GOttack: Universal Adversarial Attacks on Graph Neural Networks via Graph Orbits Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph Neural Networks (GNNs) have demonstrated superior performance in node classification tasks across diverse set of applications. However, their vulnerability to adversarial attacks, where minor perturbations can mislead model predictions, poses significant challenges. This study introduces "GOttack", a novel adversarial attack framework that exploits the topological structure of graphs to undermine the integrity of GNN predictions systematically. By defining a topology-aware method to manipulate graph orbits, our approach can generate adversarial modifications that are both subtle and effective, posing a severe test to the robustness of GNNs. We evaluate the efficacy of GOttack across multiple prominent GNN architectures, including GCN, GIN, and GraphSAGE, using standard benchmark datasets. Our results show that GOttack not only outperforms existing state-of-the-art adversarial techniques but also completes training in approximately 85% of the time required by the fastest competing model, achieving the highest average misclassification rate in 65 tasks. This work not only sheds light on the susceptibility of GNNs to structured adversarial attacks but also shows that certain topological patterns may play significant role in the underlying robustness of the GNNs.

1 Introduction

Recent advances in Graph Neural Networks (GNNs) have brought significant progress in node classification tasks, utilizing the power of graph topology and node features to generate insightful inferences across various application domains such as social networks [11], bioinformatics [45] and communication systems [14]. Despite their effectiveness, GNNs exhibit inherent vulnerabilities to adversarial attacks; a minor yet strategically designed perturbation in the graph structure or nodal features can deceive the model into erroneous predictions. This susceptibility not only undermines the reliability of GNNs but also poses a grave security risk in critical applications.

Existing approaches predominantly rely on direct node feature manipulation or edge modifications without considering their topological impact. We address this limitation by designing a novel adversarial attack framework that systematically alters the graph topology to induce misclassification errors. Distinct from existing methods, we leverage node connectivity patterns (i.e., orbits) in local graph structures to maximize adversarial efficacy.

Our studies of graph topology yield a surprising result; we have uncovered a universal attack strategy commonly employed by several well-known gradient-based adversarial models. This strategy uses two graph orbits to delineate the resilience of GNNs to adversarial manipulations. Following this discovery, we introduce the GOttack algorithm, an advanced method that identifies and exploits these vulnerable graph orbits. GOttack not only enhances attack misclassification rates but also operates with greater efficiency, reducing the complexity of the attack search associated with such adversarial interventions.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

Through rigorous experiments across three GNN node classification backbones, four adversarial models and five benchmark datasets, we demonstrate that GOttack not only achieves higher misclassification rates compared to state-of-the-art adversarial methods but also maintains a lower computational overhead, making it a potent tool against GNNs.

Our contributions can be summarized as follows:

- We determine a key topological equivalence group among graph nodes, revealing its frequent use in the selection process of gradient-based adversarial models.
- Our findings present a new vulnerability in GNNs related to the orbital characteristics of graphlets.
- Our proposed attack strategy, GOttack, achieves the highest misclassification rate and represents the first scalable attack model suitable for large graphs.

2 Notation and Preliminaries

We consider the task of node classification in a graph, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ represents the set of nodes, $\mathcal{E} \subseteq \{(v, w) \mid v, w \in \mathcal{V}\}$ is the set of edges, and $\mathbf{X} = \{x_0, x_1, \ldots, x_{n-1}\}$ comprises feature vectors such that $x_i \in \mathbb{R}^M$ is the M-dimensional feature vector of node i. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ for the graph $\mathcal{G}$ has elements $\mathbf{A}_{vw} = 1$ if there is an edge e_{vw} connecting nodes v and w, and 0 otherwise.

A subset of nodes $\mathcal{V}_L \subseteq \mathcal{V}$ is labeled, each associated with class labels from the set $\mathcal{C} = \{1, \ldots, c\}$, where y_v denotes the true label of node v in $\mathcal{V}_L$. This setup facilitates examining how shared labels influence edge formation between nodes, an essential aspect of understanding neighbor influence on node classification. Homophily [50] in graphs is traditionally characterized by the similarity between connected node pairs, where nodes are considered similar if they share identical labels. The homophily ratio is constructed based on this premise as follows:

Definition 1 (Homophily Ratio)**.** *Let $\mathcal{G}$ denote the aforementioned graph and $\mathbf{y}$ represent the vector of node labels. The homophily ratio is defined as the proportion of edges connecting nodes with the same labels, formally given by:*

$$h(\mathcal{G}, \{y_v; v \in \mathcal{V}\}) = \frac{1}{|\mathcal{E}|} \sum_{(v,w) \in \mathcal{E}} \mathbb{1}(y_v = y_w),$$

where $\mathbb{1}(\cdot)$ denotes the indicator function.

A graph is considered highly homophilous if the homophily ratio $h(\cdot)$ is large, typically within the range $0.5 \leq h(\cdot) \leq 1$. Conversely, a low homophily ratio indicates a heterophilous graph.

Node Classification. The goal of node classification is to infer a function $g : \mathcal{V} \to \mathbb{P}(\mathcal{C})$, that assigns a probability distribution over the class set $\mathcal{C}$ to each unlabeled node v, where $\hat{y}_v$ is the predicted class for node v, identified as the class with the highest probability in $g(v)$. This setup, characterized as transductive learning, implies that the model predictions are based on instances both seen and used during training.

The Graph Convolutional Network (GCN), as introduced by Kipf and Welling [21], provides a foundational model for understanding and analyzing the vulnerabilities exposed by our proposed attack model. GCN employs a message-passing technique that utilizes the features of neighbouring nodes, making it susceptible to adversarial manipulations that can alter node connections and lead to misclassifications. As a result, this section will define our attack using the GCN operations, which are detailed in Appendix 9.1.

Adversarial Attack. Adversarial attacks on graph data aim to subtly perturb graph structures or node features, causing GCN to misclassify specific nodes. This entails creating a new graph $\mathcal{G}' = (\mathbf{A}', \mathbf{X}')$ from the original $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, with changes to $\mathbf{A}$ (i.e., structural attacks) or $\mathbf{X}$ (i.e., feature attacks). In an attack, a subset of nodes $v \in \mathcal{V}_T \subseteq \mathcal{V}$ are targeted to have the GCN misclassify their labels $\hat{y}_v \neq y_v$ within a specified budget Δ as follows:

$$\sum_{u} \sum_{f} \left| \mathbf{X}_{uf} - \mathbf{X}'_{uf} \right| + \sum_{u<w} \left| \mathbf{A}_{uw} - \mathbf{A}'_{uw} \right| \leq \Delta \tag{1}$$

The node v may be directly affected (i.e., $u = v$), or influence attacks can impact any other node within the graph (i.e., $u \neq v$). The attacks take various forms and occur during different phases. i) Poisoning attacks occur during training time, aiming to compromise the model by manipulating the training dataset. Evasion attacks occur at test time, attempting to generate deceptive samples that evade detection by a trained model. ii) In targeted attacks, the objective is to misclassify specific target nodes $\mathcal{V}_T$, while in non-targeted attacks, the goal is to reduce the overall accuracy of the model.

3 Related Work

Recent interest has grown in adversarial attacks on graph neural networks. These attacks demonstrate how minor changes to input features or graph structure can alter network outputs, often causing incorrect classifications. We begin with an overview of gradient-based attacks and then discuss non-gradient-based methods (refer to Table 31 for a complete classification).

