'}^{(t-1)}$ and $\mathbf{p}_{G'}^{(t-1)}$ are the adjacency matrix and posterior reputation scores in the $(t-1)$th iteration, respectively.

After having posterior reputation scores in the tth iteration, we can update the adversarial matrix $\mathbf{B}^{(t)}$ by solving

$$\begin{aligned}\min_{\mathbf{B}^{(t)}} \quad & O(\mathbf{B}^{(t)}) = \sum_{u\in S} \mathbf{b}_u^{(t)}\mathbf{c}_u^T + n \times C_{node} + \mathbf{d}^T\mathbf{p}_{G'}^{(t+1)} \\ s.t. \quad & \textstyle\sum_v \bar{B}_{uv} \le K, \quad \text{for } u \in S\end{aligned} \tag{9}$$

The above optimization problem is expressed in vector form, where the objective function is $O(\mathbf{B})$, $\mathbf{b}_u^{(t)}$ is the uth row of the $\mathbf{B}^{(t)}$ corresponding to the new node u, $\mathbf{c}_u$ is the uth row of the cost matrix $\mathbf{C}$, $\mathbf{d}$ is a column vector for Lagrangian multipliers defined as:

$$d_u = \begin{cases} \alpha, & u \in S \\ \beta, & u \in T \\ 0, & \text{Otherwise} \end{cases} \tag{10}$$

Given a graph, a set of new nodes and a set of chosen target nodes, $\mathbf{d}$ is fixed.

We search the optimal value of $\mathbf{B}^{(t)}$ by using the gradient of each row $\mathbf{b}_u^{(t)}$, i.e., finding the optimal connections of each new node one by one. In each iteration, we first reset $\mathbf{b}_u^{(t)}$ to be $\mathbf{0}$ before we recompute $\mathbf{b}_u^{(t)}$ for the new node u. In other words, when we decide to re-choose the neighbors for the new node u, we treat the node u as a brand new node with no neighbor, which is also consistent with the attacker's capability — inserting new edges instead of modifying existing edges.

For each new node u, we compute the gradient as follows:

$$\begin{aligned}\frac{\partial O\left(\mathbf{B}^{(t)}\right)}{\partial \mathbf{b}_u^{(t)}} &= \mathbf{c}_u + \frac{\partial\left\{\mathbf{d}^T\left[\mathbf{B}^{(t)} \odot \mathbf{W}_{G'}\mathbf{p}_{G'}^{(t)}\right]\right\}}{\partial \mathbf{b}_u^{(t)}} \\ &= \mathbf{c}_u + \left[\mathbf{d}_u^T\left(p_u\mathbf{W}_u^d\right) + \alpha\left(\mathbf{w}_u\mathbf{P}_{G'}^{(t)}\right)\right]\end{aligned} \tag{11}$$

Each entry in $\mathbf{d}_u$ is $d_{u,i} = d_i$ for $i \neq u$, and $d_{u,u} = 0$. $\mathbf{w}_u$ is the uth row of $\mathbf{W}_{G'}$. $\mathbf{W}_u^d$ is a diagonal matrix, and the elements on the diagonal are the elements in $\mathbf{w}_u$, i.e., $\mathbf{W}_u^d(i,i) = \mathbf{w}_u(i)$. $\mathbf{P}_{G'}^{(t)}$ is a diagonal matrix and $\mathbf{P}_{G'}^{(t)}(i,i) = p_i^{(t)}$, which is the posterior reputation score of the ith node in the tth iteration.

Note that the gradient we obtained for the new node u is a constant, which means that we can keep going along the opposite direction of the gradient to minimize the objective function, without worrying about the change of the gradient when we have a different $\mathbf{b}_u^{(t)}$. Recall that $\mathbf{b}_u^{(t)}$ is initialized to be $\mathbf{0}$, so the recomputed connections for new node u is

$$\mathbf{b}_u^{(t)} = -\lambda \times \frac{\partial O\left(\mathbf{B}^{(t)}\right)}{\partial \mathbf{b}_u^{(t)}} \tag{12}$$

where $\lambda > 0$ is the step size. As one of the constraints, the maximum number of neighbors that each new node can have is bounded by

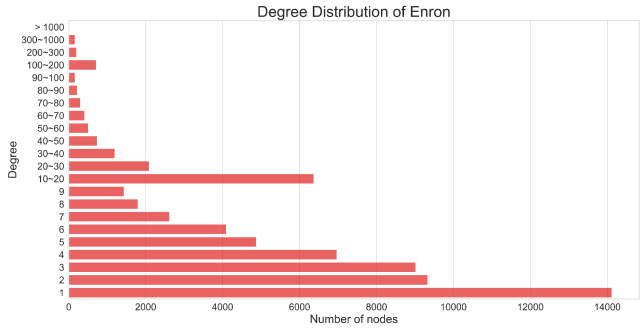


Figure 1: The degree distribution of the Enron dataset.

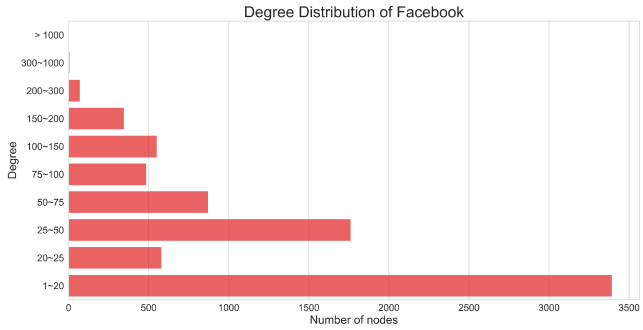


Figure 2: The degree distribution of the Facebook dataset.

