TDGIA: Effective Injection Attacks on Graph Neural Networks

Xu Zou[†], Qinkai Zheng[+], Yuxiao Dong[‡], Xinyu Guan[•]
Evgeny Kharlamov[◇], Jialiang Lu[+], Jie Tang[†§]
[†] Department of Computer Science and Technology, Tsinghua University, [‡] Facebook AI
[+] Shanghai Jiao Tong University, [•] Biendata, [◇] Bosch Center for Artificial Intelligence
zoux18@mails.tsinghua.edu.cn
{qinkai.zheng1028,ericdongyx,guanxinyu}@gmail.com,evgeny.kharlamov@de.bosch.com
jialiang.lu@sjtu.edu.cn,jietang@tsinghua.edu.cn

ABSTRACT

Graph Neural Networks (GNNs) have achieved promising performance in various real-world applications. However, recent studies find that GNNs are vulnerable to adversarial attacks. In this paper, we study a recently-introduced realistic attack scenario on graphs—graph injection attack (GIA). In the GIA scenario, the adversary is not able to modify the existing link structure or node attributes of the input graph, instead the attack is performed by injecting adversarial nodes into it. We present an analysis on the topological vulnerability of GNNs under GIA setting, based on which we propose the Topological Defective Graph Injection Attack (TDGIA) for effective injection attacks. TDGIA first introduces the topological defective edge selection strategy to choose the original nodes for connecting with the injected ones. It then designs the smooth feature optimization objective to generate the features for the injected nodes. Extensive experiments on large-scale datasets show that TDGIA can consistently and significantly outperform various attack baselines in attacking dozens of defense GNN models. Notably, the performance drop on target GNNs resultant from TDGIA is more than double the damage brought by the best attack solution among hundreds of submissions on KDD-CUP 2020.

CCS CONCEPTS

• **Security and privacy → Software and application security**; • **Mathematics of computing → Graph algorithms**.

KEYWORDS

Graph Neural Networks; Adversarial Machine Learning; Graph Injection Attack; Graph Mining; Graph Adversarial Attack

ACM Reference Format:
Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, Jie Tang. 2021. TDGIA: Effective Injection Attacks on Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467314

[§]Jie Tang is the corresponding author.

1 INTRODUCTION

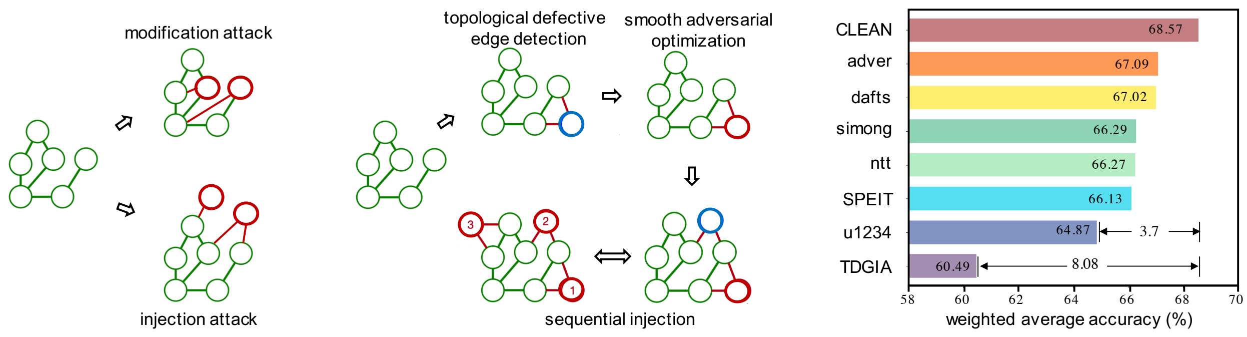
Recent years have witnessed widespread adoption of graph machine learning for modeling structured and relational data. Particularly, the emergence of graph neural networks (GNNs) has offered promising results in diverse graph applications, such as node classification [14], social recommendation [22], and drug design [12].

Despite the exciting progress, studies have shown that neural networks are commonly vulnerable to adversarial attacks, where slight, imperceptible but intentionally-designed perturbations on inputs can cause incorrect predictions [7, 11, 18]. Attacks on general neural networks usually focus on modifying the attributes/features of the input instances, such as minor perturbations in individual pixels of an image. Uniquely, adversarial attacks can also be applied to graph-structured input, requiring dedicated strategies for exploring the specific vulnerabilities of the underlying models.

The early attacks on GNNs usually follow the setting of graph modification attack (GMA) [5, 17, 25, 26], as illustrated in Figure 1 (a): given an input graph with attributes, the adversary can directly modify the links between its nodes (red links) and the attributes of existing nodes (red nodes). However, in real-world scenarios, it is often unrealistic for the adversary to get the authority to modify existing data. Take the citation graph for example, it automatically forms when papers are published, making it practically difficult to change the citations and attributes of one publication afterwards. But what is relatively easy is to inject new nodes and links into the existing citation graph, e.g., by "publishing" fake papers, to mislead the predictions of GNNs.

In view of the gap, very recent efforts [17, 19], including the KDD-CUP 2020 competition[1], have been devoted to adversarial attacks on GNNs under the setting of graph injection attack (GIA). Specifically, the GIA task in KDD-CUP 2020 is formulated as follows: (1) Black-box attack, where the adversaries do not have access to the target GNN model or the correct labels of the target nodes; (2) Evasion attack, where the attacks can only be performed during the inference stage. The GIA settings present unique challenges that are not faced by GMA, such as how to connect existing nodes with injected nodes and how to generate features for injected nodes from scratch. Consequently, though numerous attacks are submitted by hundreds of teams, the resultant performance drops are relatively limited and no principled models emerge from them.

[1]https://www.biendata.xyz/competition/kddcup_2020_formal/



(a) GMA vs. GIA (b) Topological Defective Graph Injection Attack (TDGIA) (c) TDGIA vs. The Best Results in KDD-CUP 2020

Figure 1: An overview of graph injection attack, the proposed TDGIA method, and its performance.

Contributions. In this work, we study the problem of GIA under the black-box and evasion attack settings, where the goal is to design an effective injection attack framework that can best fool the target GNN models and thus worsen their prediction capability. To understand the problem and its challenges, we present an in-depth analysis of the vulnerability of GNNs under the graph injection attack and show that GNNs, as non-structural-ignorant models, are GIA-attackable. Based on this, we present the topological defective graph injection attack (TDGIA) (Cf. Figure 1 (b)). TDGIA consists of two modules—topological defective edge selection and smooth adversarial optimization for injected node attribute generation—that corresponds to the problem setup of GIA. Specifically, we leverage the topological vulnerabilities of the original graph to detect existing nodes that can best help the attack and then inject new nodes surrounding them in a sequential manner. With that, we design a smooth loss function to optimize the nodes' features for minimizing the the performance of the target GNN model.

Both the studied problem and the proposed TDGIA method differ from existing (injection) attack methods. Table 1 summarizes the differences. First, NIPA [17] and AFGSM [19] are developed under the poison setting, which requires the re-training of the defense models for each attack. Differently, TDGIA follows KDD-CUP 2020 to use the evasion attack setting, where different attacks are evaluated based on the same set of models and weights. Second, the design of TDGIA enables it to attack large-scale graphs that can not be handled by the reinforcement-learning based NIPA. Third, compared with AFGSM, TDGIA proposes a more general way to consume the topological information, resulting in significant performance improvements. Finally, attacks in previous works are only evaluated on weak defense models like raw GCN, while TDGIA is shown to be effective even against the top defense solutions examined in KDD-CUP 2020.

We conduct extensive experiments on large-scale datasets to demonstrate the performance and transferability of the proposed attack method. Figure 1 (c) lists the results for TDGIA and the top submissions on KDD-CUP 2020 as measured by weighted average accuracy. The experimental results show that TDGIA significantly

Table 1: Summary of graph adversarial attacks.

Attack	Task	Type	Method	Attack Setting
Dai et al. [5]	Node cl. Graph cl.	GMA	Reinforcement learning	Evasion
Nettack [25]	Node cl. Graph cl.	GMA	Greedy algorithm Model linearization	Evasion Poison
Meta [26]	Node cl.	GMA	Meta-learning	Poison
NIPA [17]	Node cl.	GIA	Reinforcement learning	Poison
AFGSM [19]	Node cl.	GIA	Fast Gradient Sign Model linearization	Poison
TDGIA (ours)	Node cl.	GIA	Defective edge selection Smooth optimization	Evasion

and consistently outperforms various baseline methods. For example, the best KDD-CUP 2020 attack (u1234) can make the performance of the target GNN models drop 3.7%, while TDGIA can drag its performance down by 8.08%—a 118% increase in damage. Moreover, TDGIA achieves this attack performance by injecting only a limited number of nodes (1% of target nodes). Additionally, various ablation studies demonstrate the effectiveness of each module in TDGIA.

