Adversarial Attacks on Deep Graph Matching

Zijie Zhang, Auburn University
Zeru Zhang, Auburn University
Yang Zhou, Auburn University
Yelong Shen, Microsoft Dynamics 365 AI
Ruoming Jin, Kent State University
Dejing Dou, University of Oregon, Baidu Research
Adversarial Attacks on Deep Graph Matching

Zijie Zhang  
Auburn University  
zzz0092@auburn.edu

Zeru Zhang  
Auburn University  
zzz0054@auburn.edu

Yang Zhou  
Auburn University  
yangzhou@auburn.edu

Yelong Shen  
Microsoft Dynamics 365 AI  
yeshe@microsoft.com

Ruoming Jin  
Kent State University  
rjin1@kent.edu

Dejing Dou  
University of Oregon, Baidu Research  
dou@cs.uoregon.edu, doudejing@baidu.com

Abstract

Despite achieving remarkable performance, deep graph learning models, such as node classification and network embedding, suffer from harassment caused by small adversarial perturbations. However, the vulnerability analysis of graph matching under adversarial attacks has not been fully investigated yet. This paper proposes an adversarial attack model with two novel attack techniques to perturb the graph structure and degrade the quality of deep graph matching: (1) a kernel density estimation approach is utilized to estimate and maximize node densities to derive imperceptible perturbations, by pushing attacked nodes to dense regions in two graphs, such that they are indistinguishable from many neighbors; and (2) a meta learning-based projected gradient descent method is developed to well choose attack starting points and to improve the search performance for producing effective perturbations. We evaluate the effectiveness of the attack model on real datasets and validate that the attacks can be transferable to other graph learning models.

1 Introduction

Graph matching is one of the most important research topics in the graph domain, which aims to match the same entities (i.e., nodes) across two or more graphs [91, 98, 43, 46, 48, 72, 54, 105, 13, 75]. It has been widely applied to many real-world applications ranging from protein network matching in bioinformatics [33, 63], user account linking in different social networks [62, 51, 100, 37, 101, 21, 38], and knowledge translation in multilingual knowledge bases [87, 124], to geometric keypoint matching in computer vision [22]. Existing research efforts on graph matching can be classified into three broad categories: (1) structure-based techniques, which rely only upon the topological information to match two or multiple input graphs [43, 49, 98, 46, 54, 105, 13, 96, 84, 40, 67, 57, 88, 24]; (2) attribute-based approaches, which utilize highly discriminative structure and/or attribute features for ensuring the matching effectiveness [23, 94, 51, 10, 65, 16, 88, 100, 28, 39, 13, 97, 50, 52, 22]; and (3) heterogeneous methods, which employ heterogeneous structural, content, spatial, and temporal features to further improve the matching performance [92, 34, 44, 98, 83, 99, 77, 59, 102, 103, 21].

Recent literature has shown that both traditional and deep graph learning algorithms remain highly sensitive to adversarial attacks, i.e., carefully designed small perturbations in graph structure and attributes can cause the models to produce wrong prediction results [14, 126, 64, 125, 123, 69, 90, 74, 45, 85, 80, 127]. We have witnessed various effective attack models to cause failures of
A large number of research advances in adversarial attacks on graph data utilize iterative gradient-based methods to produce effective adversarial perturbations that fool a graph learning model. However, a recent study reports that the iterative gradient-based methods, such as Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD), often lead to invalid attacks since the iterative trajectories have difficulties crossing decision boundary of target learning model with small perturbation. Can we find a shortcut across the decision boundary to derive more effective attacks by beginning from good attack starting points in the graph matching?

A meta learning-based projected gradient descent (MLPGD) model to quickly adapt to a variety of target learning model with small perturbation. Can we find a shortcut across the decision boundary to derive more effective attacks by beginning from good attack starting points in the graph matching?

Traditionally, graph matching techniques are based on the assumption of feature consistency across graphs: Two nodes in different graphs are more likely to be found to be matching if they have similar topological and/or attribute features in respective graphs. These methods compute the similarity (or distance) scores between pairwise nodes across graphs and choose the node pairs with largest similarity (or smallest distance) as matching results. Intuitively, if an attacker perturbs a node by throwing it into a dense region in the graph with many similar nodes, it is hard for humans or defender programs to recognize it from the node pile. In addition, if two matched nodes are simultaneously moved to such dense regions in respective graphs, then this dramatically increases the difficulty in matching them correctly among many similar candidate nodes.

To our best knowledge, this work is the first to study adversarial attacks on graph matching. We propose to utilize kernel density estimation (KDE) technique to estimate the probability density function of nodes in two graphs, to understand the intrinsic distribution of graphs. By maximizing the estimated densities of nodes to be attacked, we push them to dense regions in respective graphs to generate adversarial nodes that are indistinguishable from many neighbors in dense regions. This increases the chance of producing wrong matching results as well as reduces the risk of perturbations being detected by humans or by defender programs. Our analysis is the first to introduce the KDE technique to conduct imperceptible attacks on graph data.

Searching for good attack starting points in large graphs is computationally inefficient. We develop a meta learning-based projected gradient descent (MLPGD) model to quickly adapt to a variety of new search tasks on multiple batches of target nodes for deriving effective attacks. However, the MLPGD model is non-smooth and non-differential, as the perturbation is a multi-step process and the projection at each step is non-differential. A Gaussian smoothing method is designed to approximate a smoothed model, and a Monte Carlo REINFORCE method is used to estimate the model gradient.