K, which implies that at most K entries of $\bar{\mathbf{b}}_u^{(t)}$ can be 1. To this end, we can select the largest non-negative K entries in $\mathbf{b}_u^{(t)}$, and convert them to 1 and set other entries to 0, in order to obtain binary values $\bar{\mathbf{b}}_u^{(t)}$. $\bar{\mathbf{b}}_u^{(t)}$ is the uth row in the adversarial matrix in the tth iteration. The posterior reputation scores and the adversarial matrix are iteratively updated as aforementioned until the convergence or the predefined maximum number of iterations is reached.

If we want the new node u to connect to r target nodes, we only need to reserve the exact number of entries for r target nodes, which have the largest entries among target nodes.

5 EVALUATION

5.1 Experimental Setup

5.1.1 Dataset Description. In order to conduct meaningful evaluation of the new attack on different types of graphs, such as different graph sizes and density of nodes, we choose two real-world graphs: Enron and Facebook graphs. The two datasets containing only negative nodes are originally from SNAP [18]. We obtained the datasets with synthesized positive nodes from one of authors of [34] and the website of another author [11]. The authors of [34] cited the previous works [1, 12, 38], and they synthesized positive nodes and their edges by replicating the negative nodes and their edges without incurring the influence of structural differences between negative and positive nodes.

The Enron graph with synthesized positive nodes has 67,392 nodes, where half of them are negative nodes and the other half are positive nodes. Each node in the Enron graph stands for an email address, and an edge between two nodes means that these two email addresses sent at least one email to each other. A negative node in the Enron graph represents an normal email address, while a positive node represents a spamming email address.

The Facebook graph with synthesized positive nodes has 8,078 nodes, where positive nodes and negative nodes each share half of them. Each node in the Facebook graph represents a user, and two users are connected by an edge if they are friends on Facebook. A negative node stands for a normal user and a positive node means that this user is malicious.

5.1.2 Node Degree Distribution. Figure 1 shows the node degree distribution of the Enron dataset. The average degree of the Enron dataset is 11. Most nodes have degrees less than 10, but a few nodes have vast degrees, some with more than 1000 degrees.

Figure 2 shows the degree distribution of the Facebook dataset. The average degree of the Facebook dataset is 44. Most nodes have degrees less than 50, but some nodes have degrees larger than 1000.

Note that the average degrees are both larger than 10 and the reasonable weight w should be around $1/degree$. Thus, in the experiments, we consider $w \in (0, 0.1]$.

5.1.3 Training Dataset and Testing Dataset. Each training dataset contains 100 positive nodes and 100 negative nodes, all of which are randomly selected. We set $\theta = 0.4$ in LinLBP. To be more specific, the prior reputation scores of the 100 positive nodes are set as 0.4, which means highly positive, and the prior reputation scores of the 100 negative nodes are set as -0.4, which means highly negative. All other nodes are the testing dataset, and their prior reputation scores are set as 0.

5.1.4 Target Nodes Selection. An attacker's goal is to choose some target nodes that can be utilized to help new and target nodes evade detection. The target nodes can be a subset of the positive nodes. Without loss of generality, we consider that the attacker randomly picks 100 target nodes from positive nodes as target nodes.

5.1.5 Edge Cost. Assume the costs of inserting new edges for different node pairs are uniformly distributed among [1, 10]. Each time, a number is generated between [1, 10] to represent the cost of inserting a new edge between a new node and an existing node. Note that we do not need to consider the cost between two existing nodes because the attack does not modify any existing edges in the original graph. Note that the values of cost here are only to simulate different difficulties of inserting edges.

5.1.6 Parameters Setting. We set $w = 0.01$, $l = 20$, and $\alpha = \beta = 1000$. For the prior reputation scores of nodes in the training dataset, as mentioned above, we set $\theta = 0.4$. The nodes in the training dataset are clearly labeled, and each of them is either highly positive or negative. LinLBP knows nothing about the newly added fake nodes, because they are new to LinLBP. However, LinLBP may consider the nodes as suspicious if there are improper activities from the attacker. For instance, an attacker creates too many fake accounts using the same IP and/or in a very short period of time. Taking these factors into consideration, LinLBP labels the new fake nodes as slightly positive at the beginning, e.g., sets the prior reputation scores of the new nodes to be 0.1.