To sum up, this work makes the following contributions:

- We study the GIA problem with the black-box and evasion settings, and theoretically show that non-structural-ignorant GNN models are vulnerable to GIA.
- We develop the Topological Defective Graph Injection Attack (TDGIA) that can explore and leverage the vulnerability of GNNs and the topological properties of the graph.
- We conduct experiments that consistently demonstrate TDGIA's significant outperformance over baselines (including the best attack submission at KDD-CUP 2020) against various defense GNN models across different datasets.

2 RELATED WORKS

Adversarial Attacks on Neural Networks. The phenomenon of adversarial examples against deep-learning based models is first discovered in computer vision [18]. Adding delicate and imperceptible perturbations to images can significantly change the predictions of deep neural networks. [8] proposes Fast Gradient Sign Method (FGSM) to generate this kind of perturbations by using the gradients of trained neural networks. Since then, more advanced attack methods are proposed [1, 3]. Adversarial attacks also grow in a wider range of fields such as natural language processing [16], or speech recognition [4]. Nowadays, adversarial attacks have become one of the major threats to neural networks.

Adversarial Attacks on GNNs. Due to the existence of adversarial examples, the vulnerability of graph learning algorithms has also been revealed. By modifying features and edges on graph-structured data, the adversary can significantly degrade the performance of GNN models. As shown in Table 1, [5] proposes a reinforcement-learning based attack on both node classification and graph classification tasks. This attack only modifies the structure of graph. [25] proposes Nettack, the first adversarial attack on attributed graphs. It shows that only few perturbations on both edges and features of the graph can be extremely harmful for models like GCN. Nettack uses a greedy approximation scheme under the constraints of unnoticeable perturbations and incorporates fast computation. Furthermore, [26] proposes a poison attack on GNNs via meta-learning, Meta-attack, which only modifies a small part of the graph but can still decrease the performance of GNNs on node classification tasks remarkably. A recent summary [13] summarizes different graph adversarial attacks.

A more realistic scenario, graph injection attack (GIA), is studied in [17, 19], which injects new vicious nodes instead of modifying the original graph. [17] proposes Node Injection Poisoning Attack (NIPA) based on reinforcement learning strategy. Under the same scenario, Approximate Fast Gradient Sign Method (AFGSM) [19] further uses an approximation strategy to linearize the model and to generate the perturbations efficiently. These works are under the *poison setting*, where models have to be re-trained after the vicious nodes are injected onto the graph.

KDD-CUP 2020 Graph Adversarial Attack & Defense. KDD-CUP 2020 introduces the GIA scenario in *Graph Adversarial Attack & Defense* competition track. The adversary doesn't have access to the target model (i.e. *black-box setting*), and can only inject no more than 500 nodes to a large citation network with more than 600,000 nodes and millions of links.

For attackers, the attack is conducted during the inference stage, a.k.a. the *evasion setting*. For defenders, they should provide robust GNN models to resist various adversarial attacks. Hundreds of attack and defense solutions are submitted during the competition, which serve as good references for further development of graph adversarial attacks.

3 PROBLEM DEFINITION

Broadly, there are two types of graph adversarial attacks: graph modification attack (GMA) and graph injection attack (GIA). The focus of this work is on graph injection attack. We formalize the attack problem and introduce the attack settings.

The problem of graph adversarial attack was first formalized in Nettack [25], which we name as graph modification attack. Specifically, we define an attributed graph $\mathbf{G} = (\mathbf{A}, \mathbf{F})$ with $\mathbf{A} \in \mathbb{R}^{N\times N}$ being the adjacency matrix of its N nodes and $\mathbf{F} \in \mathbb{R}^{N\times D}$ as its D-dimensional node features. Let $\mathcal{M} : \mathbf{G} \rightarrow \{1, 2, ..., C\}^N$ be a model that predicts the labels for all N nodes in $\mathbf{G}$, and denote its predictions as $\mathcal{M}(\mathbf{G})$. The goal of GMA is to minimize the number of correct predictions of $\mathcal{M}$ on a set of target nodes $\mathcal{T}$ by modifying the original graph $\mathbf{G}$:

$$\begin{aligned} &\min_{\mathbf{G}'} |\{\mathcal{M}(\mathbf{G}')_i = y_i, i \in \mathcal{T}\}| \\ &s.t.\ \mathbf{G}' = (\mathbf{A}',\ \mathbf{F}'), f_{\Delta_\mathbf{A}}(\mathbf{A}' - \mathbf{A}) + f_{\Delta_\mathbf{F}}(\mathbf{F}' - \mathbf{F}) \le \Delta \end{aligned} \tag{1}$$

where $\mathbf{G}'$ is the modified graph, y_i is the ground truth label of node i, $f_{\Delta_\mathbf{A}}$ and $f_{\Delta_\mathbf{F}}$ are pre-defined functions that measure the scale of modification. The constraint Δ ensures that the graph can only be slightly modified by the adversary.

Graph Injection Attack. Instead of GMA's modifications of $\mathbf{G}$'s structure and attributes, GIA directly injects N_I new nodes into $\mathbf{G}$ while keeping the original edges and attributes of N nodes unchanged. Formally, GIA constructs $\mathbf{G}' = (\mathbf{A}', \mathbf{F}')$ with

$$\mathbf{A}' = \begin{bmatrix} \mathbf{A} & \mathbf{V}_I \\ \mathbf{V}_I^T & \mathbf{A}_I \end{bmatrix}, \mathbf{A} \in \mathbb{R}^{N\times N}, \mathbf{V}_I \in \mathbb{R}^{N\times N_I}, \mathbf{A}_I \in \mathbb{R}^{N_I\times N_I} \tag{2}$$

$$\mathbf{F}' = \begin{bmatrix} \mathbf{F} \\ \mathbf{F}_I \end{bmatrix}, \mathbf{F} \in \mathbb{R}^{N\times D}, \mathbf{F}_I \in \mathbb{R}^{N_I\times D} \tag{3}$$

where $\mathbf{A}_I$ is the adjacency matrix of the injected nodes, $\mathbf{V}_I$ is a matrix that represents edges between $\mathbf{G}$'s original nodes and the injected nodes, and $\mathbf{F}_I$ is the feature matrix of the injected nodes. The objective of GIA can be then formalized as:

$$\begin{aligned} &\min_{\mathbf{G}'} |\{\mathcal{M}(\mathbf{G}')_i = y_i, i \in \mathcal{T}\}| \\ &s.t.\ \mathbf{G}' = (\mathbf{A}', \mathbf{F}'), N_I \le b, deg(v)_{v\in I} \le d, ||\mathbf{F}_I|| \le \Delta_F \end{aligned} \tag{4}$$

where I is the set of injected nodes, N_I is limited by a budget b, each injected node's degree is limited by a budget d, and the norm of injected features are restricted by Δ_F. These constraints are to ensure that GIA is as unnoticeable as possible by the defender.

The Attack Settings of GIA. GIA has recently attracted significant attentions and served as one of the KDD-CUP 2020 competition tasks. Considering its widespread significance in real-world scenarios, we follow the same settings used in the competition, that is, black-box and evasion attacks.

Black-box attack. In the black-box setting, the adversary does not have access to the target model $\mathcal{M}$, including its architecture, parameters, and defense mechanism. However, the adversary is allowed to access the original attributed graph $\mathbf{G} = (\mathbf{A}, \mathbf{F})$ and labels of training and validation nodes but not the ones to be attacked.

Evasion attack. Straightforwardly, GIA follows the evasion attack setting in which the attack is only performed to the target model during inference. This makes it different from the poison attack [17, 19], where the target model is retrained on the attacked graph.

In addition, the scale of the KDD-CUP 2020 dataset is significantly larger than those commonly used in existing graph attack studies [19, 25, 26]. This makes the task more relevant to real-world applications and also requires more scalable attacks.

The GIA Process. Due to the black-box and evasion settings, GIA needs to conduct transfer attack with the help of surrogate models as done in [5, 25]. First, a surrogate model is trained; Second, injection attack is performed on this model; Finally, the attack is transferred to one or more target models.

To handle large-scale datasets, GIA can be separated into two steps based on its definition. First, the edges between existing nodes and injected nodes $(\mathbf{A_I}, \mathbf{V})$ are generated; Second, the features of injected nodes are optimized. This breakdown can largely reduce the complexity and make GIA applicable to large-scale graphs.

The target model $\mathcal{M}$ could be any graph machine learning models. Following the community convention [19, 25, 26], the focus of this work is on graph neural networks as the target models.

4 GNNS UNDER GRAPH INJECTION ATTACK

In this section, we analyze the behavior of graph neural networks (GNNs) under the general injection attack framework. The analysis results can be used to design effective GIA strategies.

4.1 The Vulnerability of GNNs under GIA

Intuitively, the function of GIA requires the injected nodes to spread (misleading) information over edges in order to influence other (existing) nodes. Which kind of models are vulnerable to such influence? In this section, we investigate the vulnerability of GNN models under GIA.