Empirical evaluation on real datasets demonstrates the superior performance of the GMA model against several state-of-the-art adversarial attack methods on graph data. Moreover, we validate that the attack strategies are transferable to other popular graph learning models in Appendix A.2.

### 2 Problem Definition

Given two graphs $G^1$ and $G^2$ to be matched, each is denoted as $G^s = (V^s, E^s)$ ($s = 1$ or $2$), where $V^s = \{v^s_1, \ldots, v^s_{N^s}\}$ is the set of $N^s$ nodes and $E^s = \{(v^s_i, v^s_j) : 1 \leq i, j \leq N^s\}$ is the set of edges. Each $G^s$ has an $N^s \times N^s$ binary adjacency matrix $A^s$, where each entry $A^s_{ij} = 1$ if there exists an edge $(v^s_i, v^s_j) \in E^s$; otherwise $A^s_{ij} = 0$. $A^s_i$ specifies the $i$th row vector of $A^s$. In this paper, if there are no specific descriptions, we use $v^s_i$ to denote a node $v^s_i$ itself and its representation $A^s_i$, i.e., $v^s_i = A^s_i$ and we utilize $v^s_{ij}$ to specify the $j$th dimension of $v^s_i$, i.e., $v^s_{ij} = A^s_{ij}$.
The dataset is divided into two disjoint sets \( D' \) and \( D \). The former denotes a set of known matched node pairs \( D' = \{(v^1_i, v^2_k) | v^1_i \leftrightarrow v^2_k, v^1_i \in V^1, v^2_k \in V^2\} \), where \( v^1_i \leftrightarrow v^2_k \) indicates that two nodes \( v^1_i \) and \( v^2_k \) belong to the same entity. The latter, denoted by \( D = \{(v^1_i, v^2_k) | v^1_i \leftrightarrow v^2_k, v^1_i \in V^1, v^2_k \in V^2\} \), is used to evaluate the graph matching performance, where the nodes (but not their matchings) are also observed during training. The goal of graph matching is to utilize \( D' \) as the training data to identify the one-to-one matching relationships between nodes \( v^1_i \) and \( v^2_k \) in the test data \( D \). By following the same idea in existing efforts [101, 88, 39, 38], this paper aims to minimize the distances between projected source nodes \( M(v^1_i) \in D' \) and target ones \( v^2_k \in D' \). The node pairs \( (v^1_i, v^2_k) \in D \) with the smallest distances are selected as the matching results.

\[
\min L \text{ where } L = \mathbb{E}_{(v^1_i, v^2_k) \in D'} \|M(v^1_i) - v^2_k\|^2
\]

where \( M \) denotes an injective one-to-one matching function \( M : v^1_i \in V^1 \mapsto v^2_k \in V^2 \).

The adversarial attack problem is defined as maximally degrading the matching performance of \( M \) on the test data \( D \) by injecting edge perturbations (including edge insertion and deletion) into \( G^s = (V^s, E^s) \) (\( s = 1 \) or \( 2 \)), leading to two adversarial graphs \( G^s \neq (V^s, E^s) \). We assume the attacker has limited capability, so that he/she can only make small perturbations.

## 3 Imperceptible Attacks with Node Density Estimation and Maximization

![Image](image.png)

Figure 1: Imperceptible Attacks

Intuitively, in Eq. (1), if there exist nodes \( v^1_i \) similar to \( v^1_j \), i.e., \( v^1_j \approx v^1_i \), such that \( \|M(v^1_j) - v^2_k\|_2 < \|M(v^1_i) - v^2_k\|_2 \), then a wrong matching \( (v^1_j, v^2_k) \) will be generated. In addition, if there are many such \( v^1_j \)'s around \( v^1_i \), then it is hard to recognize \( v^1_i \) from a pile of similar nodes. Thus, if we move \( v^1_i \) to dense regions that contain many similar \( v^1_j \)'s, then this dramatically increases the possibility of deriving the wrong matching \( (v^1_j, v^2_k) \) among many similar candidate nodes. Also, as many \( v^1_j \)'s are around the adversarial node \( v^1_i \), it is difficult for humans or defender programs to detect \( v^1_i \), as shown in a toy example in Figure 1.

Motivated by this, we propose to employ kernel density estimation (KDE) method to generate imperceptible perturbations. In statistics, the KDE is to estimate the probability density function \( f(x) \) of a random variable \( x \) with unknown distribution [55]. It helps reveal the intrinsic distribution. Concretely, let \( v^1 \) be a \( N^1 \)-dimensional random variable to denote all nodes \( \{v^1_1, \cdots, v^1_{N^1}\} \) in graph \( G^1 \) with an unknown density \( f \). A function \( \hat{f}(v^1) \) is estimated to best approximate \( f(v^1) \).

\[
\hat{f}(v^1) = \frac{1}{N^1 \text{det}(B)} \sum_{i=1}^{N^1} K(B^{-1}(v^1 - v^1_i))
\]

where \( \text{det}(\cdot) \) denotes the determinant operation. \( B > 0 \) is a bandwidth to be estimated. It is an \( N^1 \times N^1 \) diagonal matrix \( B = \text{diag}(b_1, \cdots, b_{N^1}) \), which has strong influence on the density estimation \( \hat{f}(v^1) \). A good \( B \) should be as small as the data can allow. \( K \) is a product symmetric kernel that satisfies \( \int K(x) dx = 1 \) and \( \int xK(x) dx = 0 \). The above vector-wise form \( \hat{f}(v^1) \) can be rewritten as an element-wise form, where \( v^1_j \) represents the \( j^\text{th} \) dimension in \( v^1 \).