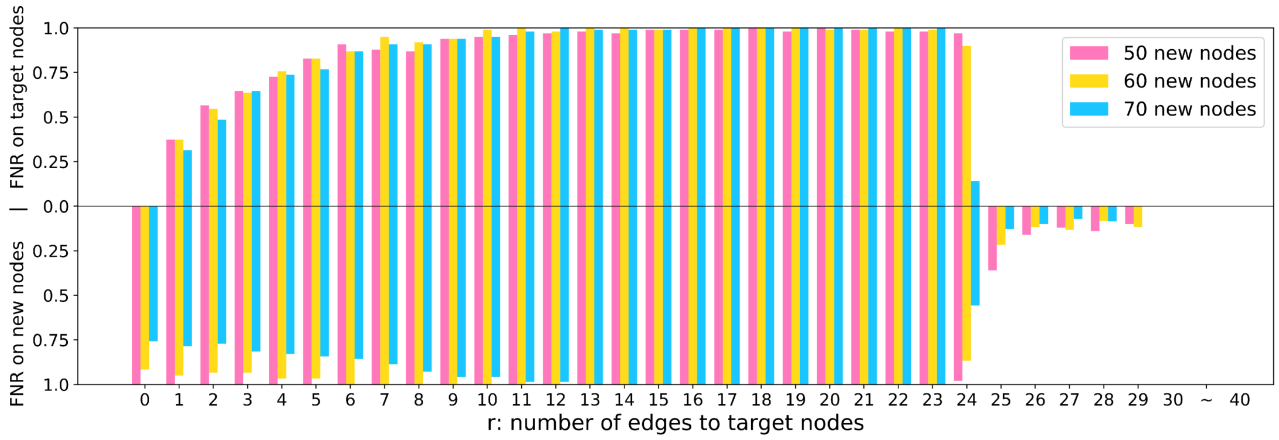


Figure 3: The impact of different values of r on FNRs for the Enron dataset.

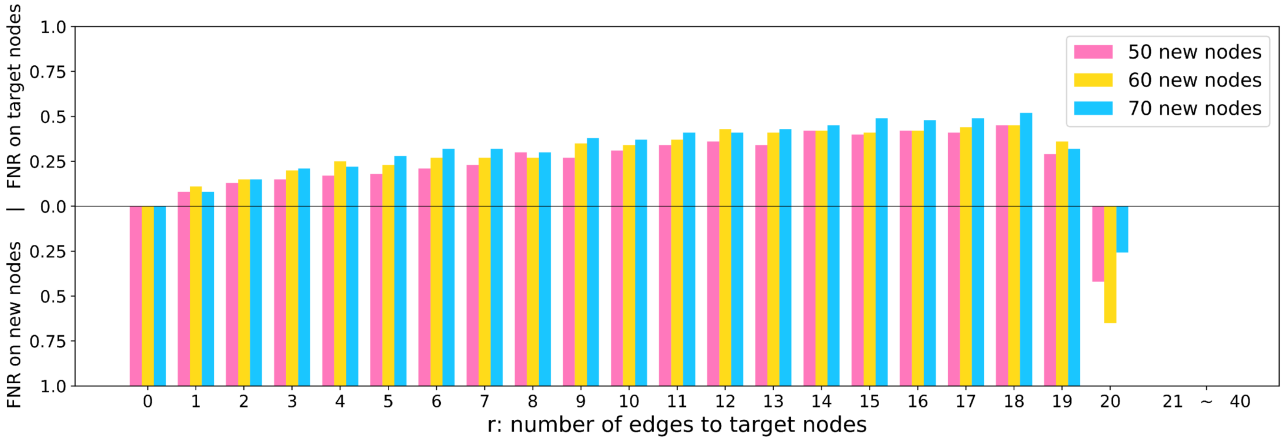


Figure 4: The impact of different values of r on FNRs for the Facebook dataset.

5.2 Experimental Results

5.2.1 *Attacker with Full Knowledge.*

Choice of the Best r. The number of edges between new nodes and target nodes could have an impact on the FNR for the target nodes. We study the impact for different numbers of new nodes on both datasets. Without loss of generality, we set $K = 40$ for both datasets, and we want to find the best r for each of them. Note that each new node has r edges connected to the target nodes. Meanwhile, the remaining $K - r$ edges are linked to other nodes, which are very likely to be negative (good) nodes.

For the Enron dataset, as shown in Figure 3, the FNR reaches the highest value when the number of edges to the target nodes is about the half of K - the maximum number of edges each new node can have. Thus, we set $r = K/2$ for the Enron dataset.

For the Facebook dataset, as shown in Figure 4, the FNR reaches the highest value when the number of edges to the target nodes is about 40% of K. Thus, we set $r = 0.4K$ for the Facebook dataset.

Note that in both figures, when r is relatively small, though new nodes can be highly negative due to lots of connections to existing negative nodes, the target nodes are barely influenced by the new nodes via only a few connections. When r is relatively large, e.g., $r \geq 30$ in Figure 3 and $r \geq 21$ in Figure 4, the new nodes themselves cannot become negative because they only have a small number of edges to existing negative nodes. As a result, the target nodes are still positive and the FNR for the target nodes is 0.

Influence of K ***and*** n. We study the influence of different value of K and n on FNRs, which are shown in Figure 5 and Figure 6. We can see that with the same number of new nodes, a larger value of K usually has a higher FNR, which implies that the more neighbors each new node can have, then the more powerful the attack is.

Specifically, in Figure 5, the FNRs for the new nodes and the target nodes remain 100% for all the different combinations of values of K and n. When K is too small, FNRs on both S and T are almost 0. This is because new nodes don't have enough edges connecting to existing negative nodes to make themselves classified as negative, which also makes them useless to target nodes. However, as K grows, FNRs can achieve 1 on both sets. Recall that the average degree of the Enron dataset is 11, and we try to find a proper K, which is suitable for the attack and also as close to the average