Definition 1 (Permutation Invariant). *Given* $\mathbf{G} = (\mathbf{A}, \mathbf{F})$*, the graph ML model* $\mathcal{M}$ *is permutation invariant, if for any* $\mathbf{G}' = (\mathbf{A}', \mathbf{F}')$ *with* $\mathbf{G}'$ *as a permutation of* $\mathbf{G}$ *such that* $\forall i \in \{1, 2, ..., N\}, \mathcal{M}(\mathbf{G}')_{\sigma_i} = \mathcal{M}(\mathbf{G})_i$*. Note that* $\mathbf{G}'$ *is a permutation of* $\mathbf{G}$*, if there exists a permutation* $\sigma\colon \{1, ..., N\} \rightarrow \{\sigma_1, ..., \sigma_N\}$ *such that* $\forall i \in \{1, 2, ..., N\}, \mathbf{F}'_{\sigma_i} = \mathbf{F}_i$ *and* $\forall (i, j) \in \{1, 2, ..., N\}^2, \mathbf{A}'_{\sigma_i \sigma_j} = \mathbf{A}_{ij}$.

Definition 2 (Gia-Attackable). *The model* $\mathcal{M}$ *is GIA-attackable, if there exist two graphs* $\mathbf{G_1}$ *and* $\mathbf{G_2}$ *containing the same node i, such that* $\mathbf{G_1}$ *is an induced subgraph of* $\mathbf{G_2}$ *and* $\mathcal{M}(\mathbf{G_1})_i \neq \mathcal{M}(\mathbf{G_2})_i$.

A GIA-attackable graph ML model is a model that an attacker can change its prediction of a certain node by injecting nodes into the original graph. By definition, the index of node i in $\mathbf{G_1}$ and $\mathbf{G_2}$ do not matter for *permutation invariant* models, since permutations can be applied to make it to index 0 in both graphs.

Definition 3 (Structural-Ignorant Model). *The model* $\mathcal{M}$ *is a structural-ignorant model, if* $\forall \mathbf{G}_1 = (\mathbf{A}_1, \mathbf{F}), \mathbf{G}_2 = (\mathbf{A}_2, \mathbf{F})$ *such that* $\forall i \in \{1, 2, ..., N\}, \mathcal{M}(\mathbf{G}_1)_i = \mathcal{M}(\mathbf{G}_2)_i$*, that is,* $\mathcal{M}$ *gives the same predictions for nodes that have the same features* $\mathbf{F}$*. On the contrary,* $\mathcal{M}$ *is non-structural-ignorant, if there exist* $\mathbf{G}_1 = (\mathbf{A}_1, \mathbf{F}), \mathbf{G}_2 = (\mathbf{A}_2, \mathbf{F})$*,* $\mathbf{A}_1 \neq \mathbf{A}_2$*, and* $\mathcal{M}(\mathbf{G}_1)_i \neq \mathcal{M}(\mathbf{G}_2)_i$.

According to this definition, most GNNs are non-structural-ignorant, as they rely on the graph structure $\mathbf{A}$ for node classification instead of only using node features $\mathbf{F}$. We use a lemma to show that non-structural-ignorant models are GIA-attackable, the proof of the lemma is included in the Appendix A.3. According to the lemma, we demonstrate that if a permutation-invariant graph ML model is not structural-ignorant, it is GIA-attackable.

Lemma 4.1 (Non-structural-ignorant Models are GIA-Attackable). *If a model* $\mathcal{M}$ *is non-structural-ignorant and permutation invariant,* $\mathcal{M}$ *is GIA-attackable.*

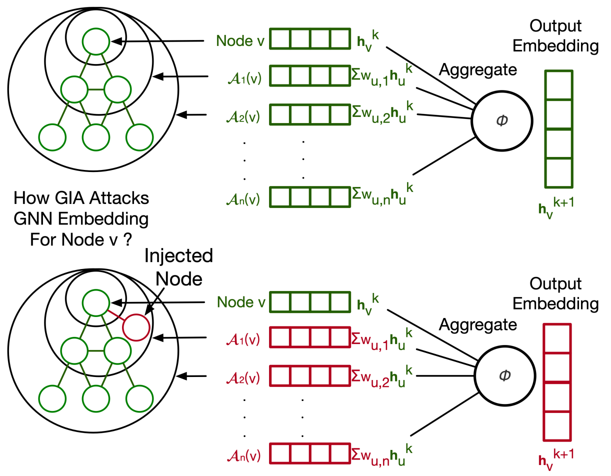


Figure 2: Top: A GNN layer aggregates information from n-hop neighbors of node v. Bottom: GIA perturbs the output embedding through 1-hop node injection.

4.2 Topological Vulnerability of GNN Layers

In order to better design injection attacks to GNNs, we explore the topological vulnerabilities of GNNs. Generally, a GNN layer performed on a node v can be represented as the aggregation process [9, 21]:

$$\mathbf{h}_v^k = \phi(\mathbf{h}_v^{k-1}, \mathbf{f}(\{\mathbf{h}_u^{k-1}\}_{u \in \mathcal{A}(v)})) \tag{5}$$

where $\phi(\cdot)$ and $\mathbf{f}(\cdot)$ are vector-valued functions, $\mathcal{A}(v)$ denotes the neighborhood of node v, and $\mathbf{h}_v^k$ is the vector-formed hidden representation of node v at layer k.

Note that $\mathcal{A}(v)$ include nodes that are directly connected to v and nodes that can be connected to v within certain number of steps. We use $\mathcal{A}_t(v)$ to represent the t-hop neighbors of v, i.e. nodes that can reach v within t steps. We use $\mathbf{f}_t(\cdot)$ as the corresponding aggregation functions. Therefore, Eq. 5 can be further expressed by

$$\begin{aligned}\mathbf{h}_v^k =& \phi(\mathbf{f}_0(\mathbf{h}_v^{k-1}), \mathbf{f}_1(\{\mathbf{h}_u^{k-1}\}_{u \in \mathcal{A}_1(v)}), \mathbf{f}_2(\{\mathbf{h}_u^{k-1}\}_{u \in \mathcal{A}_2(v)}),\\ &..., \mathbf{f}_n(\{\mathbf{h}_u^{k-1}\}_{u \in \mathcal{A}_n(v)}))\end{aligned} \tag{6}$$

Suppose we perturb the graph by injecting nodes so that $\mathcal{A}_t$ changes to $\mathcal{A}'_t$, and $\mathbf{f}_t$ is changed by a comparable small amount, i.e., $\mathbf{f}'_t - \mathbf{f}_t = \Delta \mathbf{f}_t$. The new embedding for v at layer k becomes

$$\begin{aligned}\mathbf{h}_v'^k =& \mathbf{h}_v^k + \frac{\partial \phi}{\partial \mathbf{f}_0}(\mathbf{f}'_0 - \mathbf{f}_0) + \frac{\partial \phi}{\partial \mathbf{f}_1}(\mathbf{f}'_1 - \mathbf{f}_1) + ... + \frac{\partial \phi}{\partial \mathbf{f}_n}(\mathbf{f}'_n - \mathbf{f}_n)\\ &+ O((\Delta \mathbf{f}_0)^2 + (\Delta \mathbf{f}_1)^2 + ... + (\Delta \mathbf{f}_n)^2)\end{aligned} \tag{7}$$

By denoting $\frac{\partial \phi}{\partial \mathbf{f}_t}$ at layer k as $\mathbf{p}_{t,k}$, we have

$$\Delta \mathbf{h}_v^k = \mathbf{h}_v'^k - \mathbf{h}_v^k = \sum_{t=0}^{n} \mathbf{p}_{t,k} \Delta \mathbf{f}_t + o(\sum_t \Delta \mathbf{f}_t) \tag{8}$$

Without loss of generality, we assume that $\mathbf{f}_t$ has the form of weighted average that is widely used in GNNs [14, 20, 24]

$$\mathbf{f}_t(\{\mathbf{h}_u^{k-1}\}_{u \in \mathcal{A}_t(v)}) = \sum_{u \in \mathcal{A}_t(v)} w_{u,t} \mathbf{h}_u^{k-1} \tag{9}$$

where the weight $w_{u,t}$ corresponds to the topological structure within t-hop neighborhood of node v. Therefore, we can exploit the topological vulnerabilities of GNNs under GIA setting by conducting t-hop node injection to construct $\mathcal{A}'_t(v)$ and to perturb the output of $\mathbf{f}_t$

$$\Delta \mathbf{f}_t = \sum_{u \in \mathcal{A}'_t(v)} (\Delta w_{u,t} \mathbf{h}_u^{k-1} + w_{u,t} \Delta \mathbf{h}_u^{k-1}) \tag{10}$$

Figure 2 represents an example of 1-hop node injection. The injected node can affect the embeddings of the t-hop neighborhoods of node v, resulting in final misclassification after the aggregation. The problem now becomes how to harness the topological vulnerabilities of the graph to conduct effective node injection.