Gradient-based attacks. Gradient-based attacks on graph neural networks exploit the gradients of the model to perturb node features or graph structure, aiming to mislead the network into making incorrect predictions. Zügner et al. [52] proposed Nettack, which marked the inaugural exploration of gradient attacks on attributed graphs, revealing significant accuracy declines even with minor alterations. Another novel approach by Xu et al. significantly reduced classification performance by causing a small number of edge perturbations [39]. Similarly, Li et al. [24] introduced the SGA framework to target nodes by using a smaller subgraph around the target node and leveraging gradient information for attack optimization. Fast Gradient Attack (FGA) used gradient information from GCNs and outperformed baseline methods by efficiently disturbing network embedding with minimal link rewiring [8]. To describe the robustness of deep learning models for graph-based tasks, Zügner et al. [53] introduced training-time attacks using meta-gradients to perturb graph structures, which effectively renders the model near-useless for production use, all without direct access to the target classifier. Our approach uses a gradient-based attack as well, however we reduce the amount of costly gradient computations.

Non-gradient-based attacks. Sun et al. [32] introduced a novel node injection poisoning attack which used hierarchical Q-learning to optimize the injection process. Likewise, Chang et al. [7] proposed the GF-Attack for conducting black-box adversarial attacks on graph embedding models without access to labels. Hussain et al. [17] proposed Structack, which makes use of structural centrality and similarity insights to efficiently lower GNN costs. Similarly, Zou et al. [51] proposed the topological defective graph injection attack, where adversaries inject adversarial nodes into existing graphs rather than modifying links or node attributes. Zhang et al. [43] proposed membership inference attacks targeting edges, also known as link-stealing attacks, which used customized attacks by introducing a group-based attack paradigm that is suited to various groups of edges. Mu et al. [29] proposed the attack as an optimization problem to minimize perturbations to the graph structure, with a particular emphasis on the difficult hard-label black-box attack scenario.

The approach we propose in this paper differs from all the above-mentioned proposals in that none have attempted to identify equivalence groups for graph nodes based on graph orbits to minimize the search space in discrete optimization. Furthermore, our proposed solution introduces a highly effective and efficient algorithm for universal attacks on node classification models, demonstrating significantly faster performance compared to existing methods (e.g., attack training in approximately 85% of the time required by *the fastest competing model*).

4 Methodology

We propose the GOttack framework to execute adversarial attacks on node classification GNNs while minimizing changes to the graph's structure. Figure 1 illustrates the overview of the complete GOttack research workflow.

Challenge. A significant attack challenge in our task is the substantial time complexity, as structural attacks on the underlying graph data might require up to $O(2^{|\mathcal{V}| \times |\mathcal{V}|})$ steps for finding the optimal set of edges to remove or add. This scale of complexity necessitates the development of efficient methods.

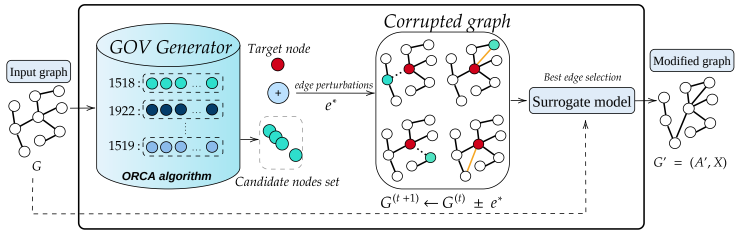


Figure 1: Complete research workflow diagram of GOttack.

GOttack Philosophy. GOttack draws inspiration from the Mapper philosophy of Topological Data Analysis (TDA), as described by Singh et al. [30]. This philosophy posits that data has shape and finding the shape may allow us to build better models. In our case, we focus on identifying and grouping nodes on the graph such that graph topology can allow us to select which nodes and edges to attack in GNNs.

GOttack Solution. GOttack utilizes and advances concepts from group theory [4] to identify node equivalence classes [2] on a graph, which guide our decisions on which edges to add or remove. This approach strategically groups nodes according to their positions on the graph in potential attack strategies, thereby enhancing both the precision and time-efficiency of our adversarial interventions.

4.1 Attack Model

We aim to target a specific node $v \in \mathcal{V}$ in a **targeted, structural, direct evasion attack** to alter its predicted class. As the prediction of v depends not only on its individual attributes but also on the characteristics of neighbouring nodes $\mathcal{N}(v)$ within the graph, we are focused on perturbations to the initial graph $\mathcal{G}$ with the condition: $\exists w, \mathcal{A}'_{vw} \neq \mathcal{A}_{vw}$ where $w \in \mathcal{V}$. However, to prevent the attacker from completely modifying the graph, we impose a constraint on the total number of allowable changes, controlled by a specified budget Δ: $\sum_{w \in \mathcal{V}} |\mathcal{A}_{vw} - \mathcal{A}'_{vw}| \leq \Delta$.

Problem Statement. Consider an initial graph $\mathcal{G} = (\mathcal{A}, \mathcal{X})$, a target node v, and its true class y_v. Our goal is to modify the graph's structure such that the classification of v changes from y_v to $y_{v'}$, thereby maximizing the difference from its original classification. The proposed attacks can be mathematically formulated as a bi-level optimization problem:

$$\arg \max_{(\mathcal{A}', \mathcal{X}) \in \mathcal{G}'} \max_{y_{v'} \neq y_v} \ln Z^*_{v, y_{v'}} - \ln Z^*_{v, y_v} \tag{2}$$

where $\mathbf{Z}^* = f_{\theta^*}(\mathcal{A}', \mathcal{X})$ and $\theta^* = \arg\min_\theta \mathcal{L}(\theta; \mathcal{A}', \mathcal{X})$ subject to the budget constraint. Specifically, we aim to find a modified graph $\mathcal{G}' = (\mathcal{A}', \mathcal{X})$ in which the target node v is assigned a label $y_{v'}$ that maximizes the difference from its original label y_v in terms of probability scores.

4.2 Equivalence Classes in Structural Attacks

We utilize an attack strategy based on node equivalence groups to guide our decisions regarding the addition and deletion of edges. This section begins by defining graphlets, which are instrumental in identifying node equivalence groups. We then discuss orbits, representing the specific positions a node can assume within a graphlet to facilitate effective grouping. As a last step, we compute Graph Orbit Vectors from all graphlets to develop a multi-orbit based attack strategy.

Definition 2 (Graphlet [22])**.** *A graphlet $\mathcal{G}_{gp}$ within a larger graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ is a connected induced subgraph $\mathcal{G}s' = (\mathcal{V}', \mathcal{E}', \mathcal{X}')$, where $\mathcal{V}' \subseteq \mathcal{V}$, and $\mathcal{E}'$ includes all edges $e_{uv} \in \mathcal{E}$ with both u and v in $\mathcal{V}'$, and $|\mathcal{V}'|$ typically equals 5 (as defined in Appendix Figure 6).*

There are 30 distinct graphlets of 5-nodes (see Appendix Figure 6 for the shapes). For instance, consider the path $u \rightarrow x \rightarrow v \rightarrow k \rightarrow y$ in Figure 2, which forms a graphlet with $|\mathcal{V}'| = 5$.