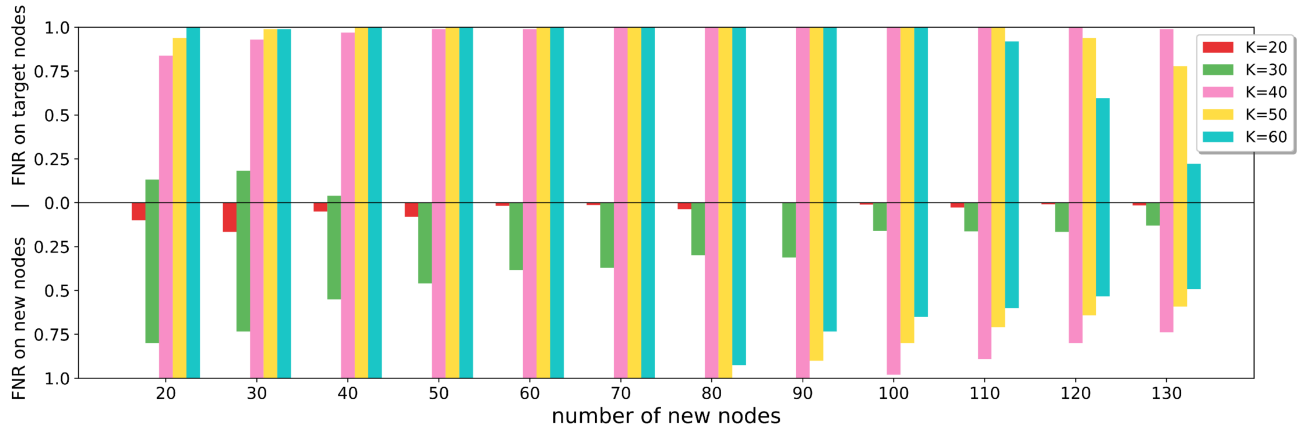


Figure 5: The impact of different values of K and n on FNRs for the Enron dataset.

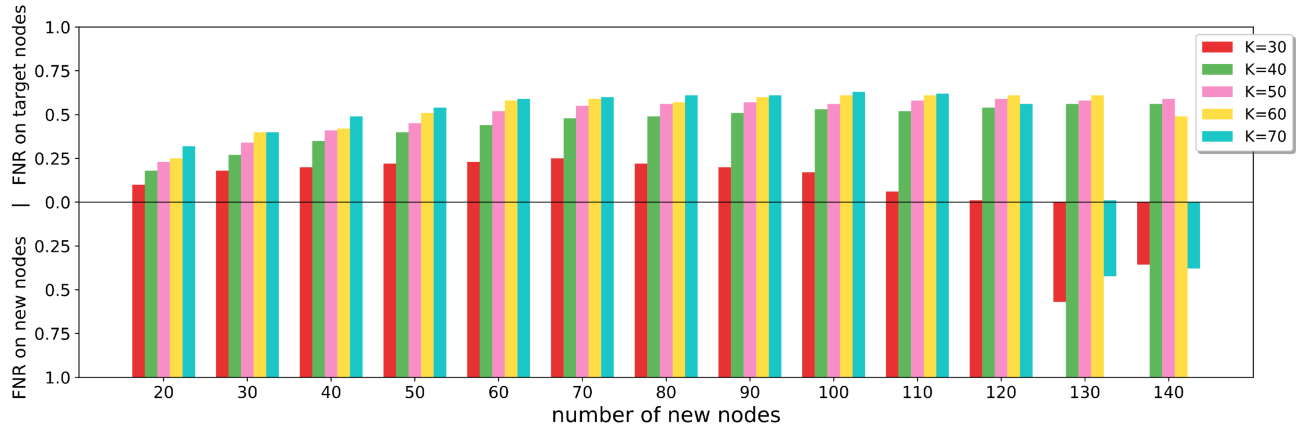


Figure 6: The impact of different values of K and n on FNRs for the Facebook dataset.

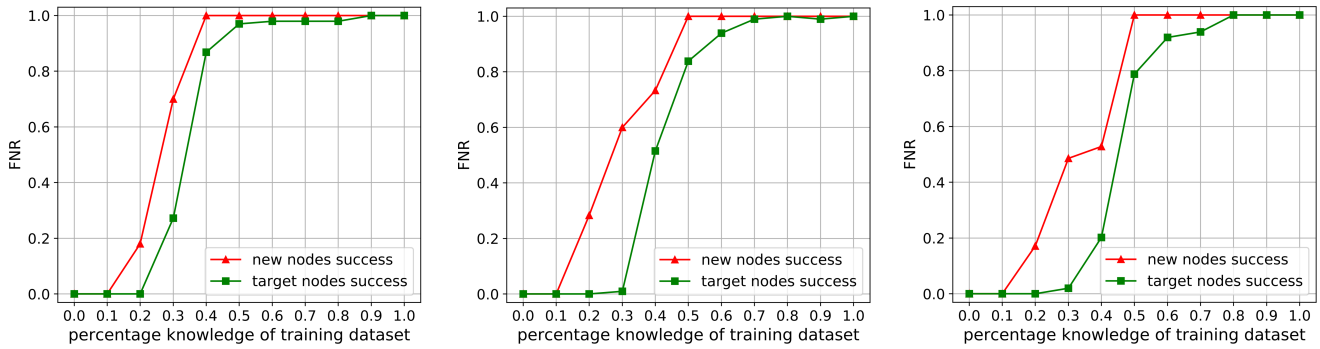
degree as possible. Since $K = 20, 30$ is not enough to launch a successful attack, for consistency, we set $K = 40$ for the Enron dataset.