5 THE TDGIA FRAMEWORK

We present the Topological Defective Graph Injection Attack (TDGIA) framework for effective attacks on graph neural networks. Its design is based on the vulnerability analysis of GNNs in Section 4. The overall process of TDGIA is illustrated in Figure 1 (b).

Specifically, TDGIA consists of two steps that corresponds to the general GIA process: topological defective edge selection and smooth adversarial optimization. First, we identify important nodes according to the topological properties of the original graph and inject new nodes around them sequentially. Second, to minimize the performance of node classification, we optimize the features of the injected nodes with a smooth loss function.

5.1 Topological Defective Edge Selection

In light of the topological vulnerability of a GNN layer (Cf. Section 4.2), we design an edge selection scheme to generate defective edges between injected nodes and original nodes to attack GNNs.

For a node v, TDGIA can change its t-hop neighbors $\mathcal{A}_t(v)$ to $\mathcal{A}'_t(v)$. For original nodes $u \in \mathcal{A}_t(v)$, their features $\mathbf{h}_u$ remain unchanged. For injected nodes $u \in \mathcal{A}'_t(v) \backslash \mathcal{A}_t(v)$, their features $\mathbf{h}_u$ are initialized by zeros, then $\Delta \mathbf{h}_u = \mathbf{h}'_u$. Then, Eq. 10 can be developed into

$$\Delta \mathbf{f}_t = \sum_{u \in \mathcal{A}_t(v)} \Delta w_{u,t} \mathbf{h}_u^{k-1} + \sum_{u \in \mathcal{A}'_t(v) \backslash \mathcal{A}_t(v)} w_{u,t} \mathbf{h}_u'^{k-1} \tag{11}$$

We start from attacking a single-layer GNN, i.e. $k = 1$. According to Eq. 8, we shall maximize $\sum_{t=0}^{1} \mathbf{p}_{t1} \Delta \mathbf{f}_t$ to maximize $\Delta \mathbf{h}_v^1$. Note that $\Delta \mathbf{f}_0 = \Delta \mathbf{h}_v^0 = \mathbf{0}$ since the original features are unchanged. Thus, we only need to maximize $\Delta \mathbf{f}_1$

$$\Delta \mathbf{f}_1 = \sum_{u \in \mathcal{A}_1(v)} \Delta w_{u,1} \mathbf{h}_u^0 + \sum_{u \in \mathcal{A}'_1(v) \backslash \mathcal{A}_1(v)} w_{u,1} \mathbf{h}_u'^0 \tag{12}$$

During edge selection stage, the features of injected nodes are not yet determined. Thus, our strategy is to first maximize the influence on $\sum w_{u,1}, u \in \mathcal{A}'_t(v)$ and to perturb $\Delta \mathbf{f}_1$ as much as possible.

We start from the common choices of $w_{u,1}$ used in GNNs. Following GCN [14], various types of GNNs [6, 20, 24] use

$$w_{u,1} = \frac{1}{\sqrt{deg(u)deg(v)}}, u \in \mathcal{A}_1(v) \tag{13}$$

while mean-pooling based GNNs like GraphSAGE [9] use

$$w_{u,1} = \frac{1}{deg(v)}, u \in \mathcal{A}_1(v) \tag{14}$$

In TDGIA, when deciding which nodes the injected nodes should be linked to, we use a combination of weights from Eq. 13 and Eq. 14 to scale the topological vulnerability of node v:

$$\lambda_v = k_1 \frac{1}{\sqrt{deg(v)d}} + k_2 \frac{1}{deg(v)} \tag{15}$$

where $deg(v)$ is the degree of the target node v and d is the budget on degree of injected nodes. The higher λ_v is, the more likely a node may be attacked by GIA. In TDGIA, we connect injected nodes to existing nodes with higher λ_v by constructing defective edges.

In Appendix A.2, we theoretically demonstrate that Eq. 10 can be generalized to multi-layer GNNs, thus this topological defective edge selection strategy still works under general GNNs.

5.2 Smooth Adversarial Optimization

Once the topological defective edges of injected nodes have been selected, the next step is to generate features for the injected nodes to advance the effect of the attacks. Specifically, given a model $\mathcal{M}$, an adjacency matrix $\mathbf{A}'$ after node injection, we further optimize the features $\mathbf{F}_I$ of the injected nodes in order to (negatively) influence the model prediction $\mathcal{M}(\mathbf{A}', \mathbf{F}^{'})$. To that end, we design a smooth adversarial feature optimization with a smooth loss function.

Smooth Loss Function. Usually, in adversarial attack, we optimize reversely the loss function used for training a model. For example, we can use the inverse of KL divergence as the attack loss for a node v in target set $\mathcal{T}$

$$\mathcal{L}_v = -D_{KL}(\mathrm{Y}_{\text{pred}} || \mathrm{Y}_{\text{test}}) = \ln(p_{y_{v,\text{pred}} = y_{v,\text{test}}}) = \ln p_v \tag{16}$$

where p_v is for simplicity the probability that $\mathcal{M}$ correctly classifies v. Using this loss may cause gradient explosion, as the derivative

$$\frac{\partial \mathcal{L}_v}{\partial p_v} = \frac{\partial \ln p_v}{\partial p_v} = \frac{1}{p_v} \tag{17}$$

goes to ∞ when $p_v \longrightarrow 0$. To prevent such unstable behavior during optimization, we use a smooth loss function

$$\mathcal{L}_v = \max(r + \ln p_v, 0)^2 \tag{18}$$

where r is a control factor. Therefore the derivative becomes

$$\frac{\partial \mathcal{L}_v}{\partial p_v} = \begin{cases} \frac{2(r + \ln p_v)}{p_v}, & e^{-r} < p_v \leq 1 \\ 0, & 0 \leq p_v \leq e^{-r} \end{cases} \tag{19}$$

where $\frac{\partial \mathcal{L}_v}{\partial p_v} \longrightarrow 0$ when $p_v \longrightarrow 0$, and the optimization becomes stable. Finally, the objective is to find optimal features F_I for injected nodes, which minimize the loss in Eq. 18 for all target nodes:

$$\arg\min_{F_I} \frac{1}{|\mathcal{T}|} \sum_{v \in \mathcal{T}} \max(r + \ln p_v, 0)^2 \tag{20}$$

Smooth Feature Optimization. Under GIA settings, there's a constraint on the range of features of the injected nodes. Otherwise, the defenders can easily filter out injected nodes based on abnormal features. In TDGIA, we simply apply *Clamp* function during optimization process to limit the range of features

$$Clamp(x, min, max) = \begin{cases} min, & x < min \\ x, & min < x < max \\ max, & x > max \end{cases} \tag{21}$$

However, this function may lead to zero gradient. If a feature exceeds the range, it will be stuck at maximal or minimal. To smooth the optimization process of TDGIA, we design a *Smoothmap* function that remaps features onto (min, max) smoothly by using

$$Smoothmap(x, min, max) = \frac{max + min}{2} + \frac{max - min}{2} sin(x). \quad (22)$$

5.3 Overall attack process of TDGIA

In addition to topological defective edge selection and smooth adversarial optimization, we also include the sequential attack and the use of surrogate models in TDGIA.

Sequential Attack. In TDGIA, we adopt the idea of sequential attack [19] and inject nodes in batches. In each batch we add a small number of nodes to the graph, select their edges, and optimize their features. We repeat this process until the injection budget is fulfilled.

Surrogate Model. Under the black-box setting, the attacker has no information about the models being attacked, thus the attack has to be performed on a surrogate model. Specifically, we first train a surrogate model $\mathcal{M}$ using the given training data on the input graph and generate the surrogate labels $\{\hat{y}_v, v \in \mathcal{T}\}$ using $\mathcal{M}$. Then we optimize the TDGIA attack to lower the accuracy of $\mathcal{M}$ for $\{\hat{y}_v, v \in \mathcal{T}\}$. Note that when selecting defective edges, besides λ_v, we also use the correct probability p_v based on the softmax output of $\mathcal{M}$ on node v for its surrogate label $\hat{y}_v$. We then define the defective score μ_v as shown in Algorithm 1.

Complexity. Given a base model $\mathcal{M}$ with complexity T. Usually for GNNs $T = O(ED)$, where E is the number of edges and D being the dimension of input features. For edge selection, we needs to inference $\mathcal{M}$ once to generate p_v, which costs $O(T)$, and computation for λ_v costs $O(E) = o(T)$, so the computation costs $O(T)$. For optimization, suppose Δ_S is the number of epochs and B is the number of batches for sequential injection, the optimization costs $O(\Delta_S BT)$. So the overall complexity for TDGIA is $O(\Delta_S BT)$. In practice, $\Delta_S B$ for TDGIA is usually set to be smaller than the number of epochs for training $\mathcal{M}$, therefore generating attacks using TDGIA costs less time than training $\mathcal{M}$. TDGIA is very scalable and can work for any GNN as base model.