\[
\hat{f}(v^1) = \frac{1}{N^1} \prod_{j=1}^{N^1} \frac{1}{b_j} K\left(\frac{v^1_j - v^1_i}{b_j}\right)
\]

The derivative \( \frac{\partial \hat{f}(v^1)}{\partial b_j} \) w.r.t. each bandwidth \( b_j \) in \( B \) is computed as follows, where \( K(x) = \frac{d \log K(x)}{dx} \).

\[
\frac{\partial \hat{f}(v^1)}{\partial b_j} = \frac{1}{N^1} \sum_{i=1}^{N^1} \frac{\partial}{\partial b_j} \left[ \prod_{l=1}^{N^1} \frac{1}{b_j} K\left(\frac{v^1_l - v^1_i}{b_j}\right) \right] = -\frac{1}{N^1} \sum_{i=1}^{N^1} \left( \frac{1}{b_j} + \frac{v^1_l - v^1_i}{b^2_j} K\left(\frac{v^1_l - v^1_i}{b_j}\right) \right) \prod_{l=1}^{N^1} \frac{1}{b_j} K\left(\frac{v^1_l - v^1_i}{b_j}\right)
\]

Traditional KDE methods often fail on high-dimensional data [29, 60, 32, 56], when bandwidths need to be selected for each dimension. A greedy search method is utilized to select bandwidths in
the KDE: If a dimension $j$ is insignificant, then changing the bandwidth $b_j$ for that dimension should have a weak impact on $\hat{f}(v_i)$, while the changing $b_j$ for an important $j$ should cause a large change in $\hat{f}(v_i)$. Fortunately, $\frac{\partial \hat{f}(v_i)}{\partial b_j}$ can differentiate these two types of dimensions. Based on the above analysis, we greedily decrease $b_j$ with a sequence $b_0, b_0s, b_0s^2, \cdots$ for a parameter $0 < s < 1$, until $b_j$ is smaller than a certain threshold $\tau_j$, to see if a small change in $b_j$ can result in a large change in $\hat{f}(v_i)$. The method also offers a good way to estimate $[\frac{\partial \hat{f}(v_i)}{\partial b_1}, \cdots, \frac{\partial \hat{f}(v_i)}{\partial b_{N^1}}]$ along a sparse path.

Concretely, $\hat{f}(v_i)$ is estimated by beginning with an initial $B = \text{diag}(b_0, \cdots, b_0)$ for a large $b_0$, and then estimate $\frac{\partial \hat{f}(v_i)}{\partial b_j}$ as follows and decrease $b_j$ if $\frac{\partial \hat{f}(v_i)}{\partial b_j}$ is large.

$$
\frac{\partial \hat{f}(v_i)}{\partial b_j} = \frac{1}{N^1} \sum_{i=1}^{N^1} \frac{\partial}{\partial b_j} \left[ \prod_{l=1}^{N^1} \frac{1}{b_l^2} K\left( \frac{v_i - v_i^l}{b_l} \right) \right] = \frac{1}{N^1} \sum_{i=1}^{N^1} \frac{\partial}{\partial b_j} \left[ \prod_{l=1}^{N^1} \frac{1}{b_l^2} K\left( \frac{v_i - v_i^l}{b_l} \right) \right] K\left( \frac{v_i - v_i^l}{b_l} \right) = \frac{1}{N^1} \sum_{i=1}^{N^1} \frac{\partial \hat{f}(v_i)}{\partial b_j} 
$$

The corresponding variance $\text{Var}\left( \frac{\partial \hat{f}(v_i)}{\partial b_j} \right)$ is given below.

$$
\text{Var}\left( \frac{\partial \hat{f}(v_i)}{\partial b_j} \right) = \text{Var}\left( \frac{1}{N^1} \sum_{i=1}^{N^1} \frac{\partial \hat{f}(v_i)}{\partial b_j} \right) 
$$

Theorems 1 and 2 in Appendix A.4 provide the theoretical analysis about the density estimation, derivatives, and variances for well understanding the KDE technique.

In this work, assuming that the graph data follow the Gaussian distribution, a product Gaussian kernel $K$ is used to estimate the node density $\hat{f}(v_i)$. $\frac{\partial \hat{f}(v_i)}{\partial b_j}$ is accordingly updated as follows.

$$
\frac{\partial \hat{f}(v_i)}{\partial b_j} = \frac{C}{N^1} \sum_{i=1}^{N^1} \left( (v_i - v_i^l)^2 - b_j^2 \right) \prod_{l=1}^{N^1} K\left( \frac{v_i - v_i^l}{b_l} \right) \times \frac{1}{N^1} \sum_{i=1}^{N^1} \left( (v_i - v_i^l)^2 - b_j^2 \right) \prod_{l=1}^{N^1} K\left( \frac{v_i - v_i^l}{b_l} \right) 
$$

$$
= \frac{1}{N^1} \sum_{i=1}^{N^1} \left( (v_i - v_i^l)^2 - b_j^2 \right) \exp \left( - \sum_{l=1}^{N^1} \frac{(v_i - v_i^l)^2}{2b_j^2} \right) 
$$

where $C$ denotes a proportionality constant $C = \frac{1}{N^1} \prod_{l=1}^{N^1} \frac{1}{b_l^2}$. It can be safely ignored to avoid computation overflow when $b_l \rightarrow 0$ for large $N^1$. The bandwidth estimation is presented in Algorithm 1.