Orbits are defined by automorphisms of the graphlet; an automorphism σ of a graphlet $\mathcal{G}_{gp}$ satisfies $\sigma \cdot \mathcal{G}_{gp} = \mathcal{G}_{gp}$. Nodes v and w in $\mathcal{V}$ are similar if there exists an automorphism σ such that $\sigma(v) = w$. The orbit of a node v, denoted by $\mathrm{Orb}(\mathcal{G}_{gp}, v)$, is the set of all nodes $w \in \mathcal{V}$ that can be mapped onto v by some automorphism of the graphlet:

Definition 3 (Orbit [2])**.** $Orb(\mathcal{G}_{gp}, v) = \{w \in \mathcal{V} \mid \sigma \in Aut(\mathcal{G}_{gp}) : \sigma(v) = w\}$

where $\text{Aut}(\mathcal{G}_{gp})$ is the group of automorphisms of $\mathcal{G}_{gp}$. Each orbit is denoted by Orb_j, where j is a unique identifier for each orbit within a specific graphlet. A node v *touches* an orbit Orb_j if v is part of an induced subgraph in the graph and v belongs to Orb_j. By extension, a node may appear in multiple graphlets and hence occupy multiple orbits. For example, in Figure 2, node x appears in graphlets comprising of nodesets $\{x, v, k, l, w\}$ and $\{x, v, k, y, l\}$ and so on. In these two specified graphlets, x appears in orbits 15, and 18; overall the 30 graphlets create 73 distinct orbits (see Appendix Figure 6 for the orbit positions).

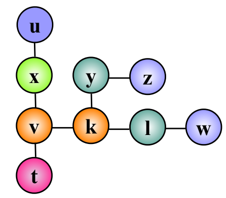


Figure 2: A toy graph where shared node colors imply similar orbit counts. Nodes u, z and w have 15 and 18 orbits respectively.

Graph Orbit Vector. We propose a Graph Orbit Vector (GOV) as a numerical representation of a node's participation across different orbits in a graph. Let $\mathbf{GOV_v} \in \mathbb{Z}^n_{\geq 0}$ be an $n = 73$-dimensional vector. We compute GOV by mapping the frequency or presence of a node in specific orbits, where each element of the vector corresponds to an orbit identified by a unique identifier Orb_j. This vector is computed based on the node's presence in various graphlets, reflecting the structural roles the node assumes in the network. Specifically, the vector element for Orb_j is incremented each time a node v appears in orbit Orb_j of any graphlet where Orb_j is an induced subgraph and v is a part of Orb_j. Thus, the Graph Orbit Vector provides a comprehensive profile of a node's topological embedding within the graph, capturing its involvement in different graphlets.

We note that orbit discovery on graphlets is completed for the entire graph as a pre-processing step, hence does not require recomputations for each target node (See Figure 1 for the Graph Orbit Vector computations overview).

4.3 GOttack: Graph Structure Poisoning via Orbit Learning

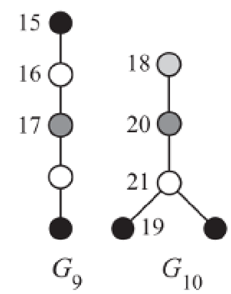


Figure 3: Graphlets where orbits 15 and 18 are defined.

In our exploration of graph topologies, we uncover a distinctive feature shaped by orbits 15 and 18: nodes touching these orbits appear in peninsula-like subgraphs (such as the one formed in Figure 2 by nodes $\{u, x, v, k, t\}$). These orbits (see Figure 3) are characteristic of being three or four edges away from another endpoint in the graphlet, and serve as critical indicators of topological peripheries within a graph. This geometric arrangement lends a substantial foundation to our subsequent analyses and strategic manipulations within adversarial attacks on graph data, as discussed in the following hypotheses and experimental sections.

We start by stating our topological observation as influenced by the Mapper philosophy: 1) **Orbit Proxy.** Under the homophily assumption, node classification posits that graph neighbors are similar to a node in label. Consequently, nodes in more distant positions, as can be identified by orbits, are less similar. 2) **Periphery Orbits.** Orbits 15 and 18 indicate topological periphery in a graph and provide a useful proxy for identifying distant nodes that differ in labels.

We claim that the path distance within the graph encodes a notion of remoteness, which in turn yields minimal information about a node's label. This indicates that physical proximity within the network strongly influences the predictive accuracy regarding node labels. In our experiments (Section 5), we also provide empirical evidence demonstrating the utility of the Orbit Proxy in heterophilic graph cases, albeit in a weaker form.

In Theorem 1, we formally state that nodes located in orbits 15 and 18, due to their peripheral placement, are particularly effective for establishing paths to remote parts of the graph. This has significant implications for the design of network protocols and algorithms that rely on efficient data traversal and retrieval mechanisms.

Theorem 1 (Remote Connection Candidates)**.** *Let $H(v, w)$ denote the expected random walk hitting time from node v to node w in $\mathcal{G}$. For any target node $v \in V$, nodes in orbits 15 and 18 are the most effective candidates for establishing paths to the most remote parts of $\mathcal{G}$, due to their longer expected hitting times $H(v, w)$ compared to other nodes not in these orbits.*

Due to space limitations, the proof is given in Appendix 8.

The Periphery Orbits observation forms the backbone of our attack strategy; we identify nodes of periphery orbits (i.e., 15 and 18) and affect the adjacency matrix (i.e., add or remove edges to these nodes) accordingly to confuse a GNN to misclassify a target. Consider the target node v in Figure 2. Our hypothesis posits that creating an edge from v (or any other node) to either of the (15 and 18) nodes u, z or w will yield the highest misclassification error in GCN. The selection among periphery nodes is carried out using a gradient-based method as we detail below in surrogate loss.

Orbits $(15, 18)$**.** We utilize two ordered orbit categories based on the largest and second-largest orbit count values. The largest orbit count value is identified using $Orb^v_{\text{max}} = \max(\text{GOV}_v)$, and the second-largest orbit count value is defined by $Orb^v_{\text{sec}} = \max(\{j \in \text{GOV}_v \mid j < \max(\text{GOV}_v)\})$. Consequently, each node is assigned to an orbit category denoted as $Orb_{\text{cat}} = Orb_{\text{max}} \| Orb_{\text{sec}}$. It is crucial to note that $Orb_{\text{max}} \| Orb_{\text{sec}}$ is treated as equivalent to $Orb_{\text{sec}} \| Orb_{\text{max}}$.

Example 1. *Consider a node v in graph $\mathcal{G}$. Although GOttack works with $k = 5$-node graphlets, for simplicity, this example employs three-node graphlets ($k = 3$) for orbit counting, yielding a 4-dimensional vector to store all possible orbits. Suppose the orbit count vector for node v is GOV$_v$ $= [4, 15, 11, 12]$. The task is to identify the largest and second-largest orbit count values in GOV$_v$. The largest orbit count value is 15, denoted by $Orb^v_{max} = 01$, and the second-largest orbit count value is 12, denoted by $Orb^v_{sec} = 03$. Thus, node v is categorized into the orbit category 0103, denoted as Orb^v_{0103}.*

Surrogate loss. Our objective is to maximize the discrepancy in log-probabilities for the target node v within a specified budget Δ. The log-probabilities can be simplified to $\hat{\mathcal{A}}^2\mathcal{X}\mathcal{W}$. We linearize the model by replacing the nonlinearity $\sigma(.)$ with a simple linear activation function. Therefore, from Eq. 4, $Z' = \text{softmax}(\hat{A}\hat{A}XW^1W^2) = \text{softmax}(\hat{A}^2XW)$. Therefore, the surrogate loss function, $\mathcal{L}_s(\mathcal{A}, \mathcal{X}; \mathcal{W}, v)$, is designed to optimize the following objective: $\underset{(A',X)\in\mathcal{G}'}{\arg\max}\, \mathcal{L}_s(\mathcal{A}', \mathcal{X}; \mathcal{W}, v)$, where, the surrogate loss function $\mathcal{L}_s$ is defined as $\mathcal{L}_s(\mathcal{A}', \mathcal{X}; \mathcal{W}, v) = \max_{w \neq z}[\hat{A}^2XW]_{v,z} - [\hat{A}^2XW]_{v,w}$. This function aims to solve the maximum loss over a set of permissible changes in $\mathcal{A}$ of $\mathcal{G}$.