In summary, the attack of TDGIA is to first inject new nodes (and edges) into the original graph and then learn the features for the injected nodes. The injection of new nodes is determined by the topological vulnerabilities of the graph and GNNs. The features are learned via the smooth adversarial optimization. The overall attack process of TDGIA is illustrated in Algorithm 1.

Algorithm 1: The process of Topological Defective Graph Injection Attack (TDGIA).

```
Input: Original graph $G = \{A, F\}$; surrogate model $\mathcal{M}$;set
       of target nodes $\mathcal{T}$;
Output: Attacked graph $G' = (A', F')$;
Parameter: Budget on number of injected nodes $b$; budget
           on degree of each injected node $d$; constraint
           on range of features $\Delta_F$;
/* Initialization */
1  $G' \leftarrow G; V_I \leftarrow 0^{N \times N_I}; A_I \leftarrow 0^{N_I \times N_I}; F_I \leftarrow \mathcal{N}(0, \sigma)^{N_I \times D}$;
/* Sequential injection */
2  while $b > 0$ do
       /* Topological Defective Edge Selection */
3      for $v \in \mathcal{T}$ do
4          Calculate the correct probability $p_v$ using $\mathcal{M}(G')$;
5          Calculate the defective factor $\lambda_v$ using Eq. 15;
6          Calculate the defective score $\mu_v = (\alpha p_v + (1 - \alpha))\lambda_v$;
7      end
8      Set up the number of injected nodes $b_{seq} \leq b$;
9      $V_I \leftarrow$ Connect $b_{seq}$ injected nodes to $b_{seq} \times d$ target
         nodes in $\mathcal{T}$ with the highest defective score $\mu_v$;
       /* Smooth Adversarial Optimization */
10     $F_I \leftarrow$ Optimize the features of injected nodes smoothly
         (Eq. 20) using Clamp (Eq. 21) and Smoothmap (Eq. 22) ;
11     $b \leftarrow b - b_{seq}$, update the budget;
12     $A', F' \leftarrow$ Update $A'$ and $F'$ by $V_I$, $F_I$ using Eq. 2 and 3;
13     $G' \leftarrow (A', F')$;
14 end
15 return $G'$
```

6 EXPERIMENTS

6.1 Basic Settings

Datasets. We conduct our experiments on three large-scale public datasets including 1) KDD-CUP dataset[2], a large-scale citation dataset used in KDD-CUP 2020 *Graph Adversarial Attack & Defense* competition 2) ogbn-arxiv [10], a benchmark citation dataset and 3) Reddit [9], a well-known online forum post dataset[3]. Statistics of these datasets and injection constraints are displayed in Table 2.

Constraints. For each dataset, we set up the budget on the number of injected nodes $b = 500$ and the budget on degree $d = 100$. The feature limit Δ_F is set according to the range of features in the dataset (Cf Table 2). For experiments on KDD-CUP dataset, most submitted defense methods include preprocessing that filters out nodes with degree approaches to 100. Therefore, in our experiments, we apply an artificial limit of 88 to avoid being filtered out. These constraints is applied to both TDGIA and baseline attack methods.

Evaluation Metric. To better evaluate GIA methods, we consider both the performance reduction and the transferability. Our evaluation is mainly based on the weighted average accuracy proposed in KDD-CUP dataset. The metric attaches a weight to each defense model based on its robustness under GIA, i.e. more robust defense gets higher weight. This encourages the adversary to focus on transferability across all defense models, and to design more general attacks. In addition to weighted average accuracy, we also provide the average accuracy among all defense models, and the average accuracy of the Top-3 defense models. The three evaluation metrics are formulated below:

$$s_{avg} = \frac{1}{n} \sum_{i=1}^{n} s_i, \quad (23)$$

[2]https://www.biendata.xyz/competition/kddcup_2020_formal/

[3]A previously-existing dataset originally extracted and obtained by a third party, and hosted by pushshift.io, and downloaded from http://snap.stanford.edu/graphsage/#datasets

Table 2: **Statistics of datasets. We consider only unique undirected edges.**

Dataset	Nodes	Train nodes	Val nodes	Test nodes	Edges	Features	Classes	Feature range	Injection feature range	Injected nodes	Injection degree limit
KDD-CUP	659,574	580,000	29,574	50,000	2,878,577	100	18	-1.74~1.63	-1~ 1	500	100
ogbn-arxiv	169,343	90,941	29,799	48,603	1,157,799	128	40	-1.39~1.64	-1~1	500	100
Reddit	232,965	153,932	23,699	55,334	11,606,919	602	41	-0.27~0.26	-0.25~0.25	500	100

$$s_{\text{top-3}} = \frac{1}{n}\sum_{i=1}^{3} s_i, \tag{24}$$

$$s_{\text{weighted}} = \sum_{i=1}^{n} w_i s_i, \sum_{i=1}^{n} w_i = 1, w_1 \geq w_2 \geq ... \geq w_n. \tag{25}$$

where $s_1, s_2, ...s_n$ are descending accuracy scores of n different defense models against one GIA attack, i.e. $s_1 \geq s_2 ... \geq s_n$. For KDD-CUP dataset, we use the given weights, for ogbn-arxiv and Reddit, we set the weights in a similar way. More reproducibility details are introduced in Appendix A.1.

6.2 Attack & Defense Settings

Baseline Attack Methods. We compare our TDGIA approach with different baselines, including FGSM [18], AFGSM [19], and the SPEIT method [23], the open-source attack method released by the champion team of KDD-CUP 2020. For KDD-CUP dataset we also include the top five attack submissions in addition to the above baselines. Specifically, FGSM and AFGSM are adapted to the GIA settings with black-box and evasion attacks. For FGSM [18], we randomly connect injected nodes to the target nodes, and optimize their features with inverse KL divergence (Eq. 16). AFGSM [19] offers an improvement to FGSM, we also adapt it to our GIA settings. Note that NIPA [17] covered in Table 1 is not scalable enough for the large-scale datasets.

Surrogate Attack Model. GCN [14] is the most fundamental and most widely-used model among all GNN variants. Vanilla GCNs are easy to attack [19, 25]. However, when incorporated with LayerNorm [2], it becomes much more robust. Therefore, it is used by some top-competitors in KDD-CUP and achieved good defense results. In our experiments, we mainly use GCN as the surrogate model to conduct transfer attacks. We use GCNs (with LayerNorm) with 3 hidden layers of dimension 256, 128, 64 respectively. Following the black-box setting, we first train the surrogate GCN model, perform TDGIA and various GIA on it, and transfer the injected nodes to all defense methods.

Baseline Defense Models. For KDD-CUP dataset, it offers 12 best defense submissions (including models and weights), which are considered as defense models. Note that these defense methods are well-formed, which are much more robust than weak methods like raw GCN (without LayerNorm or any other defense mechanism) evaluated in previous works [5, 17, 19, 25, 26]. Most of top attack methods can lower the performance of raw GCN from 68.37% to less than 35%. However, they can hardly reduce the 68.57% weighted average accuracy on these defenses by more than 4%.

For Reddit and ogbn-arxiv datasets, we implement the 7 most representative defense GNN models (also appeared in top KDD-CUP defense submissions), GCN [14] (with LayerNorm), SGCN [20], TAGCN [6], GraphSAGE [9], RobustGCN [24], GIN [21] and [15] as defense models. We train these models on the original graph and fix them for defense evaluation against GIA methods. Details of these models are listed in Appendix A.1.

6.3 Performance of TDGIA

We use GCN(with LayerNorm) as our surrogate attack model for our experiments. We first evaluate the proposed TDGIA on KDD-CUP dataset. Table 3 illustrates the average performance of TDGIA and other GIA methods over 12 best defense submissions at KDD-CUP competition. Different from previous works, TDGIA aims at the common topological vulnerability of GNN layers, which makes it more transferable cross different defense GNN models. As can be seen, TDGIA significantly outperforms all baseline attack methods by a large margin with more than 8% reduction on weighted average accuracy.

We also test the generalization ability of TDGIA on other datasets. As shown in Table 4, when attacking the 7 representative defense GNN models on Reddit and ogbn-arxiv, TDGIA still shows dominant performance on reducing the weighted average accuracy. This suggests that TDGIA can well generalize across different datasets.

To summarize, the experiments demonstrate that TDGIA is an effective injection attack method with promising transferability as well as generalization ability.

Table 3: Performance(%) of different GIA methods on KDD-CUP over 12 best KDD-CUP defense submissions.