Based on the estimated $B$ and the Gaussian kernel $K$, the closed form of $\hat{f}(v_i)$ is derived below.

$$
\hat{f}(v_i) = \frac{1}{N^1} \sum_{i=1}^{N^1} \prod_{j=1}^{N^1} K\left( \frac{v_i - v_i^l}{b_l} \right) \sqrt{\frac{B + \Sigma}{|\Sigma|}} \times \exp \left( - \frac{(v_i - \mu)^T (\Sigma^{-1} - (B + \Sigma)^{-1}) (v_i - \mu)}{2} \right) 
$$

where $\mu$ and $\Sigma$ are the maximum likelihood estimation of the mean vector and covariance matrix of the Gaussian distribution. Please refer to Appendices A.6 and A.7 for detailed derivation of $\hat{f}(v_i)$.

As two graphs $G^1$ and $G^2$ often have different structures and distributions and thus the same KDE method as Algorithm 1 is utilized to estimate the density $\hat{g}(v_i)$ of $v_i$ and $\hat{g}(v_i^2)$, the attacker aims to maximize the following loss $L_D$ with imperceptible perturbations.

$$
L_D = \sum_{(v_i^1, v_i^2) \in D} L(\hat{v}_i^1, \hat{v}_i^2) \text{ where } L(\hat{v}_i^1, \hat{v}_i^2) = \|M(\hat{v}_i^1) - \hat{v}_i^2\|^2 + \hat{f}(\hat{v}_i^1) + \hat{g}(\hat{v}_i^2) 
$$

where $\hat{v}_i^1 = v_i^1 + \delta_1^i$ (and $\hat{v}_i^2 = v_i^2 + \delta_2^i$) denote adversarial versions of clean nodes $v_i^1$ (and $v_i^2$) in $G^1$ (and $G^2$) by adding a small amount of edge perturbations $\delta_1^i$ (and $\delta_2^i$) through our proposed MLPGD method in the next section, such that $M(\hat{v}_i^1)$ is far away from $\hat{v}_i^2$ and thus the matching accuracy is decreased. In addition, we push $v_i^1$ and $v_i^2$ to dense regions to generate $\hat{v}_i^1$ and $\hat{v}_i^2$, by maximizing $\hat{f}(\hat{v}_i^1)$ and $\hat{g}(\hat{v}_i^2)$, such that $\hat{v}_i^1$ and $\hat{v}_i^2$ are indistinguishable from their neighbors in perturbed graphs. This reduces the possibility of perturbation detection by humans or defender programs.
Algorithm 1 Bandwidth Matrix Estimation

Input: graph $G^1 = (V^1, E^1)$, parameter $0 < s < 1$, initial bandwidth $b_0$, and parameter $C$.
Output: Bandwidth matrix $C$.
1: Initialize all $b_1, \ldots, b_N$ with $b_0$.
2: for each $j = 1$ to $N^1$
3: do
4: Estimate the derivative $\frac{\partial f(x_j)}{\partial x_j}$ and variance $\text{Var}(\frac{\partial f(x_j)}{\partial x_j})$ in Eqs. (6)-(7):
5: Compute the threshold $\tau_j = \sqrt{2 \cdot \text{Var}(\frac{\partial f(x_j)}{\partial x_j})} \cdot \log(cN^1)$;
6: if $|\frac{\partial f(x_j)}{\partial x_j}| > \tau_j$, then Update $b_j = b_j s$;
7: while $|\frac{\partial f(x_j)}{\partial x_j}| > \tau_j$
8: Return $B$.

Algorithm 2 Meta Learning-based Projected Gradient Descent (MLPGD)

Input: Batches $D_1, \ldots, D_C$ in a set $D$ of node pairs, initial general policy parameters $\{\theta^1, \theta^2\}$, adaptation step size $\alpha$, meta step size $\beta$.
Output: Optimized $\{\theta^1, \theta^2\}$.
1: Repeat until convergence
2: Sample $C$ batches of anchor node pairs $D_1, \ldots, D_C$;
3: for $c = 1$ to $C$
4: Estimate gradient $e_c = e(D_c, \{\theta^1, \theta^2\})$;
5: Compute adapted parameters $\{\theta_c^1, \theta_c^2\} = \{\theta^1, \theta^2\} + \alpha e_c$;
6: Update parameters $\{\theta^1, \theta^2\} = \{\theta^1, \theta^2\} + \beta \sum_{c=1}^{C} e(D_c, \{\theta_c^1, \theta_c^2\})$;
7: Return $\{\theta^1, \theta^2\}$.

4 Effective Attacks via Meta Learning-based Projected Gradient Descent

In Figure 2, two dashed purple curves denote the decision boundary of graph matching. If we move a clean node $v^1_k$ across the decision boundary to generate an adversarial node $\hat{v}^1_k$, then we have other nodes $v^1_j$ to make $M(v^1_j)$ and $v^2_k$ become more similar than $M(\hat{v}^1_k)$ and $v^2_k$, and thus a wrong matching $(v^1_j, v^2_k)$ will be produced. Blue and green polylines denote attack trajectories starting from original and good starting points with gradient descent method respectively. A shortcut from good starting points $(v^1_0)^0$ or $(v^2_0)^0$ is able to cross the peak of the decision boundary and converge quickly, while the trajectories from the original nodes $v^1_0$ or $v^2_0$ take long walks to cross the non-peak boundary.