Structure poisoning. We compute a candidate node set called the orbit category, denoted as Orb_{cat}, consisting only of allowable elements (v, u) where the edge changes from 0 to 1 (i.e., adding an edge) or vice versa. Specifically, for a given target node v, we create a candidate set such that $u \in \mathcal{V}$ where $Orb^u_{\text{max}} = 15$ or 18, and $Orb^u_{\text{sec}} = 15$ or 18. Among the candidate edge changes, we select the one that yields the highest surrogate loss. However, to compute the surrogate loss score, we first need to determine the class prediction of the target node v after adding or removing an edge (u, v). Here, we are optimizing the loss score with respect to $\mathcal{A}$, the term $\mathcal{X}\mathcal{W}$ is constant. The log-probabilities of node v are then given by $g(v) = [\hat{\mathcal{A}}^2]_v \cdot C$, where $[\hat{\mathcal{A}}^2]_v$ denotes a row vector and C is the constant term $(\mathcal{X}\mathcal{W})$. Thus, we only need to inspect how this row vector changes to determine the optimal edge manipulation. Following the insight developed by Zügner [53], we can derive an incremental update, so there is no need to recompute the updated $[\hat{\mathcal{A}}^2]_v$ from scratch.

5 Experiments

This section presents the experimental evaluation we carried out to show the effectiveness of the proposed approach. We attempt to answer the following questions: i) *How effective is the GOttack approach in terms of misclassification rate compared with existing state-of-the-art approaches?* ii) *How efficient is the proposed model in terms of computation time compared with the existing models?*.

Datasets. We conduct experiments on five widely used node classification datasets, the statistics of which are provided in Table 1. Cora [41], Citeseer [41] and Pubmed [41] datasets are citation networks with undirected edges and binary features where nodes are publications and edges are citation links. In the Polblogs [1] dataset nodes are political blogs and edges are links between them. In the BlogCatalog [33] dataset, nodes' attributes are constructed by keywords, which are generated by users as a short

Table 1: Dataset statistics and homophily ratios.

Dataset	Hom.	Nodes	Edges	Features	Labels
Cora [41]	0.81	2,485	5,069	1,433	7
Citeseer [41]	0.74	2,110	3,668	3,703	6
Polblogs [1]	0.91	1,222	16,714	1,490	2
Pubmed [41]	0.81	19,717	44,325	500	3
BlogCatalog [33]	0.40	5,196	171,743	8,189	6

description of their blogs. We split the network into labeled (20%) and unlabeled nodes (80%). We further split the labelled nodes in equal parts *training* and *validation* sets to train the surrogate model. We have used the ORCA algorithm [15] for the orbit discovery process on these datasets.

Experimental Setup. We have conducted the experiments under the transductive, semi-supervised learning setting. We have used the GCN [21], GIN [40], and GraphSAGE [13] models as the backbone node classifier GNN in our adversarial attacks. We provide the implementation of GOttack at `https://anonymous.4open.science/r/GOttack/`.

Table 2: Misclassification rate (in $\%$) ($\uparrow$) of target nodes in five datasets where three backbone GNNs (GCN, GIN and GraphSAGE) are attacked in node classification with budget $\Delta = 1$.

	Cora			Citeseer			Polblogs			BlogCatalog			Pubmed		
Method	GSAGE	GCN	GIN	GSAGE	GCN	GIN	GSAGE	GCN	GIN	GSAGE	GCN	GIN	GSAGE	GCN	GIN
Random	55	22	43	<u>64</u>	34	54	<u>31</u>	34	12	<u>51</u>	12	<u>63</u>	47	15	50
Nettack	58	34	<u>46</u>	**66**	<u>46</u>	<u>57</u>	29	<u>38</u>	13	50	20	**65**	<u>52</u>	<u>50</u>	47
FGA	54	32	40	60	31	44	22	31	14	46	10	61	42	32	<u>52</u>
SGA	**61**	<u>41</u>	**57**	60	41	<u>57</u>	**35**	37	**35**	<u>51</u>	**24**	61	30	**57**	47
GOttack (ours)	<u>59</u>	**41**	37	61	**46**	**57**	29	**41**	<u>15</u>	**52**	<u>22</u>	<u>63</u>	**55**	**57**	**52**

We average over five different random initializations/splits, where for each, we follow these steps. Initially, we train the surrogate model on the labeled data. From the test set, among all nodes correctly classified, we select: (i) the 10 nodes with the highest margin of classification, indicating clear correctness, (ii) the 10 nodes with the lowest margin (still correctly classified), and (iii) 20 additional nodes randomly chosen. These selected nodes will be the targets for the attacks. All results presented here are computed by us in the same settings.

We compare GOttack against four recent graph adversarial attack frameworks: Nettack [52], FGAttack (FGA)[8] and SGAttack (SGA) [24] as well as a Random (dummy) baseline (see Section 9.2 for descriptions). We report the misclassification rate, which is the percentage of nodes that were incorrectly classified by the model in relation to the total number of nodes being classified.

In our experiments, we utilized the PyG (PyTorch Geometric) library, which employs PyTorch as the backend for implementing GNN models. Additionally, we used the PyTorch adversarial library DeepRobust [25] for robustness evaluations. The experiments were conducted using Python (Version 3.6) and PyTorch (Version 1.10.2). The computational environment was an off-the-shelf computer running Windows 11 (Version 22H2), equipped with a 3.20 GHz Intel(R) Core(TM) i7-8700 processor and 16 GB of RAM.

Parameters setting. Our models were trained using the Adam [10] optimization algorithm, employing a fixed learning rate of 0.01. Training sessions were conducted over a span of 200 epochs. Throughout the training regimen, we employed the *softmax* activation function.

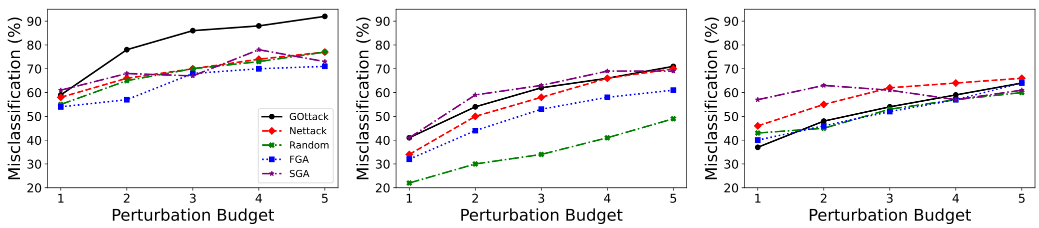
5.1 GOttack Misclassification Results

Table 2 shows the misclassification rate (i.e., 1 - accuracy) achieved with a single edge perturbation (addition or deletion). OOR indicates run times exceeding 3 days or missing results due to memory issues. GOttack yields higher misclassification rates in 8 out of 13 attack settings. SGAttack yields 4 best attacks and Nettack yields the best attack on the GraphSAGE model for the Citeseer dataset. Most importantly, GOttack provides an attack strategy on the GCN model for every dataset.

In Figure 4 we gradually increase the attack budget and represent the corresponding misclassification rates on the Cora dataset. We observe that for GraphSAGE GOttack yields better misclassification whereas in the GCN model GOttack performance is close to the SGAttack. Particularly, by using the GraphSAGE model, the proposed approach achieves the maximum 92% misclassification score for 5 perturbations. Likewise, for the GCN model, the proposed approach achieves a 71% misclassification score, which outperforms the three SOTA models.

Table 3: Comparison of orbit-based node selection in sequential Nettack phases.