Figure 6 shows that for the Facebook dataset, the highest FNR for the target nodes is around 0.5, which means that only half of the target nodes can be mis-classified. Recall that the average degree of the Facebook dataset is 44, and it is much denser than the Enron dataset. Besides, an attacker does not modify existing connections in the original graph, which implies that it is much more difficult for an attacker to affect the target nodes under the strong influence from the original neighbors of the target nodes. With similar consideration of K of the Enron dataset, we set $K = 50$ for the Facebook dataset.

Interestingly, the FNR decreases dramatically as the number of new nodes increases, especially when K is either relatively small or large. For instance, in Figure 6, when $K = 30$ or $K = 70$, the FNR for the target nodes gradually drops as the number of new nodes n grows, and even becomes 0 when $n \geq 130$. The following two factors cause the phenomenon shown in Figure 5 and Figure 6.

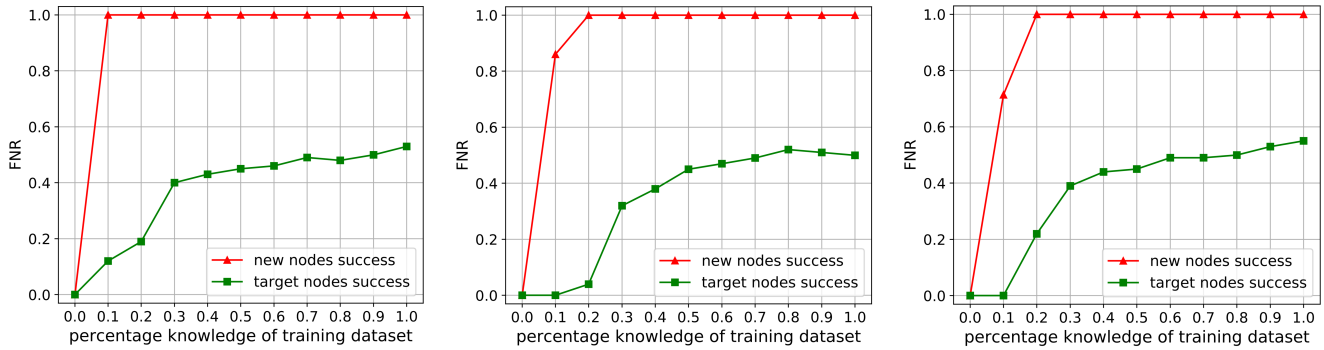
- New nodes choose their potential neighbors according to their preference lists. With the fixed K and the constraint l, the most popular existing nodes are gradually occupied (connected) by the new nodes. As the number of new nodes grows, since top candidate nodes in the preference list of later new nodes may be already fully occupied and become no longer available, the later new nodes are more likely to connect to some "bad" neighbors, which finally leads to a decreasing FNR.
- Smaller K means that each new node is allowed to connect fewer existing nodes. Though new nodes can still be classified as negative, they are closer to the classification boundary (i.e., the posterior reputation score equals to 0) than when K is larger, due to the limited help from existing negative nodes, which will further result in less stability against the "bad" neighbors mentioned above. In other words, FNRs for new nodes will drop faster when the number of connected "bad" neighbors increases.

Figure 5 and Figure 6 have similar trends. Here, we take Figure 6 as an example to explain the results in detail. Comparing $K = 30$ and $K = 40$, since K is still relatively small, the popular existing nodes are occupied slower, which means that the first factor does not make a big influence when K is small. Thus, the second factor dominates



(a) $n = 50$ (b) $n = 60$ (c) $n = 70$

Figure 7: The impact of varying knowledge of the training dataset, with different numbers of new nodes, on FNRs for the Enron dataset.



(a) $n = 50$ (b) $n = 60$ (c) $n = 70$

Figure 8: The impact of varying knowledge of the training dataset, with different numbers of new nodes, on FNRs for the Facebook dataset.

here, explaining why the FNR for the new nodes decreases when $n \geq 130$ for $K = 30$. On the other hand, comparing $K = 60$ and $K = 70$, since K is large here, the influence of the second factor becomes negligible. Nevertheless, the popular existing nodes are occupied much faster, and the new nodes connect to more "bad" neighbors. Thus, the first factor dominates in this scenario, explaining why FNRs for the new nodes are quite different for $K = 60$ and $K = 70$ when $n \geq 130$.

5.2.2 *Attacker with Partial Knowledge.*

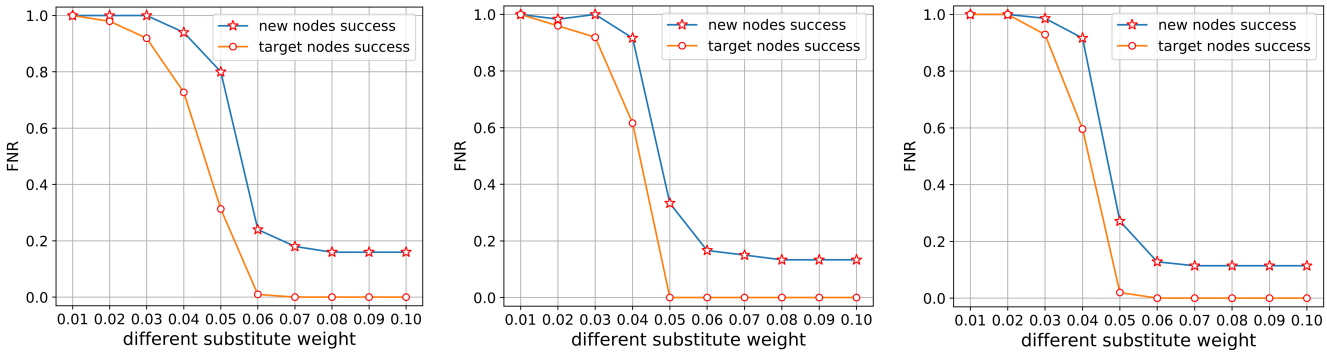
Partial Knowledge of Training Dataset. Training dataset is the basic known knowledge of a graph and it acts as a starting point for LinLBP to propagate beliefs and then calculate the posterior reputation scores for all nodes. In some cases, the entire training dataset may not be fully available to the attacker, and the attacker only knows a part of it. Figures 7 and 8 show how an attacker's knowledge of the training dataset affects FNRs.