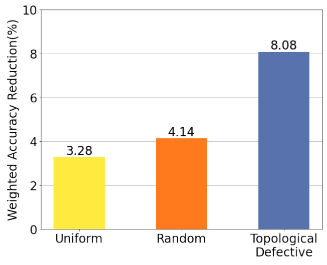
	Attack Method	Average Accuracy	Top-3 Defense	Weighted Average	Reduction
Clean	-	65.54	70.02	68.57	-
KDD-CUP Top-5 Attack Submissions	advers	63.44	68.85	67.09	1.48
	dafts	63.91	68.50	67.02	1.55
	ntt	60.21	68.80	66.27	2.30
	simong	60.02	68.59	66.29	2.28
	u1234	61.18	67.95	64.87	3.70
Baseline Methods	FGSM	59.80	67.44	65.04	3.53
	AFGSM	59.22	67.37	64.74	3.83
	SPEIT	61.89	68.16	66.13	2.44
TDGIA	TDGIA	**55.00**	**64.49**	**60.49**	**8.08**

6.4 Ablation Studies

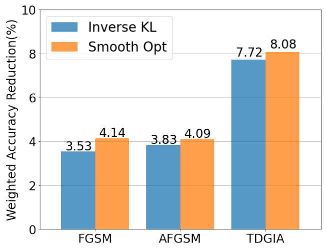
In this section, we analyse in details the performance of TDGIA under different conditions, and TDGIA's transferability when we use different surrogate models to attack different defense models.

Table 4: Performance(%) of different GIA methods on Reddit and ogbn-arxiv over 7 representative defense models.

Dataset	Attack Method	Average Accuracy	Top-3 Defense	Weighted Average	Reduction
Reddit	Clean	94.86	95.94	95.62	-
	FGSM	92.26	94.61	93.80	1.82
	AFGSM	91.46	94.64	93.61	2.01
	SPEIT	93.35	94.27	93.99	1.63
	TDGIA	**86.11**	**88.95**	**88.14**	**7.48**
ogbn-arxiv	Clean	70.86	71.61	71.34	-
	FGSM	66.40	69.57	68.62	2.72
	AFGSM	62.60	69.08	66.96	4.38
	SPEIT	66.93	69.56	68.63	2.71
	TDGIA	**57.00**	**59.23**	**58.53**	**12.81**



(a) Different Edge Selection Methods



(b) Different Optimization Methods

Figure 3: Left:Performance of different edge selection methods based on smooth adversarial optimization. Right: Comparison of smooth adversarial optimization and inverse KL-divergence minimization. Results on KDD-CUP dataset.

Topological Defective Edge Selection. In Section 5.1 we propose a new edge selection method based on topological properties. We analysis the effect of this method by illustrating experimental results of different edge selection methods in Figure 3 (a). "Uniform" method connects injected nodes to targeted nodes uniformly, i.e. each target node receives the same number of links from injected nodes, which is the most common strategy used by KDD-CUP candidates. "Random" method randomly assigns links between target nodes and injected nodes. As illustrated, the topological defective edge selection contributes a lot to attack performance, and almost doubles the reduction on weighted accuracy of defense models.

Smooth Adversarial Optimization. The smooth adversarial optimization, proposed in Section 5.2, also has its own advantages. Figure 3 (b) shows the results of TDGIA and FGSM/AFGSM with/without smooth adversarial optimization. The strategy prevents the issues of gradient explosion and vanishing and does contribute to the attack performance of all three methods.

Transferability across Different Models. We study the influence of different surrogate models on TDGIA. Figure 4 illustrates the transferability of TDGIA across different models. An interesting result is that GCN turns out to be the best surrogate model, i.e. TDGIA applied on GCN can be better transferred to other models.

Figure 5 further offers the visualization of transferability of TDGIA on KDD-CUP and ogbn-arxiv. The heat-map shows that TDGIA

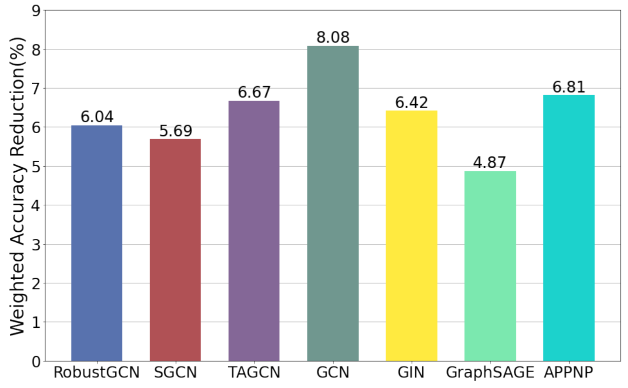


Figure 4: Weighted Accuracy Reduction of TDGIA on KDD-CUP dataset using different surrogate models for attack. GCN yields the best result.

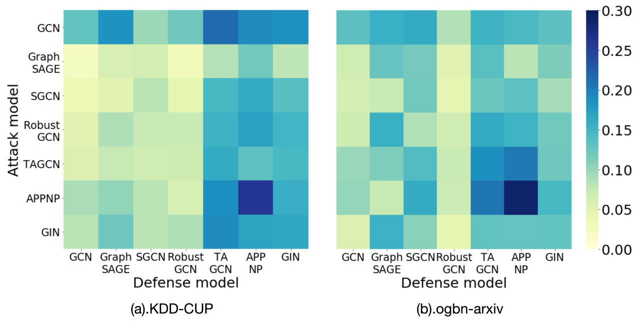
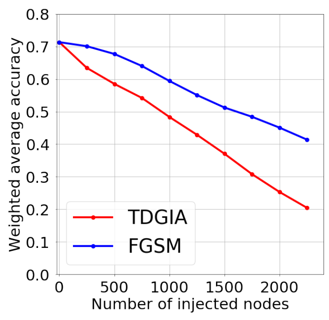


Figure 5: Transferability of TDGIA across different models. Using surrogate models for attack, and evaluate on defense models. Darker color suggests larger performance reduction. Left: KDD-CUP. Right: ogbn-arxiv.

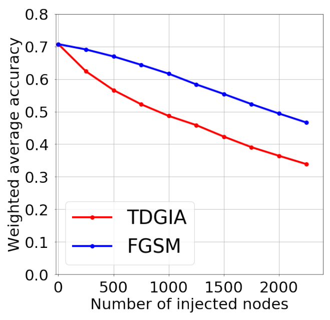
is effective for whatever surrogate model we use, and can be transferred to all defense GNN models listed in this paper, despite that the scale of transferability may vary. Again, we can see GCN yields attacks with better transferability. A probable explanation may be that most of GNN variants are designed based on GCN, making them more similar to GCN. Therefore, TDGIA can be better transferred using GCN as surrogate models. We also notice that RobustGCN is more robust as defense models, as it is intentionally designed to resist adversarial attacks. Still, our TDGIA is able to reduce its performance.

Magnitude of Injection. In previous experiments, we limit the number of injected nodes under 500. Note that this number is only 1% compared to the number of target nodes. To show the power of TDGIA, we investigate the effect of the magnitude of injection. Figure 6 (a) (b) show the attack performance of TDGIA and FGSM on ogbn-arxiv and KDD-CUP datasets. For any magnitude of injection, TDGIA always outperforms FGSM significantly, with a gap for more than 10% reduction on weighted average accuracy.

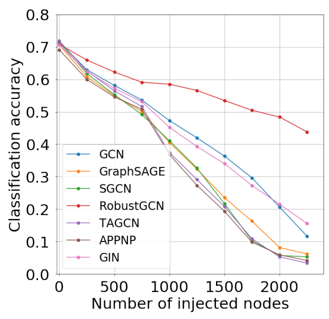
Figure 6 (c) (d) further illustrate the detailed performance of TDGIA on different defense models. When we expand the number of injected nodes, there is a continuous performance drop for all defense models. As the number increased to 2000, the accuracy of several defense models (e.g. GraphSAGE, SGCN, APPNP) even



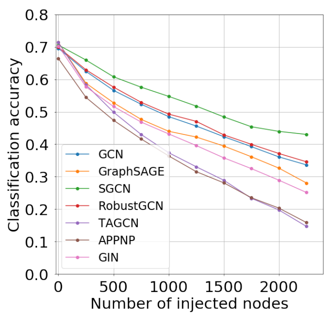
(a) TDGIA vs. FGSM (ogbn-arxiv)



(b) TDGIA vs. FGSM (KDD-CUP)



(c) TDGIA on Defense Models (ogbn-arxiv)



(d) TDGIA on Defense Models (KDD-CUP)

Figure 6: TDGIA under different numbers of injected nodes.

drop less than 10%. However, this number of injected nodes is still less than 5% the size of target nodes, or 1.4% (ogbn-arxiv) / 0.4% (KDD-CUP) the size of the whole graph. The results demonstrate that most GNN models are quite vulnerable towards TDGIA.

7 CONCLUSION

In this work, we explore deeply into the graph injection attack (GIA) problem and present the TDGIA attack method. TDGIA consists of two modules: the topological defective edge selection for injecting nodes and smooth adversarial optimization for generating features of injected nodes. TDGIA achieves the best attack performance in attacking a variety of defense GNN models, compared with various baseline attack methods including the champion solution of KDD-CUP 2020. It is also worth mentioning that with only a few number of injected nodes, TDGIA is able to effectively attack GNNs under the black-box and evasion settings.

In this work, we mainly focus on leveraging the first-level neighborhood on the graph to design the attack strategies. In the future, we would like to involve higher levels of neighborhood information for advanced attacks. Another interesting finding is that among all the GNN variants, using GCN as the surrogate model achieves the best results. Further studies on this observation may deepen our understandings of how different GNN variants work, and thus inspire more effective attack designs.