Dataset	Orbits	% of nodes	% in 1^{st} Attack	% in 2^{nd} Attack
Cora ($h = 0.81$)	1518	24%	77%	71.1%
	1519	14.41%	5.7 %	14.29%
	1819	11.59%	10 %	12.5%
Citeseer ($h = 0.74$)	1518	21.99%	51.6%	61.29%
	1519	21.18%	12.5 %	15%
	1819	11.56%	3.29 %	0%
Polblogs ($h = 0.91$)	1518	9.41%	97.5%	60%
	1519	2.29%	0%	0%
	1819	12.93%	2.5 %	27.5%
Pubmed ($h = 0.81$)	1518	20.14%	25%	10%
	1519	23.07%	32.5 %	52.5%
	1618	2.24%	22.5 %	10%
BlogCatalog ($h = 0.40$)	1518	3.25%	2.5%	22.5%
	1519	61.57%	62.5 %	37.5%
	1922	19.77%	35 %	40%



(a) GraphSAGE model (b) GCN model (c) GIN model

Figure 4: Budgeted attack results on the Cora dataset.

Extending the analysis of the relationship between increasing budgets and misclassification rates to all datasets, we observe the following behavior (see Section 11 for complete results): **GOttack achieves the highest misclassification rate in** 36 **out of** 65 **tasks**. The numbers are 20 for SGAttack, 11 for Nettack, and 1 for FGA (ties are counted extra). **GOttack achieves the highest average rate of** 0.58 **over all budgets and datasets compared to the second best model of FGA with** 0.57 (Appendix Table 7). Furthermore, as we show in Table 4, GOttack is a computationally more efficient and scalable attack strategy that can be applied to large graphs (in Pubmed, GOttack takes 85% of the Nettack time). For example, GOttack takes less than 30 mins to create attack candidates for the largest dataset BlogCatalog. Nettack and GOttack both offer scalability for large graphs, but they utilize different approaches. Nettack relies on degree distribution to select candidates in each iteration, whereas GOttack employs orbits. This choice is strategic: although the orbit discovery in GOttack incurs additional costs, it effectively narrows down the candidate set to approximately 23% of all nodes (see Appendix Figure 8), justifying the initial expense.

Time Complexity. Orbit discovery is the main cost of GOttack. The total time complexity for computing all orbits for all nodes is given by $O(|E| \times d + |V| \times d^4)$, where $O(|V| \times d^4)$ represents the time required to enumerate complete five-node graphlets, and d represents the maximum degree in the graph. GOttack provides a scalable attack. For example, orbit discovery on CORA takes only 0.17 seconds.

5.2 Insights from GOttack Results

GOttack strategically selects nodes on the graph topology. This selection criterion prompts several thoughts and questions, which we address below:

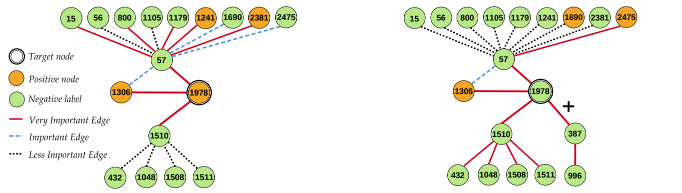
The periphery definition. A visual inspection of Figure 6 shows that more orbits, such as 19, 39, and 27, could fit the periphery definition. The primary reason for not considering these orbits is their relative scarcity in all the graph datasets. For example, the correlation matrix of node orbits from the Cora dataset in Appendix Figure 8 shows that most nodes predominantly feature orbits within the 1518 and 1922 groups. Furthermore, as we show in the extended results with these orbits in Appendix Section11, attacks based on 1819, 1519 and 1922 yield less powerful attacks.

Higher order orbits. GOttack uses two orbits: 1518. The decision against using > 2 orbits is influenced by the fact that nodes typically do not touch more orbits. For the single orbit case, our experiments met larger time complexity, as a larger pool of nodes was considered, yet the efficacy of attacks remained similar. These findings suggest a unique efficiency in utilizing orbits 15 and 18, which seem to foster a universally effective attack vector.

Gradient-based models target 1518 **nodes.** An interesting result of our studies is the discovery

Table 4: End-to-end time costs in seconds (↓). See Appendix Table 27 for stds. GOttack scales to large graphs.

	GOttack	Nettack	FGA	SGA
Cora	66.69	82.62	86.57	**57.49**
Citeseer	**75.57**	92.56	128.52	98.48
Polblogs	123.08	143.81	**93.60**	94.35
BlogCatalog	1735.16	2003.69	**1298.8**	$> 24h$
Pubmed	**1347.90**	1582.35	$2.84h$	$> 24h$



(a) Computation graph before attack (b) Computation graph after attack

Figure 5: The computation graph for the targeted node 1978 from the CORA datasets, as identified by GNNExplainer [26]. The edge $(1978, 387)$ is added during the successful attack. Edge importances change considerably after the attack and negative class gains importance due to the newly added nodes.

that gradient-based models predominantly target 1518 nodes in homophily datasets, as shown for Nettack in Table 3 and other models in Appendix Tables 5 and 6. For example, Table 3 shows that 97.5% of the initial attacks involve 1518 nodes in the highly homophilous Polblogs, while only 9.41% of the nodes are 1518 nodes. Although the numbers remain high in the heterophilous BlogCatalog dataset (2.5% and 22.5% in $\Delta = 1, 2$ attacks), they are comparatively lower than those observed in homophilous contexts. This pattern underscores the strategic importance of selecting orbit 1518 nodes in network attacks.

Node, Homophily, Distance and Subgraph-based Explanations for GOttack. We conducted a comprehensive analysis to understand how GOttack causes node misclassifications. Initially, we focused on node-centric metrics to determine the positional changes of the target node within the graph following the attack. As detailed in Appendix Table 23, the target nodes' clustering coefficient, degree, betweenness, and closeness centralities did not show a significant difference compared to those influenced by other attacks. Secondly, we examined whether the attack brought in nodes with different labels into the computation graph of the target node (see Appendix Table 25). We found that after the attack, both similar and differently labeled nodes increased by 0.92 and 3.36 in average, respectively. However, these values are not as pronounced as those seen in other attacks, suggesting that the induced label diversity change by GOttack is not substantial.

Next, we computed the shortest path distances from the target nodes to nodes with similar and different labels in the entire graph (see Appendix Table 24). This analysis provided empirical proof supporting our Theorem 1, which stated that the 1518 strategy more significantly reduces the distance to nodes of different labels (-0.03) than to those of similar labels (-0.02). This behavior, which indicates a targeted modification in network dynamics, was not observed with other orbits, highlighting the distinct impact of the GOttack strategy.

Lastly, we turned to GNN explainers (see Appendix Section 12.1) which factor in the subgraph effects and node features in perturbations which were overlooked in previous analyses. Figure 5 shows an example where misclassifications are due to changes in the importance of existing edges within the computation graph, which offers evidence that explanations specific to nodes or edges alone are insufficient to adequately explain an attack. However, the explainers do not concur on the specific explanation (see Appendix Figure 9b). Hence, we leave the study of subgraph patterns to future work.

Limitations. Our methodology does not incorporate node features in the attack strategy, which potentially limits its effectiveness since node features often play a crucial role in the vulnerability and defense mechanisms of GNNs. Similarly, GOttack does not consider directed and weighted edges in its attacks, which may exclude useful information, such as the amounts in financial networks.

6 Conclusion

We have identified a crucial equivalence group for graph nodes based on graphlet orbits and demonstrated how gradient-based attack models predominantly utilize this group in their attacks. By revealing this novel vulnerability linked to the orbital structures within graphlets, our work both exposes the susceptibility of GNNs to orbit-based attacks and advances the development of efficient attack models. The GOttack algorithm, born out of these insights, not only improves misclassification rates but does so in a scalable way, making a strong case for its use in reinforcing the security and integrity of GNNs across various applications.