As we can see in Figure 7 and Figure 8, an attacker can still launch a powerful attack while only knowing a part of the training dataset. It is worth noting that an attacker can always achieve 100% FNRs for new nodes with only partial knowledge of the training dataset for both Enron and Facebook datasets. As the number of new nodes increases, more of the training dataset needs to be known to launch a successful attack. Specifically, 40% of the training dataset is sufficient if an attacker only wants to launch a small scale attack, e.g., only adding 50 new fake nodes to the graph ($n = 50$).

Attack with Substitute Weight. Under this situation, the attacker knows the entire training dataset but does not have any idea of the weight w in LinLBP. The attacker may use a substitute weight and still achieve a high FNR.

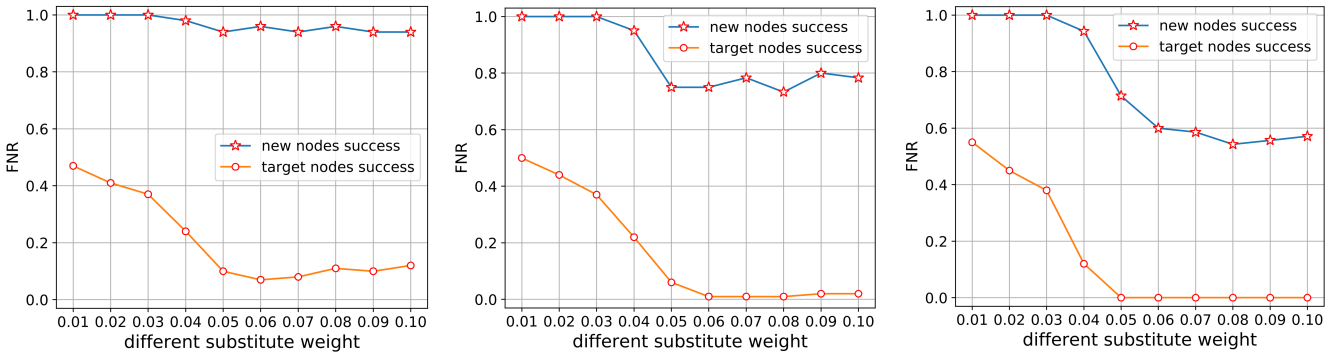
Figure 9 and Figure 10 show that the attack performance decreases as the substitute weight is further from the true weight. However, the attacker still has about a 50% chance to launch an effective attack even if the attacker randomly picks a value from the weight range.

Besides, in Figure 9 and Figure 10, with a fixed substitute weight, FNR drops as the number of new nodes increases. This is because an attacker using a substitute weight to propagate reputation scores



(a) $n = 50$ (b) $n = 60$ (c) $n = 70$

Figure 9: The impact of varying substitute weight values, with different numbers of new nodes, on FNRs for the Enron dataset.



(a) $n = 50$ (b) $n = 60$ (c) $n = 70$

Figure 10: The impact of varying substitute weight values, with different numbers of new nodes, on FNRs for the Facebook dataset.

among a larger number of new nodes is more likely to obtain inaccurate posterior reputation scores. This implies that if an attacker has no idea about the weight, the attacker can launch a more effective attack by adding a smaller number of new nodes.

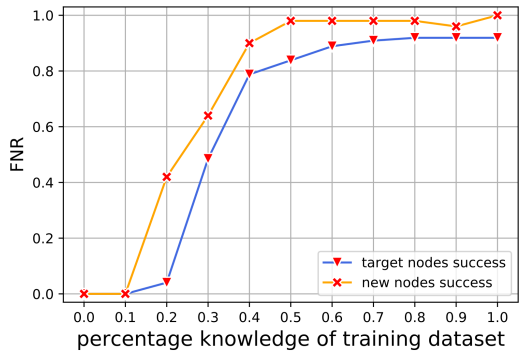
Attack with Substitute Weight and Partial Training Dataset. In this scenario, an attacker has even less knowledge of LinLBP, neither the weight w nor the full knowledge of the training dataset. In the experiments, we fix one of the two factors (substitute weight and the knowledge of training dataset) to explore FNRs under the different values of the other factor. By default, we assume that an attacker uses a substitute weight 0.03 and knows 40% of the training dataset. Since the attacker only has little knowledge of LinLBP, as mentioned before, the attacker may prefer to perform a small scale attack by choosing $n = 50$, i.e., only adding 50 new fake nodes. We also run the experiments for larger scale attacks, such as $n = 60, 70, 80$, and 90. When n increases, the trends of the FNR are similar to those shown in Figures 7, 8, 9, and 10.