Acknowledgements. Jie Tang and Tsinghua University are funded by National Natural Science Foundation of China(Key Program, No. 61836013), National Science Foundation for Distinguished Young Scholars (No. 61825602) and Tsinghua-Bosch Joint ML Center. We thank Yang Yang for helpful discussions, Shuwen Liu and Yicheng Zhao for the dataset preparation work on KDD CUP 2020, and all participants of KDD CUP 2020 ML2 Track.

REFERENCES

[1] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML'18*. PMLR, 274–283.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[3] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*. IEEE, 39–57.

[4] Nicholas Carlini and David Wagner. 2018. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops*. IEEE.

[5] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML'18*.

[6] Jian Du, Shanghang Zhang, Guanhang Wu, José MF Moura, and Soummya Kar. 2017. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370* (2017).

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NeurIPS'14*. 2672–2680.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS'17*. 1024–1034.

[10] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).

[11] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[12] Mingjian Jiang, Zhen Li, Shugang Zhang, Shuang Wang, Xiaofeng Wang, Qing Yuan, and Zhiqiang Wei. 2020. Drug–target affinity prediction using graph neural network and contact maps. *RSC Advances* 10, 35 (2020), 20701–20712.

[13] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. 2021. Adversarial Attacks and Defenses on Graphs. *ACM SIGKDD Explorations Newsletter* 22, 2 (2021), 19–34.

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

[15] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[16] J Li, S Ji, T Du, B Li, and T Wang. 2019. TextBugger: Generating Adversarial Text Against Real-world Applications. In *26th Annual Network and Distributed System Security Symposium*.

[17] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Adversarial Attacks on Graph Neural Networks via Node Injections: A Hierarchical Reinforcement Learning Approach. In *WWW'20*. 673–683.

[18] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[19] Jihong Wang, Minnan Luo, Fnu Suya, Jundong Li, Zijiang Yang, and Qinghua Zheng. 2020. Scalable Attack on Graph Data by Injecting Vicious Nodes. *arXiv preprint arXiv:2004.13825* (2020).

[20] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML'19*. PMLR, 6861–6871.

[21] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *ICLR'18*.

[22] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*. 974–983.

[23] Qinkai Zheng, Yixiao Fei, Yanhao Li, Qingmin Liu, Minhao Hu, and Qibo Sun. 2020. *KDD CUP 2020 ML Track 2 Adversarial Attacks and Defense on Academic Graph 1st Place Solution*. https://github.com/Stanislas0/KDD_CUP_2020_MLTrack2_SPEIT.

[24] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *KDD'19*. 1399–1407.

[25] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD'18*. 2847–2856.

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

A APPENDIX

In appendix we follow the citation index in the main article.

A.1 Reproducibility Details

In this section, we introduce the experimental details for TDGIA.

Surrogate Attack Models. Under the black-box setting, we need to train surrogate models for attack. Each model is trained twice using different random seeds. The first one is the surrogate model. The second one is used as defense models. For KDD-CUP, we evaluate directly based on candidate submitted models and parameters. All models are trained for 10000 epochs using Adam, with a learning rate of 0.001 and dropout rate of 0.1. We evaluate the model on the validation set every 20 epochs and select the one with the highest validation accuracy to be the final model.

Detailed Description of Baseline Defense Models. In section 6.2, We only provide detailed introduction to GCN (with LayerNorm). Here we explain in detail about the other defense models.

SGCN [20]. SGCN aims to simplify the GCN structure by removing the activations while improving the aggregation process. The method introduces more aggregation than other methods and thus has a much larger sensitivity area for each node, making the model more robust against tiny local neighborhood perturbations. We use an initial linear transformation that transforms the input into 140 dimensions, and 2 SGC layers with 120 and 100 channels respectively, with $k = 4$ and LayerNorm, a final linear transformation that transfers that into the number of classes.

TAGCN [6]. TAGCN is a GCN variant that combines multiple-level neighborhoods in every single-layer of GCN. This is the method used by SPEIT, the champion of the competition. The model in our experiments has 3 hidden layers, each with 128 channels and with the propagation factor $k = 3$.

GraphSAGE [9]. GraphSAGE represents a type of node-based neighborhood aggregation mechanism, which aggregates direct neighbors on each layer. The aggregation function is free to adjust. Many teams use this framework in their submissions, their submissions vary in aggregation functions. We select a representative GraphSAGE method that aggregates neighborhoods based on L_2-norm of neighborhood features. The model has 4 hidden layers, each with 70 dimensions.

RobustGCN [24]. RobustGCN is a GCN variant designed to counter adversarial attacks on graphs. The model borrows the idea of random perturbation of features from VAE, and tries to encode both the mean and variation of the node representation and keep being robust against small perturbations. We use 3 hidden layers with 150 dimensions each.

GIN [21]. GIN is introduced to maximize representation power of GNNs by aggregating self-connected features and neighborhood features of each node with different weights. Team "Ntt Docomo Labs" uses this method as defensive model. We use 4 hidden layers with 144 dimensions each.

APPNP [15]. APPNP is designed for fast approximation of personalized prediction for graph propagation. Like SGCN, APPNP propagates on graph dozens of times in a different way and therefore more robust to local perturbations. In application we first transform the input with a 2-layer fully-connected network with hidden size 128, then propagate for 10 times.

Attack Parameters. We conduct our attacks using batch-based smooth adversarial optimization. r in Eq. 18 is set to 4. We follow the Algorithm 1. For λ_v in Eq. 15, we take $k_1 = 0.9$ and $k_2 = 0.1$, for α mentioned in Algorithm 1 is set to 0.33. We don't follow exactly the AFGSM description of one-by-one injection, as it costs too much time to optimize on large graphs with hundreds of injected nodes, instead we add nodes in batches, each batch contains nodes equal to 20% of the injection budget, and is optimized under smooth adversarial optimization with Adam optimizer with a learning rate of 1, features are initialized by $N(0, 1)$. Each batch of injected nodes is optimized on surrogate models to lower its prediction accuracy on approximate test labels for 2,000 epochs.

Evaluation Mertric. The w mentioned in Eq. 25 is set to $[0.24, 0.18, 0.12, 0.1, 0, 08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01]$ in the KDD-CUP dataset, which offers a variety of 12 top candidate defense submissions. For evaluation on ogbn-arxiv and Reddit, w is set to $[0.3, 0, 24, 0.18, 0.12, 0.08, 0.05, 0.03]$ for the 7 defense models. In Figure 6 the KDD-CUP evaluation is based on the same models and weights as ogbn-arxiv.

Additional Information on Datasets. In the raw data of the Reddit dataset, unlike other dimensions, dimension 0 and 1 are integers ranging from 1 to 22737. We apply a transformation $f(x) = 0.025 \times \log(x)$ to regularize their range to $[-0.27, 0.26]$ to match the scale of other dimensions. The ogbn-arxiv dataset has 1,166,243 raw links, however 8,444 of them are duplicated and only 1,157,799 are unique bidirectional links. So we take 1,157,799 as the number of links for ogbn-arxiv dataset.

A.2 Generalizing Topological Properties of Single-Layer GNN to Multi-Layer GNNs

In this section, we elaborate the topological properties of GNNs from single-layer to multi-layer. We show in multi-layer GNNs, the perturbation of $\mathbf{h}$ only relies on $\{\Delta w_{u,t} \| t = 1, 2...\}$, therefore the topological defective edge selection in Section 4.2 can be generalized to multi-layer GNNs.

We start from Eq. 10, for a GNN layer,

$$\Delta \mathbf{f}_t = \sum_{u \in \mathcal{A}'_t(v)} (\Delta w_{u,t} \mathbf{h}_u + w_{u,t} \Delta \mathbf{h}_u) \tag{26}$$

Here, $\Delta w_u = w'_u - w_u, \Delta \mathbf{h}_u = \mathbf{h}'_u - \mathbf{h}_u$ if $u \in \mathcal{A}'_t(v)$ and $u \in \mathcal{A}_t(v)$. For $u \in \mathcal{A}'_t(v), u \notin \mathcal{A}_t(v)$, $\Delta w_u = w'_u, \Delta \mathbf{h}_u = \mathbf{h}'_u$. For GIA, $\mathcal{A}_t(v) \subseteq \mathcal{A}'_t(v)$. Recall that $\mathbf{p}_{t,k} = \frac{\partial \mathbf{h}_v^k}{\partial \mathbf{f}_t}$. Then

$$\Delta \mathbf{h}_v^k = \sum_{t=0}^{n} \mathbf{p}_{tk} \sum_{u \in \mathcal{A}'_t(v)} (\Delta w_u \mathbf{h}_u^{k-1} + w_u \Delta \mathbf{h}_u^{k-1}) \tag{27}$$

$$\Delta \mathbf{h}_v^k = \sum_{t} \sum_{u \in \mathcal{A}'_t(v)} (\Delta w_{u,t} \mathbf{p}_{t,k} \mathbf{h}_u^{k-1} + w_{u,t} \mathbf{p}_{tk} \Delta \mathbf{h}_u^{k-1}) \tag{28}$$

This suggests that for single-layer GNNs the perturbation only relies on $\{\Delta w_{u,t} \| t = 1, 2...\}$.