References

[1] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

[2] Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

[3] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018.

[4] Oleg Vladimirovič Bogopolskij. *Introduction to group theory*, volume 6. European Mathematical Society, 2008.

[5] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *International Conference on Machine Learning*, pages 695–704. PMLR, 2019.

[6] Raffaella Burioni and Davide Cassi. Random walks on graphs: ideas, techniques and results. *Journal of Physics A: Mathematical and General*, 38(8):R45, 2005.

[7] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Xin Wang, Wenwu Zhu, and Junzhou Huang. Adversarial attack framework on graph embedding models with limited knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4499–4513, 2022.

[8] Jinyin Chen, Yangyang Wu, Xuanheng Xu, Yixian Chen, Haibin Zheng, and Qi Xuan. Fast gradient attack on network embedding. *arXiv preprint arXiv:1809.02797*, 2018.

[9] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *International conference on machine learning*, pages 1115–1124. PMLR, 2018.

[10] Jimmy Ba Diederik P Kingma. Adam: A method for stochastic optimization. (2014). In *arXiv preprint arXiv:1412.6980*, 2014.

[11] Wenqi Fan, Yao Ma, Qing Li, Jianping Wang, Guoyong Cai, Jiliang Tang, and Dawei Yin. A graph neural network framework for social recommendations. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2033–2047, 2020.

[12] Jian Feng and Shaojian Chen. Link prediction based on orbit counting and graph auto-encoder. *IEEE Access*, 8:226773–226783, 2020.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[14] Shiwen He, Shaowen Xiong, Yeyu Ou, Jian Zhang, Jiaheng Wang, Yongming Huang, and Yaoxue Zhang. An overview on the application of graph neural networks in wireless networks. *IEEE Open Journal of the Communications Society*, 2:2547–2565, 2021.

[15] Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014.

[16] Tomaz Hocevar and Janez Demsar. A combinatorial approach to graphlet counting. *Bioinform.*, 30(4):559–565, 2014.

[17] Hussain Hussain, Tomislav Duricic, Elisabeth Lex, Denis Helic, Markus Strohmaier, and Roman Kern. Structack: Structure-based adversarial attacks on graph neural networks. In *Proceedings of the 32nd ACM Conference on Hypertext and Social Media*, pages 111–120, 2021.

[18] Di Jin, Bingdao Feng, Siqi Guo, Xiaobao Wang, Jianguo Wei, and Zhen Wang. Local-global defense against unsupervised adversarial attacks on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8105–8113, 2023.

[19] Jeff D Kahn, Nathan Linial, Noam Nisan, and Michael E Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2:121–128, 1989.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[22] Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.

[23] Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discret. Math.*, 27(2):892–909, 2013.

[24] Jintang Li, Tao Xie, Liang Chen, Fenfang Xie, Xiangnan He, and Zibin Zheng. Adversarial attack on large scale graph. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):82–95, 2021.

[25] Yaxin Li, Wei Jin, Han Xu, and Jiliang Tang. Deeprobust: A pytorch library for adversarial attacks and defenses. *arXiv preprint arXiv:2005.06149*, 2020.

[26] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.

[27] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. Towards more practical adversarial attacks on graph neural networks. *Advances in neural information processing systems*, 33:4756–4766, 2020.

[28] Ine Melckenbeeck, Pieter Audenaert, Didier Colle, and Mario Pickavet. Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. *Bioinform.*, 34(8):1372–1380, 2018.

[29] Jiaming Mu, Binghui Wang, Qi Li, Kun Sun, Mingwei Xu, and Zhuotao Liu. A hard label black-box adversarial attack against graph neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 108–125, 2021.

[30] Gurjeet Singh, Facundo Mémoli, Gunnar E Carlsson, et al. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *PBG@ Eurographics*, 2:091–100, 2007.

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

[32] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Non-target-specific node injection attacks on graph neural networks: A hierarchical reinforcement learning approach. In *Proc. WWW*, volume 3, 2020.

[33] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826, 2009.

[34] Shuchang Tao, Qi Cao, Huawei Shen, Junjie Huang, Yunfan Wu, and Xueqi Cheng. Single node injection attack against graph neural networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1794–1803, 2021.

[35] Shuchang Tao, Qi Cao, Huawei Shen, Yunfan Wu, Liang Hou, Fei Sun, and Xueqi Cheng. Adversarial camouflage for node injection attack on graphs. *Information Sciences*, 649:119611, 2023.

[36] Marcin Waniek, Tomasz P Michalak, Michael J Wooldridge, and Talal Rahwan. Hiding individuals and communities in a social network. *Nature Human Behaviour*, 2(2):139–147, 2018.

[37] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

[38] Yihan Wu, Aleksandar Bojchevski, and Heng Huang. Adversarial weight perturbation improves generalization in graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10417–10425, 2023.

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

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

[41] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[42] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[43] He Zhang, Bang Wu, Shuo Wang, Xiangwen Yang, Minhui Xue, Shirui Pan, and Xingliang Yuan. Demystifying uneven vulnerability of link stealing attacks against graph neural networks. In *International Conference on Machine Learning*, pages 41737–41752. PMLR, 2023.

[44] Xiang Zhang and Marinka Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. *Advances in neural information processing systems*, 33:9263–9275, 2020.

[45] Xiao-Meng Zhang, Li Liang, Lin Liu, and Ming-Jing Tang. Graph neural networks and their current applications in bioinformatics. *Frontiers in genetics*, 12:690049, 2021.

[46] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Trans. Knowl. Data Eng.*, 34(1):249–270, 2022.

[47] He Zhao, Zhiwei Zeng, Yongwei Wang, Deheng Ye, and Chunyan Miao. Hgattack: Transferable heterogeneous graph adversarial attack. *arXiv preprint arXiv:2401.09945*, 2024.

[48] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[49] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. Robust graph convolutional networks against adversarial attacks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1399–1407, 2019.

[50] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems*, 33:7793–7804, 2020.

[51] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. Tdgia: Effective injection attacks on graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2461–2471, 2021.

[52] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2847–2856, 2018.

[53] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[54] Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning, 2024.

7 Broader Impact

The proposed methodology makes an important step toward addressing the major existing roadblock in graph neural networks: the vulnerability of the GNN models. These models are widely used across various domains, from social networks to bioinformatics. Undoubtedly, the application of GOttack will have a substantial positive impact in a broad range of applications such as reinforcing the security and integrity of GNNs across various applications.

However, the critical negative impact of the proposed methodology is associated with our current inability to incorporate node features in the attack strategy, potentially excluding valuable information, such as transaction amounts in financial networks. This is a fundamental question that needs to be addressed in the future.

Appendix

Contents

8 Random Walks to Orbits 15 and 18

We will start by defining hitting time as used in random walks over graphs [6].

The expected hitting time of a random walk starting from node v and reaching node w is

$$H(v, w) = \mathbb{E}\left[\min\{t \in \mathbb{N} \setminus \{0\} : X_t = w\} \mid X_0 = v\right].$$

By definition, the first hitting time of a node on itself is typically defined as zero, i.e., $H(v, v) = 0$.