Figure 11 reveals the attack performance on the Enron dataset and the Facebook dataset under the least knowledge of LinLBP. Specifically, Figure 11(a) and Figure 11(c) show that with a substitute weight 0.03, the attacker is still able to successfully launch attacks with partial training knowledge, and only suffers a slight decrease of the FNR for the target nodes, comparing to the performance with the knowledge of the true weight. Also, as we can see from Figure 11(b) and Figure 11(d), with only 40% of the training dataset, an attacker can choose a relatively small substitute weight to make LinLBP misclassify almost all the new fake nodes and a part of the target nodes at the same time. Note that the FNR on the target node set for the Facebook dataset is lower than that for the Enron dataset. The reason is that the Facebook dataset is much denser than the Enron dataset, which is similar to the reason of the difference between Figure 5 and Figure 6.

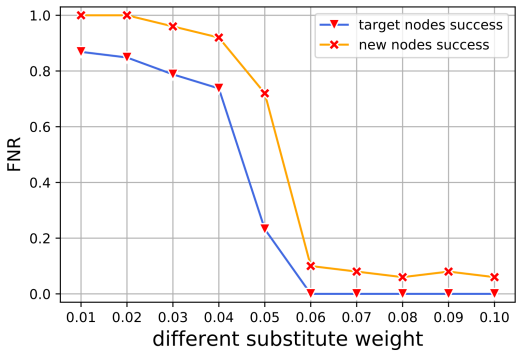
5.3 Evaluation Summary

According to the experimental results shown above, different values of parameters play an essential role in the effectiveness of attacks. For different datasets, the choice of the best value for each parameter varies. However, the trends are similar as the parameters vary.

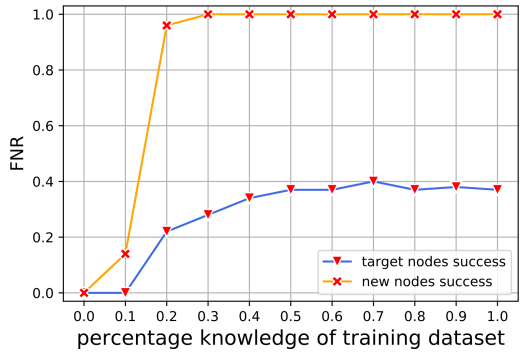
For each new node, a balance between values of r and $K - r$ is necessary for an attacker to achieve high FNRs on both the new



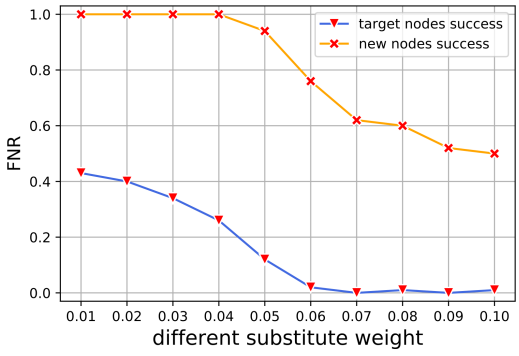
(a) Impact of knowledge of the training data on FNRs for Enron



(b) Impact of substitute weight on FNRs for Enron



(c) Impact of knowledge of the training data on FNRs for Facebook



(d) Impact of substitute weight on FNRs for Facebook

Figure 11: The impact under the least knowledge on FNRs for the Enron dataset and the Facebook dataset. When exploring the impact of knowledge of the training data, a substitute weight of 0.03 is used. When exploring the impact of the substitute weight, 40% of the training dataset is used.

node set and the target node set. An extremely small or large value of r can lead to a poor FNR.

When the number of new nodes n is relatively small, a larger value of K can make the attack more powerful. On the contrary, a larger value of K can harm the effectiveness of the attack when n has a relatively large value. Thus, it is critical for an attacker to find an appropriate combination of values of K and n to launch a successful attack. For the Enron dataset, for example, we achieve 100% FNRs for the new nodes and the target nodes under various combinations of values of K and n, such as $K = 40$, $n = 90$ and $K = 50$, $n = 70$.

With partial knowledge of the training dataset and a substitute weight, adding a smaller number of new nodes can make the attack more effective, which means the attacker can achieve a higher FNR and needs less training dataset. Besides, the FNR increases when the attacker either has more knowledge about the training dataset, or uses a substitute weight closer to the true weight. For instance, with a substitute weight = 0.03 and only 40% of the training dataset, the attack still can achieve FNR = 0.8 for the target nodes of the Enron dataset for $K = 40$ and $n = 50$.

6 CONCLUSION

In this paper, we proposed a new attack on collective classification methods for graphs by adding fake nodes into existing graphs. The new attack is more realistic and practical than the attacks in several literature. We formulated the new attack as an optimization problem and utilized a gradient-based method to generate edges of newly added fake nodes. Our extensive experiments on real-world social network data show that the attack can not only make new fake nodes evade detection, but also make the detector mis-classify most of the chosen target (bad) nodes. The new attack works well even when the attacker has only partial knowledge about the training dataset and/or the detector's model.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1828363, CNS-1564128, CNS-1824440, CNS-2016589, CNS-1856380 and CNS-2016415. We would like to thank the authors of [34], Binghui Wang and Neil Zhenqiang Gong, for sharing their source code and datasets. We also thank anonymous reviewers for their valuable feedback.

REFERENCES

[1] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. 2013. Sok: The evolution of sybil defense via social networks. In *2013 ieee symposium on security and privacy*. IEEE, 382–396.