Based on the attack loss in Eq. (9), we propose to integrate meta learning and PGD into an MLPGD model, to produce more effective adversarial nodes with good starting points towards graph matching.

\[
(v^1_t)^{(t+1)} = \Pi_{\Delta^1_t} \text{sgn}[\text{ReLU}(\nabla_{(v^1_t)^t} \mathcal{L}((v^1_t)^t, (v^2_t)^t))] \\
(v^2_t)^{(t+1)} = \Pi_{\Delta^2_t} \text{sgn}[\text{ReLU}(\nabla_{(v^2_t)^t} \mathcal{L}((v^1_t)^t, (v^2_t)^t))], \quad t = 1, \ldots, T
\]

where $(v^1_t)^t$ and $(v^2_t)^t$ denotes the adversarial nodes of $v^1_t$ and $v^2_t$ derived at step $t$. $\epsilon$ specifies the budget of allowed perturbed edges for each attacked node. $\Delta^1_t = \{\delta^1_t\}^T \leq \epsilon, (\delta^1_t)^t \in \{0, 1\}^{N_1}$, where $(\delta^1_t)^t = \|((v^1_t)^t) - (v^1_t)^t\|^2_2$, represents the constraint set of the projection operator $\Pi$, i.e., it encodes whether an edge of $v^1_t$ is modified or not. $\Delta^2_t$ has the similar definition for $v^2_t$. The composition of the ReLU and sign operators guarantees $(v^1_0)^t \in \{0, 1\}^{N_1}$ and $(v^2_0)^t \in \{0, 1\}^{N_2}$, as it adds (or removes) an edge or keeps it unchanged when an derivative in the gradient is positive (or negative). The outputs $(v^1_T)^T$ and $(v^2_T)^T$ at final step $T$ are used as the adversarial nodes $\hat{v}^1_T$ and $\hat{v}^2_T$.

Searching for attack starting points for each $(v^1_0, v^2_0)$ in large graphs is computationally inefficient. Meta learning techniques aim to train a general model with general parameters that can quickly adapt to a variety of new learning tasks with refined parameters [23, 3, 42, 56]. This offers a great opportunity to find good attack starting points $(v^1_0)^0$ and $(v^2_0)^0$ for all $(v^1, v^2) \in D$ with lower cost, such that the generated $\hat{v}^1_T$ and $\hat{v}^2_T$ by the PGD model can maximize the attack loss $\mathcal{L}_D$ in Eq. (9).
Algorithm 3 Gradient Estimation $e(D_c, \{\theta^1, \theta^2\}, N, \lambda$

**Input:** Batch $D_c$, general parameters $\{\theta^1, \theta^2\}$, number of samples $N$ in Monte Carlo REINFORCE, smoothing parameter $\lambda$.

**Output:** Gradient estimation of $a$.

1: Sample $N$ i.i.d. Gaussian matrices $g_1, \ldots, g_N \sim \mathcal{N}(0, I)$;
2: **Return** gradient estimation $\frac{1}{N\lambda} \sum_{i=1}^{N} a(D_c, \{\theta^1, \theta^2\} + \lambda g_i)$.

Algorithm 4 Adversarial Attack $a(D_c, \{\theta^1_c, \theta^2_c\})$

**Input:** Batch $D_c$, perturbation budget $\epsilon$, specific parameters $\{\theta^1_c, \theta^2_c\}$

**Output:** Attack loss $L_{D_c}$ on $D_c$.

1: $L_{D_c} = 0$;
2: **for** each $(v^1_c, v^2_c) \in D_c$
3: Generate attack starting points $(v^1_c)^0 = h^1((v^1_c)^0)$ and $(v^2_c)^0 = h^2((v^2_c)^0)$;
4: Utilize PGD attack to generate adversarial nodes $(v^1_c)^T$ and $(v^2_c)^T$ in Eq.(10);
5: Aggregate attack loss $L_{D_c} = L((v^1_c)^T, (v^2_c)^T)$ in Eq.(9);
6: **Return** $L_{D_c}$.

Algorithm 2 presents the pseudo code of our MLPGD model. $D$ is partitioned into $C$ batches $D_1, \ldots, D_C$, each with equal size of $|D|/C$. The search process on each batch $D_c$ ($1 \leq c \leq C$) is treated as a single task, which aims to find good $(\theta^1)^0$ and $(\theta^2)^0$ for $D_c$ to maximize the attack loss $L_{D_c} = \sum_{(v^1_c, v^2_c) \in D_c} L((v^1_c), (v^2_c))$. A general model that has general parameters $\theta^1, \theta^2$ is learnt to quickly adapt to search tasks on multiple batches. The learnt $\theta^1, \theta^2$ should be sensitive to changes of each $D_c$, such that small changes in $\theta^1, \theta^2$ will produce high rise on $L_{D_c}$ over any of $D_1, \ldots, D_C$. Line 4 estimates the gradient of $L_{D_c}$ by calling Algorithm 3. In Line 5, when adapting to the task on a new $D_c$, $\theta^1, \theta^2$ become specific parameters $\theta^1_c, \theta^2_c$ for $D_c$. Here, we use $\{\theta^1_c, \theta^2_c\}$ to denote the concatenation matrix of $\theta^1$ and $\theta^2$. The parameters are trained by maximizing the attack loss $a(D_c, \{\theta^1_c, \theta^2_c\})$ w.r.t. general parameters $\theta^1, \theta^2$ across batches. The meta objective is given below.