Theorem 1 (Remote Connection Candidates)**.** *Let $H(v, w)$ denote the expected hitting time from node v to node w in $\mathcal{G}$. For any node $v \in V$, nodes in orbits 15 and 18 are the most effective candidates for establishing paths to the most remote parts of $\mathcal{G}$, due to their longer expected hitting times $H(v, w)$ compared to other nodes not in these orbits.*

Proof: It is known that the hitting time $H(v, w)$ is influenced by the structural configuration of the graph and the position of the nodes within it. Kahn et al. have proved that for any regular graph $\mathcal{G}$ = ($\mathcal{V}$, $\mathcal{E}$), the maximum hitting time $H(v, w)$ is bounded by $O(n^2)$ [19], where n is the number of nodes. The expected time of the first hit from v to w is affected by the structural properties of the graph. Consider a node w appearing in orbits 15, 18, or both. By the definition of graphlets $\mathcal{G}_9$ and $\mathcal{G}_{10}$, we have i) $\deg(w) = 1$ when connected to a single graphlet and ii) the first hitting time of w is influenced by the configuration of nodes adjacent to w. As a result, $H(v, w)$ is greater than $H(v, z)$ for all $z \in N(w)$. Extending this, on the shortest path from v to w, $H(v, w)$ is the maximum hitting time among all paths from v to any node in $\mathcal{V}$.

Target node v can be at the center or away from the center.

Center case. Due to their peripheral placement in graphlets, nodes in orbits 15 and 18 are further away from central nodes or densely connected regions of the graph; their hitting times are expected to approach the upper bound of n^2 due to their increased distance from other nodes in $\mathcal{G}$.

Periphery case. The target node v may not necessarily be at the center of the graph. However, by the definition of orbits 15 and 18, there is at least one node in these orbits (perhaps the other end of the same graphlet) that has the longest distance to the target node v. This further increases the hitting time, as the random walk must navigate through central nodes and potentially longer paths to reach these peripheral nodes. $\square$

9 Graph Neural Network

Graph Neural Networks are pivotal in learning node embeddings by capturing node features and their local network neighbourhoods. These embeddings encapsulate the essential characteristics of the nodes into condensed representations by leveraging both the graph structure and feature information from neighbouring nodes. Such embeddings have practical applications across various domains, which are detailed further in related work section 3.

9.1 Backbone Models

In our evaluation of various models, we incorporated baseline models utilizing Graph Neural Networks (GNNs). In this section, we will elucidate the rationale behind each baseline model utilized in our study.

The Graph Convolutional Network (GCN): as introduced by Kipf and Welling [21], provides a foundational model for understanding and analyzing the vulnerabilities exposed by our proposed attack model. GCN employs a message-passing technique that utilizes the features of neighbouring nodes, making it susceptible to adversarial manipulations that can alter node connections and lead to misclassifications. Here, we provide a detailed overview of the GCN architecture, particularly focusing on the structure of its graph convolutional layer (i.e., hidden layer $\mathbf{H}^{(l+1)}$):

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right) \tag{3}$$

In this formulation, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ represents the adjacency matrix of the undirected graph augmented with self-loops, and $\mathbf{I}_N$ is the identity matrix. The matrix $\mathbf{W}^{(l)}$ denotes the trainable weight matrix for layer l, optimized during backpropagation. $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ defines the degree matrix, and σ represents a non-linear activation function.

Setting $\mathbf{H}^{(0)} = \mathbf{X}$ (i.e., the initial node features), the GCN model with l layers computes node classifications as follows:

$$\mathbf{Z} = f(\mathbf{A}, \mathbf{X}) = \text{softmax}(\mathbf{A}\mathbf{X}\Theta) = \text{softmax}(\hat{\mathbf{A}}\sigma(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^1)\mathbf{W}^2) \tag{4}$$

Here, $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ acts as the renormalized adjacency matrix, $\mathbf{X}$ is the feature matrix, and Θ includes the set of parameters (e.g., $\mathbf{W}^1$, $\mathbf{W}^2$) to be learned. The matrix $\mathbf{Z} \in \mathbb{R}^{n \times c}$ represents the probabilities of $C = \{c_i\}$ for each node $v \in V$, with each row indicating the likelihood of each class label c for a node.

GCNs operate under both inductive and transductive settings. We focus on transductive classification, where all node connections and features are accessible during the training phase. For such tasks, the softmax function normalizes the final output matrix $\mathbf{Z}$, and the cross-entropy loss $L = -\sum_{v \in \mathcal{V}_{\text{tr}}} \log \mathbf{Z}_{v,y_v}$ is calculated, comparing the predicted probabilities to the true labels, where y_v represents the true class of node v. Weight updates are performed using gradient descent optimization algorithms, such as Adam [20].

The Graph Isomorphism Network(GIN): is another type of GNN designed to respect graph isomorphisms. It produces the same embedding for isomorphic graphs. Although the learning process is similar to the GCN, it uses a different aggregation function. The following equation [40] shows the calculation of the hidden layer $\mathbf{H}^{(l+1)}$. where, $\epsilon^{(l)}$ is a trainable parameter and $\mathbf{MLP}^{(l)}$ is a multi layer perception.

$$\mathbf{H}^{(l+1)} = \sigma\left((1 + \epsilon^{(l)}) \cdot \mathbf{MLP}^{(l)}(\mathbf{H}^{(l)})\right) \tag{5}$$

GraphSAGE: is a type of GNN which also gathers information from the neighboring nodes like GCN but in a slightly difference way. The following equation [13] shows the aggregation of node feature $\mathbf{X}$ using a sampling strategy, where, $\mathbf{AGG}$ is an aggregation function such as mean or max pooling.

$$\mathbf{H}_v^{(l+1)} = \mathbf{AGG}\left(\{\mathbf{H}_u^{(l)}, \forall u \in \mathcal{N}(v)\}\right) \tag{6}$$

9.2 Baseline Models

In this subsection, we present a brief idea of the existing state-of-the-art adversarial techniques that we have considered as comparable baseline methods.

Nettack [52]: is a targeted attack method to enforce misclassification on the target nodes using edge and feature perturbations, which can handle both direct and influence attacks.

FGA attack [8]: is a targeted attack method to enforce misclassification on the target nodes using graph perturbations by generating adversarial graph networks based on the gradient information of GCN.

SGAttack [24]: is a targeted attack method to enforce misclassification on the target nodes using features or edges perturbations through a multi-stage attack framework, which needs only a much smaller subgraph.

Random attack: it randomly selects non-adjacent node pairs and introduces fake edges or removing existing edges between them.

10 Further analysis of graphlets and orbits

In this section, we present an overview of graphlets and their topological properties, followed by the orbit 1518, which is the topological node embedding. Finally, we discuss the hierarchy of orbits based on relation algebra.

Table 5: Comparison of orbit-based node selection in sequential FGA phases.

Dataset	Orbits	% of nodes	% in 1^{st} Attack	% in 2^{nd} Attack
Cora	1518	24%	60%	45%
	1519	14.41%	10 %	20%
	1819	11.59%	15 %	2.5%
Citeseer	1518	21.99%	20%	15%
	1519	21.18%	32.5 %	20%
	1819	11.56%	15 %	5%
Polblogs	1518	9.41%	37.5%	37.5%
	1519	2.29%	0%	0%
	1819	12.93%	62.5 %	0%

Table 6: Comparison of orbit-based node selection in sequential SGA phases.