[2] Aleksandar Bojchevski and Stephan Günnemann. 2018. Adversarial attacks on node embeddings via graph poisoning. *arXiv preprint arXiv:1809.01093* (2018).

[3] Yazan Boshmaf, Dionysios Logothetis, Georgos Siganos, Jorge Lería, Jose Lorenzo, Matei Ripeanu, and Konstantin Beznosov. 2015. Integro: Leveraging Victim Prediction for Robust Fake Account Detection in OSNs.. In *NDSS*, Vol. 15. 8–11.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[5] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. 2012. Aiding the detection of fake accounts in large scale social online services. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 197–210.

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[7] Duen Horng "Polo" Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 131–142.

[8] Yizheng Chen, Yacin Nadji, Athanasios Kountouras, Fabian Monrose, Roberto Perdisci, Manos Antonakakis, and Nikolaos Vasiloglou. 2017. Practical attacks against graph-based clustering. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1125–1142.

[9] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2018. Adversarial network embedding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[10] Peng Gao, Binghui Wang, Neil Zhenqiang Gong, Sanjeev R Kulkarni, Kurt Thomas, and Prateek Mittal. 2018. Sybilfuse: Combining local attributes with global structure to perform robust sybil detection. In *2018 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 1–9.

[11] Neil Zhenqiang Gong. 2019. Code & Data. http://gonglab.pratt.duke.edu/code-data.

[12] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. 2014. Sybilbelief: A semi-supervised learning approach for structure-based sybil detection. *IEEE Transactions on Information Forensics and Security* 9, 6 (2014), 976–987.

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[14] Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. 2017. Random walk based fake account detection in online social networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 273–284.

[15] Jinyuan Jia, Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. 2017. AttriInfer: Inferring user attributes in online social networks using markov random fields. In *Proceedings of the 26th International Conference on World Wide Web*. 1561–1569.

[16] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[18] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[19] Huayi Li, Zhiyuan Chen, Bing Liu, Xiaokai Wei, and Jidong Shao. 2014. Spotting fake reviews via collective positive-unlabeled learning. In *2014 IEEE international conference on data mining*. IEEE, 899–904.

[20] Zhou Li, Sumayah Alrwais, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2013. Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 112–126.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[22] Abedelaziz Mohaisen, Nicholas Hopper, and Yongdae Kim. 2011. Keep your friends close: Incorporating trust into social network-based sybil defenses. In *2011 Proceedings IEEE INFOCOM*. IEEE, 1943–1951.

[23] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*. 201–210.

[24] Hogun Park and Jennifer Neville. 2019. Exploiting interaction links for node classification with deep graph neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 3223–3230.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[26] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[27] Meng Shen, Zelin Liao, Liehuang Zhu, Ke Xu, and Xiaojiang Du. 2019. VLA: A Practical Visible Light-based Attack on Face Recognition Systems in Physical World. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 3 (2019), 1–19.

[28] Gianluca Stringhini, Yun Shen, Yufei Han, and Xiangliang Zhang. 2017. Marmite: spreading malicious file reputation through download graphs. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. 91–102.

[29] Mingjie Sun, Jian Tang, Huichen Li, Bo Li, Chaowei Xiao, Yao Chen, and Dawn Song. 2018. Data poisoning attack against unsupervised node embedding methods. *arXiv preprint arXiv:1810.12881* (2018).

[30] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Non-target-specific Node Injection Attacks on Graph Neural Networks: A Hierarchical Reinforcement Learning Approach. In *Proc. WWW*, Vol. 3.

[31] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1524–1533.

[32] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[33] MohamadAli Torkamani and Daniel Lowd. 2013. Convex adversarial collective classification. In *International Conference on Machine Learning*. 642–650.

[34] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2023–2040.

[35] Binghui Wang, Neil Zhenqiang Gong, and Hao Fu. 2017. GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 465–474.

[36] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. 2018. Graph-based security and privacy analytics via collective classification with joint weight learning and propagation. *arXiv preprint arXiv:1812.01661* (2018).

[37] Binghui Wang, Jinyuan Jia, Le Zhang, and Neil Zhenqiang Gong. 2018. Structure-based sybil detection in social networks via local rule-based propagation. *IEEE Transactions on Network Science and Engineering* 6, 3 (2018), 523–537.

[38] Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. 2017. SybilSCAR: Sybil detection in online social networks via local rule based propagation. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[39] Xiaoyun Wang, Joe Eaton, Cho-Jui Hsieh, and Felix Wu. 2018. Attack graph convolutional networks by adding fake nodes. *arXiv preprint arXiv:1810.10751* (2018).

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[41] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. 2012. Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter. In *Proceedings of the 21st international conference on World Wide Web*. 71–80.

[42] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[43] Qiang Zeng, Jianhai Su, Chenglong Fu, Golam Kayas, Lannan Luo, Xiaojiang Du, Chiu C Tan, and Jie Wu. 2019. A multiversion programming inspired approach to detecting audio adversarial examples. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 39–51.

[44] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2847–2856.

[45] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. *arXiv preprint arXiv:1902.08412* (2019).

[46] Fei Zuo, Bokai Yang, Xiaopeng Li, and Qiang Zeng. 2019. Exploiting the inherent limitation of l0 adversarial examples. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*. 293–307.

[47] Fei Zuo and Qiang Zeng. 2020. Erase and Restore: Simple, Accurate and Resilient Detection of L_2 Adversarial Examples. *arXiv preprint arXiv:2001.00116* (2020).