$$\max_{\theta^1, \theta^2} \max_{\theta^1_c, \theta^2_c} \sum_{c=1}^{C} a(D_c, \{\theta^1_c, \theta^2_c\}) = \sum_{c=1}^{C} a(D_c, \{\theta^1_c, \theta^2_c\}) + \alpha_c$$

(11)

In Line 6, the meta optimization is performed over the general $\theta^1, \theta^2$, while the objective is computed using the specific $\theta^1_c, \theta^2_c$. The general $\theta^1, \theta^2$ are updated in terms of the attack loss on each batch.

$$\{\theta^1, \theta^2\} = \{\theta^1, \theta^2\} + \frac{\beta}{C} \sum_{c=1}^{C} e(D_c, \{\theta^1_c, \theta^2_c\})$$

(12)

Algorithm 4 exhibits the adversarial attack module $a(D_c, \{\theta^1_c, \theta^2_c\})$ on a batch $D_c$ ($1 \leq c \leq C$). In Line 3, two neural networks $h^1$ and $h^2$ with specific parameters $\theta^1_c$ and $\theta^2_c$ are designed to generate the attack starting points $(v^1_c)^0$ and $(v^2_c)^0$ of each $(v^1_c, v^2_c) \in D_c$. The last layers of $h^1$ and $h^2$ use the composition of the ReLU [53] and Softsign [25] as activation function to ensure $(v^1_c)^0 \in \{0, 1\}^N$ and $(v^2_c)^0 \in \{0, 1\}^N$. In Line 4, the PGD attack in Eq.(10) is utilized to generate the adversarial nodes $\tilde{v}^1_c$ and $\tilde{v}^2_c$. Line 5 calculates the attack loss $L_{D_c}$ on $D_c$ to provide task-specific feedback.

Standard meta learning utilizes gradient ascent/descent techniques to compute the updated parameters on new tasks $\{23\}, \{42\}, \{50\}$. However, the attack module in Algorithm 4 is non-smooth and non-differentiable w.r.t. parameters $\theta^1, \theta^2, \theta^1_c$ and $\theta^2_c$, since the perturbation is a multi-step process as well as the projection at each step is non-differential. Therefore, Algorithm 3 is proposed to employ Gaussian smoothing technique to approximate a smoothed attack module.

$$\hat{a}(D_c, \{\theta^1, \theta^2\}) \approx (2\pi)^{-d/2} \int a(D_c, \{\theta^1, \theta^2\} + \lambda g) \exp \left(-\frac{1}{2} \|g\|^2\right)dg$$

(13)

where $\hat{a}$ is the Gaussian smoothing of $a$ and differentiable everywhere. $\lambda$ is a smoothing parameter, and $d$ is the number of entries in $\{\theta^1, \theta^2\}$. $g \sim \mathcal{N}(0, I)$ that has the same size as $\{\theta^1_c, \theta^2_c\}$ is interpreted as policy exploration directions, i.e., as perturbations in policy space to be explored. Thus, the policy perturbations in $g$ are introduced to $\theta^1_c$ and $\theta^2_c$ respectively. $\hat{a}$ is obtained by perturbing $a$ at a given point along Gaussian directions and averaging the evaluations. And then, Algorithm 3 estimates the gradient of $\hat{a}$ via Monte Carlo REINFORCE method [79].
Table 2: Mismatching rate (%) with 5% perturbed edges

<table>
<thead>
<tr>
<th>Attack Model</th>
<th>AS</th>
<th>SNS</th>
<th>DBLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean</td>
<td>53.9</td>
<td>46.6</td>
<td>34.7</td>
</tr>
<tr>
<td>Random</td>
<td>57.5</td>
<td>49.9</td>
<td>37.6</td>
</tr>
<tr>
<td>RL-S2V</td>
<td>56.5</td>
<td>51.8</td>
<td>36.5</td>
</tr>
<tr>
<td>Meta-Self</td>
<td>63.1</td>
<td>55.1</td>
<td>45.0</td>
</tr>
<tr>
<td>CW-PGD</td>
<td>61.7</td>
<td>59.1</td>
<td>49.6</td>
</tr>
<tr>
<td>GF-Attack</td>
<td>57.9</td>
<td>53.7</td>
<td>39.5</td>
</tr>
<tr>
<td>CD-ATTACK</td>
<td>59.0</td>
<td>51.7</td>
<td>42.7</td>
</tr>
<tr>
<td>GMA</td>
<td>64.2</td>
<td>62.9</td>
<td>54.9</td>
</tr>
</tbody>
</table>

\[
e(D_c, \{\theta^1, \theta^2\}) \approx \nabla_\theta^1,\theta^2 a(D_c, \{\theta^1, \theta^2\}) \approx (2\pi)^{-\frac{d}{2}} \int a(D_c, \{\theta^1, \theta^2\}) + \lambda g \exp\left(-\frac{1}{2}\|\mathbf{g}\|^2\right) g d\mathbf{g} \\
= \frac{1}{N} \sum_{i=1}^{N} a(D_c, \{\theta^1, \theta^2\} + \lambda g_i) g_i, g_i \sim \mathcal{N}(0, I)
\]

5 Experimental Evaluation

Table 1: Experiment Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AS</th>
<th>SNS</th>
<th>DBLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph</td>
<td>v1</td>
<td>v2</td>
<td>Last.fm LiveJournal 2013 2014</td>
</tr>
<tr>
<td>#Nodes</td>
<td>10,900</td>
<td>11,113</td>
<td>5,682 17,828 28,478 26,455</td>
</tr>
<tr>
<td>#Edges</td>
<td>31,180</td>
<td>31,434</td>
<td>23,393 244,496 128,073 114,588</td>
</tr>
<tr>
<td>#Matched Nodes</td>
<td>6,462</td>
<td>2,138</td>
<td>2,343 4,000</td>
</tr>
</tbody>
</table>

BLP coauthor graphs [11], as shown in Table 1.