Now let's generalize this result to multi-layer GNNs using induction. Suppose $\Delta \mathbf{h}_v^k$ can be expressed in the following form

$$\Delta \mathbf{h}_v^k = \sum_t \sum_{u\in\mathcal{A}'_t(v)} (\Delta w_{u,t} \sum_{l=1}^{k} \mathbf{p}_{t,l}\mathbf{h}_u^{l-1} + w_{u,t}\mathbf{p}_{t,k}\Delta\mathbf{h}_u^0) \tag{29}$$

For induction, suppose it holds for $k = n$,

$$\begin{aligned}
\Delta \mathbf{h}_v^{n+1} &= \sum_t \sum_{u\in\mathcal{A}'_t(v)} (\Delta w_{u,t}\mathbf{p}_{t,k+1}\mathbf{h}_u^k + w_{u,t}\mathbf{p}_{t,k+1}\Delta\mathbf{h}_u^k) \\
&= \sum_t \sum_{u\in\mathcal{A}'_t(v)} (\Delta w_{u,t}\mathbf{p}_{t,k+1}\mathbf{h}_u^k + \\
&\quad w_{u,t}\mathbf{p}_{t,k+1} \sum_{t'} \sum_{u'\in\mathcal{A}'_{t'}(u)} (\Delta w_{u',t'} \sum_{l=1}^{k-1} \mathbf{p}_{t',l}\mathbf{h}_u^{l-1} + w_{u',t'}\mathbf{p}_{t',k}\Delta\mathbf{h}_u^0)) \\
&= \sum_t \sum_{u\in\mathcal{A}'_t(v)} (\Delta w_{u,t} \sum_{l=1}^{k} \mathbf{p}'_{t,l}\mathbf{h}_u^{l-1} + w_{u,t}\mathbf{p}'_{t,k+1}\Delta\mathbf{h}_u^0)
\end{aligned} \tag{30}$$

where $\mathbf{p}'$ is a function of the previous $\mathbf{p}$. Therefore Eq. 29 holds for $k = n + 1$. Also it obviously holds for $k = 1$, by induction we conclude that Eq. 29 holds for all k. Therefore,

$$\Delta \mathbf{h}_v^n = \sum_t \sum_{u\in\mathcal{A}'_t(v)} (\Delta w_{u,t} \sum_{l=1}^{n} \mathbf{p}_{t,l}\mathbf{h}_u^{l-1} + w_{u,t}\mathbf{p}_{t,k}\Delta\mathbf{h}_u^0) \tag{31}$$

And when we have $w_{u,t} = 0, \forall u \notin \mathcal{G}$ for GIA, assuming $\Delta w_{u,t} << 1$, the function becomes

$$\Delta \mathbf{h}_v^n = \sum_t \sum_{u\in\mathcal{A}'_t(v)} \Delta w_{u,t} \sum_{l=1}^{n} \mathbf{p}_{t,l}\mathbf{h}_u^{l-1} \tag{32}$$

This means the perturbation on multi-layer GNNs also only relies on $\{\Delta w_{u,t} | t = 1, 2...\}$. Therefore, TDGIA can capture the topological weaknesses of them, which is also demonstrated what our extensive experiments on multi-layer GNNs.

A.3 Proof of Lemma 4.1

Proof. Assume model $\mathcal{M}$ is non-structural-igonorant, then there exist $\mathbf{G}_1 = (\mathbf{A}_1, \mathbf{F}), \mathbf{G}_2 = (\mathbf{A}_2, \mathbf{F}), \mathbf{A}_1 \neq \mathbf{A}_2, \mathcal{M}(\mathbf{G}_1) \neq \mathcal{M}(\mathbf{G}_2)$. Permute a common node of $\mathbf{G}_1$ and $\mathbf{G}_2$ to position 0, then

$$A_1 = \begin{bmatrix} 0 & B_1 \\ C_1 & S_1 \end{bmatrix}, A_2 = \begin{bmatrix} 0 & B_2 \\ C_2 & S_2 \end{bmatrix}$$

$$B_1, B_2 \in \mathbb{R}^{1\times(N-1)}, C_1, C_2 \in \mathbb{R}^{(N-1)\times 1}, S_1, S_2 \in \mathbb{R}^{(N-1)\times(N-1)}$$

Consider a case of GIA in which nodes with the same features are injected to $\mathbf{G}_1$ and $\mathbf{G}_2$ in a different way, i.e. $\mathbf{G}_1^* = (\mathbf{A}_1^*, \mathbf{F}^*)$, $\mathbf{G}_2^* = (\mathbf{A}_2^*, \mathbf{F}^*)$ where

$$A_1^* = \begin{bmatrix} 0 & B_1 & B_2 \\ C_1 & S_1 & \mathbf{0} \\ C_2 & \mathbf{0} & S_2 \end{bmatrix}, A_2^* = \begin{bmatrix} 0 & B_2 & B_1 \\ C_2 & S_2 & \mathbf{0} \\ C_1 & \mathbf{0} & S_1 \end{bmatrix}$$

Suppose $\mathcal{M}$ is not GIA-attackable, by Definition 2

$$\mathcal{M}(\mathbf{G}_1^*) = \mathcal{M}(\mathbf{G}_1), \mathcal{M}(\mathbf{G}_2^*) = \mathcal{M}(\mathbf{G}_2) \tag{33}$$

Table 5: Full Performance(%) table on KDD-CUP dataset including 28 competition submissions, baselines methods, and result of GIA using all 7 different surrogate attack models, evaluated on 12 top candidate submitted defenses in KDD-CUP 2020. Best results are bolded.

	Attack Method	Average Accuracy	Top-3 Defense	Weighted Average	Reduction
Clean	-	65.54	70.02	68.57	-
KDD-CUP Attacks	advers	63.44	68.85	67.09	1.48
	dafts	63.91	68.50	67.02	1.55
	deepb	61.44	69.4	67.26	1.31
	dminers	63.76	69.39	67.48	1.09
	fengari	63.78	69.41	67.45	1.12
	grapho	63.75	69.34	67.44	1.13
	msupsu	65.49	69.97	68.52	0.05
	ntt	60.21	68.80	66.27	2.30
	neutri	63.62	69.42	67.42	1.15
	runz	63.96	69.40	67.55	1.02
	speit	61.97	69.49	67.32	1.25
	selina	64.67	69.40	67.79	0.78
	tsail	63.90	69.40	67.55	1.02
	cccn	63.11	69.26	67.28	1.29
	dhorse	63.94	69.33	67.51	1.06
	kaige	63.90	69.41	67.49	1.08
	idvl	63.57	69.42	67.39	1.18
	hhhvjk	65.00	69.38	67.93	0.64
	fashui	63.69	69.42	67.42	1.15
	shengz	63.99	69.40	67.55	1.02
	sc	64.48	69.11	67.41	1.16
	simong	60.02	68.59	66.29	2.28
	tofu	63.87	69.39	67.50	1.07
	yama	64.21	68.77	67.23	1.34
	yaowen	63.94	69.33	67.50	1.07
	tzpppp	65.01	69.38	67.94	0.63
	u1234	61.18	67.95	64.87	3.70
	zhangs	63.73	69.43	67.51	1.06
Baseline Methods	FGSM	59.80	67.44	65.04	3.53
	FGSM (Smooth)	58.45	67.13	64.43	4.14
	AFGSM	59.22	67.37	64.74	3.83
	AFGSM (Smooth)	58.52	67.15	64.48	4.09
	SPEIT	61.89	68.16	66.13	2.44
TDGIA with different surrogate models	RobustGCN	57.24	65.83	62.53	6.04
	sgcn	58.01	66.28	62.88	5.69
	tagcn	58.35	65.82	62.90	5.67
	GCN	**55.00**	**64.49**	**60.49**	**8.08**
	GIN	56.83	65.70	62.15	6.42
	GraphSAGE	59.35	66.43	63.70	4.87
	appnp	55.80	65.93	61.76	6.81

However, since $\mathcal{M}$ is permutation invariant, i.e. $\mathbf{G}_1^*$ and $\mathbf{G}_2^*$ are the same graph under permutation, then

$$\mathcal{M}(\mathbf{G}_1) = \mathcal{M}(\mathbf{G}_1^*) = \mathcal{M}(\mathbf{G}_2^*) = \mathcal{M}(\mathbf{G}_2) \tag{34}$$

which contradicts to the initial assumption that $\mathcal{M}(\mathbf{G}_1) \neq \mathcal{M}(\mathbf{G}_2)$, so $\mathcal{M}$ is GIA-attackable. □