Baselines. We compare the GMA model with six state-of-the-art graph attack models. Random Attack randomly adds and removes edges to generate perturbed graphs. RL-S2V [14, 123] generates adversarial attacks on graph data based on reinforcement learning. Meta-Self [125] is a poisoning attack model for node classification by using meta-gradients to solve the bilevel optimization problem. CW-PGD [86] developed a PGD topology attack to attack a predefined or a retrainable GNN. GF-Attack [5] attacks general learning methods by devising new loss and approximating the spectrum. CD-ATTACK [41] hides nodes in the community by attacking the graph autoencoder model. The majority of existing efforts focus on adversarial attacks on single graph learning. To our best knowledge, there are no other attack baselines on graph matching available. We replace the original losses in the baselines with the matching loss for fair comparison in the experiments.

Variants of GMA model. We evaluate four variants of GMA to show the strengths of different components. GMA-KDE only uses the KDE and density maximization to generate imperceptible attacks. GMA-PGD employs our proposed MLPGD model to well choose good attack starting points in the PGD. GMA operates with the full support of both KDE and MLPGD components.

Graph matching algorithms. We validate the effectiveness of the above attack models with three representative deep graph matching methods. SNNA [39] is an adversarial learning framework to solve the weakly-supervised identity matching problem by minimizing the distribution distance. CrossMNA [13] is a cross-network embedding-based supervised network alignment method by learning inter/intra-embedding vectors for each node and by computing pairwise node similarity scores across networks. Deep graph matching consensus (DGMC) [22] is a supervised graph matching method that reaches a data-driven neighborhood consensus between matched node pairs.

Evaluation metrics. We use two popular measures in graph matching to verify the attack quality: Accuracy [25, 10, 95] and Precision@K [101, 13, 97]. A larger mismatching rate (i.e., 1 - Accuracy on test data) or a smaller Precision@K shows a better attack. K is fixed to 30 in all tests.

Attack performance on various datasets with different matching algorithms. Table 2 exhibits the mismatching rates of three deep graph matching algorithms on test data by eight attack models over three groups of datasets. We randomly sample 10% of known matched node pairs as training data and the rest as test data. For all attack models, the number of perturbed edges is fixed to 5% in these experiments. It is observed that among eight attack methods, no matter how strong the attacks are, the GMA method achieves the highest mismatching rates on perturbed graphs in most experiments, showing the effectiveness of GMA to the adversarial attacks. Compared to the graph matching results under other attack models, GMA, on average, achieves 21.3%, 18.8%, and 19.2%
Attack performance with varying perturbation edges. Figure 3 presents the graph matching quality under eight attack models by varying the ratios of perturbed edges from 2% to 25%. It is obvious that the attacking performance improves for each attacker with an increase in the number of perturbed edges. This phenomenon indicates that current deep graph matching methods are very sensitive to adversarial attacks. GMA achieves the lowest Precision values (< 0.488), which are still better than the other seven methods in most tests. Especially, when the perturbation ratio is large than 10%, the Precision values drop quickly.

Impact of training data ratios. Figure 4 shows the quality of two graph matching algorithms on SNS by varying the ratio of training data from 2% to 25%. Here, the number of perturbed edges is fixed to 5%. We make the following observations on the performances by eight attack models. (1) The performance curves keep increasing when the training data ratio increases. (2) GMA outperforms other methods in most experiments with the lowest Precision: < 0.482 with SNNA and < 0.571 with DGMC respectively. Even when there are many training data available (≥ 20%), the quality degradation by GMA is still obvious, although more training data makes the graph matching models be resilient to poisoning attacks under a small perturbation budget.

Ablation study. Figure 5 presents the mismatching rates of graph matching on SNS with four variants of the GMA attack model. We have observed the complete GMA achieves the highest mismatching rates (> 54.9%) on AS, (> 55.7%) over SNS, and (> 74.2%) on DBLP, which are obviously better than other versions. Notice that GMA-MLPGD performs quite well in most experiments, compared with GMA-PGD. A reasonable explanation is that searching from good attack starting points can help the MLPGD converge quickly by crossing the peak of the decision boundary. In addition, GMA-KDE achieves the better attack performance than GMA-MLPGD. A rational guess is that it is difficult to correctly match two nodes results when they lie in dense regions with many similar nodes, although the main goal of KDE is to generate imperceptible attacks. These results illustrate both KDE and MLPGD models are important in producing effective attacks in graph matching.

Impact of perturbation budget $\epsilon$. Figure 6(a) measures the performance effect of $\epsilon$ in the MLPGD model for the graph matching by varying $\epsilon$ from 1 to 5. It is observed that when increasing $\epsilon$, the Precision of the GMA model decreases substantially. This demonstrates that it is difficult to train a robust graph matching model under large $\epsilon$ constraint. However, a large $\epsilon$ can be easily detected by humans or by defender programs. Notice that the average node degree of three groups of datasets is between 2.9 and 13.9. Thus we suggest generating both imperceptible and effective attacks for the graph matching task under $\epsilon$ between 2 and 3, such that $\epsilon$ is smaller than the average node degree.

Time complexity analysis. Based on [20], the complexity of meta learning is $O(d^2)$, where $d$ is the problem dimension. In the context of graph matching, it is the number of nodes in two graphs ($N^s$, $s = 1$ or 2). Both density estimation and PGD have complexity of $O((N^s)^2)$. Thus, the overall complexity is $O((N^s)^2)$, which is the same as most existing attack methods that search the entire graphs to find weak edges to attack.

Impact of meta step size $\alpha$. Figure 6(b) shows the impact of $\alpha$ in our MLPGD model over three groups of datasets. The performance curves initially raise when $\alpha$ increases. Intuitively, the MLPGD with large $\alpha$ can help the meta learning converge quickly. Later on, the performance curves keep relatively stable or even decreasing when $\alpha$ continuously increases. A reasonable explanation is that
6 Related Work

Adversarial Attacks on Graph-Based Data. Several recent studies have presented that graph learning models, especially deep learning-based models, are highly sensitive to adversarial attacks, i.e., carefully designed small deliberate perturbations in graph structure and attributes can cause the models to produce incorrect prediction results [64,69,90,74,85,80,127]. The current graph adversarial attack techniques mainly fall into two categories in terms of the attack surface: (1) evasion attacks occur after the target model is well trained in clean graphs, i.e., the learned model parameters are fixed during evasion attacks. The attacker tries to evade the graph learning models by generating malicious samples during testing phase [14,126]; and (2) poisoning attacks, known as contamination of the training data, take place during the training time of deep learning models. An adversary tries to poison the training data by injecting carefully designed examples to cause failures of the target model on some given test samples [125,69,123]. Since transductive learning is widely used in most graph analysis tasks, the test samples (but not their labels) are participated in the training stage, which leads to the popularity of poisoning attacks. Various adversarial attack models have been developed to show the vulnerability of graph learning models in node classification [14,126,74,86,125,71,19,70], community detection [9,78,7,41], network embedding [6,4,5], link prediction [104], similarity search [15], malware detection [30], and knowledge graph embedding [90].

Graph Matching. Graph data analysis has attracted active research in the last decade [110,111,11,106,12,107,108,66,113,109,112,114,35,115,117,109,112,81,82,120,121]. Graph matching is one of the most important research topics in the graph domain, which aims to match the same entities (i.e., nodes) across two or more graphs and has been a heated topic in recent years [91,98,62,46,48,101,13]. Research activities can be classified into three broad categories. (1) Topological structure-based techniques, which rely on only the structural information of nodes to match multiple or two input networks, including IONE [43], GeoAlign [45], Low-rank EigenAlign [44], FRUI-P [105], CrossMNA [13], MOANA [96], GWL [84], MSUIL [48], DeepMGGE [24], and KEMINA [122]; (2) Structure and/or attribute-based approaches, which utilize highly discriminative structure and attribute features for ensuring the matching effectiveness, such as FINAL [93,94], ULink [51], CALign [10], MASTER [65], gsaNA [88], CoLink [100], REGAL [28], UUIL [37], SNNA [49], RANA [58], CENALP [18], ORIGIN [97], OPTANE [50], and Deep Graph Matching Consensus [22]; (3) Heterogeneous methods employ heterogeneous structural, content, spatial, and temporal features to further improve the matching performance, including COSNET [98], Factoid Embedding [83], HEP [99], LHNE [77], and DPLink [21]. Several papers review key achievements of graph matching across online information networks including state-of-the-art algorithms, evaluation metrics, representative datasets, and empirical analysis [62,27,31,73].

7 Conclusions

In this work, we have studied the graph matching adversarial attack problem. First, we proposed to utilize kernel density estimation technique to estimate and maximize the densities of attacked nodes and generate imperceptible perturbations, by pushing attacked nodes to dense regions in two graphs. Second, we developed a meta learning based projected gradient descent method to well choose attack starting points and improve the search performance of PGD for producing effective perturbations. The GMA model achieves superior attack performance against several representative attack models.
Broader Impact

Graph data are ubiquitous in the real world, ranging from biological, communication, and transportation graphs, to knowledge, social, and collaborative networks. Many real-world graphs are essentially crowdsourced projects, such as social and knowledge networks, where information and knowledge are produced by internet users who came to the sites. Thus, the quality of crowdsourced graph data is not stable, depending on human knowledge and expertise. In addition, it is well known that the openness of crowdsourced websites makes them vulnerable to malicious behaviors of interested parties to gain some level of control of the websites and steal users’ sensitive information, or deliberately influence public opinion by injecting misleading information and knowledge into crowdsourced graphs.

Graph matching is one of the most important research topics in the graph domain, which aims to match the same entities (i.e., nodes) across two or more graphs [91, 98, 43, 46, 48, 72, 54, 105, 13, 75]. It has been widely applied to many real-world applications ranging from protein network matching in bioinformatics [33, 63], user account linking in different social networks [62, 51, 100, 37, 101, 21, 38], and knowledge translation in multilingual knowledge bases [87, 124], to geometric keypoint matching in computer vision [22]. Owing to the openness of crowdsourced graphs, more work is needed to analyze the vulnerability of graph matching under adversarial attacks and to future develop robust solutions that are readily applicable in production systems.

A potential downside of this research is about the application of user account linking in different social networks due to the user privacy issues. Recent advances in differential privacy and privacy preserving graph analytics have shown the superior performance of protecting sensitive information about individuals in the datasets. Therefore, these techniques offer a great opportunity to integrate them into the vulnerability analysis of graph matching, for alleviating the user privacy threats.

References