Dataset	Orbits	% of nodes	% in 1^{st} Attack	% in 2^{nd} Attack
Cora	1518	24%	51%	47%
	1519	14.41%	12 %	13%
	1819	11.59%	6 %	7%
Citeseer	1518	21.99%	37%	32%
	1519	21.18%	27 %	26%
	1819	11.56%	19 %	23%
Polblogs	1518	9.41%	52%	46%
	1519	2.29%	8%	8%
	1819	12.93%	7 %	10%

10.1 Topological properties of graphlets and orbits

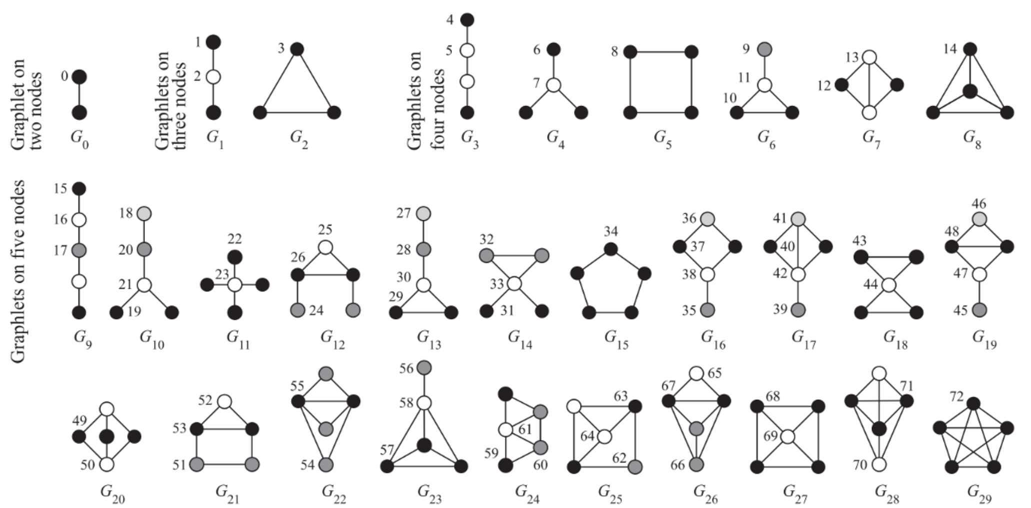


Figure 6: Graphlets with two to five nodes with the automorphism orbits of each graphlet [15].

Graphlets are small, connected, non-isomorphic induced subgraphs that represent topological patterns of interconnection between k nodes in a graph [12]. Figure 6 illustrates all graphlets with two to five nodes, including 9 different graphlets with 2 to 4 nodes and up to 30 graphlets, ranging from $\mathcal{G}_0$ to $\mathcal{G}_{29}$, with 5 nodes. However, the structural properties of a network can be represented by the frequency of graphlet appearances within the network. The orbits define the unique characteristics of the nodes within a graphlet, express different connection modes between nodes, and contain abundant high-order structural information [12]. For instance, consider the graphlet $\mathcal{G}_{11}$ from the set of five-node graphlets. It can be observed that $\mathcal{G}_{11}$ is a star graph where the node labeled with orbit 23 is the central node, while the remaining nodes, labeled with orbit 22, are leaf nodes. The topological context of a node can be determined by counting the orbits of that node.

Orbit counting is computationally expensive because the number of orbits in a graph grows exponentially with the size of the original graph. However, many advanced algorithms have been developed to mitigate the complexity of computing graphlets and orbits [22, 16, 28, 23]. Consequently, numerous studies leverage the topological insights provided by graphlet and orbit counting across various domains. For example, Feng et al. [12] used orbits counting as high-order structural features of nodes to learn efficient node representations, which were then utilized to enhance link prediction tasks.

10.2 Additional analysis of 1518 orbit nodes

Connecting the 1518 orbit node with the target node $v \in \mathcal{V}_T \subseteq \mathcal{V}$ has been demonstrated to significantly impact the prediction accuracy of GNNs f_θ on the node v. This impact is also reflected in the effectiveness of various attack methods, such as FGA and SGA, which frequently select the 1518 node for graph manipulations (e.g., adding or removing edges with v). Table 5 shows that in the Cora dataset, up to 60% and 45% of the nodes manipulated by FGA in the first and second attacks, respectively. Likewise, the node labeled as 1518 is consistently the primary target of SGA in both the first and second attack scenarios across all datasets (ref. Table 6). For instance, in the Citeseer dataset, SGA manipulates nodes in the first and second attacks at rates of 37% and 32%, respectively. Similarly, in the Polblogs dataset, these percentages are 52% and 46% for the first and second attacks.

10.3 Orbit hierarchy

Based on the analysis of the impact of GOttack discussed in 5.1, and experimental results, we have concluded that the 1518 orbit is crucial for attacking the GNNs. This raises the question: *what happens if the* 1518 *orbit node does not exist in the network?* To address this, we have developed an orbit hierarchy (or orbit transition), as shown in Figure 7, based on relational algebra. From this hierarchy, we can see that if the 15 and 18 orbit nodes are absent, the attack model can instead select nodes from orbits 4 and 6, respectively. Similarly, if nodes from orbits 4 and 6 are not present, nodes from orbit 1 can be chosen.

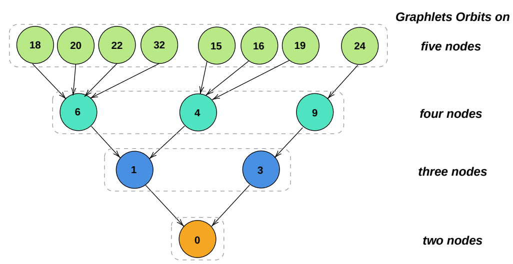


Figure 7: Orbit hierarchy.

To better understand how the orbit transition approach works, let us consider the following example.

Example 2. *Consider a* 5*-node graphlet* $\mathcal{G}_{10}$*, where orbits* 18*,* 19*,* 20 *and* 21 *are present (see Figure 6). If orbit* 18 *is removed from* $\mathcal{G}_{10}$*, then it becomes the* 4*-node graphlet* $\mathcal{G}_4$*. Suppose* $\mathcal{G}_{10} = (\mathcal{V}, \mathcal{E})$ *with* $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5\}$ *and edges* $\mathcal{E}$*, where node* $v \in \mathcal{V}$ *belongs to one of the orbits* Orb^v_{18}*,* Orb^v_{19}*,* Orb^v_{20} *and* Orb^v_{21}*. If the removal operation* ρ *removes orbit node* $Orb^{v_1}_{18}$*, resulting in* $\mathcal{G}_4 = (\mathcal{V}', \mathcal{E}')$*, where* $\mathcal{V}' = \mathcal{V} - \{v_1\}$*. Therefore, by using the projection operations (*π*) the transition of orbits can be written as:* $\pi_{Orb^v_{21}}(\mathcal{G}_{10}) \rightarrow Orb^v_7$ *and* $\pi_{Orb^v_{19}, Orb^v_{20}}(\mathcal{G}_{10}) \rightarrow Orb^v_6$*, indicating that orbit* Orb^v_{21} *transitions to orbit* Orb^v_7*, and orbit* Orb^v_{19} *and* Orb^v_{20} *now belong to orbit* Orb^v_6*.*

11 Experimental Results on all Datasets

In this section, we present results of our experiments. In Table 7, we demonstrate the success of our GOttack strategy across five different datasets, with the results averaged over these datasets. Detailed results for each individual dataset are provided in the subsequent Tables 8 to 22. We present the results of various attack strategies, including GOttack and its variants such as 1819, 1519, and 1922. The results are presented as the mean and standard deviation computed from five independent runs.

Table 7: Summary of Attack Results averaged over five datasets. OOR tasks have been excluded.

Budget	**1**	**2**	**3**	**4**	**5**
Random	0.36	0.44	0.47	0.51	0.54
Nettack	0.43	0.51	0.56	0.61	0.61
GOttack	<u>0.44</u>	<u>0.54</u>	**0.61**	**0.65**	**0.68**
1819 orbit attack	0.35	0.42	0.46	0.47	0.50
1519 orbit attack	0.37	0.41	0.44	0.45	0.46
1922 orbit attack	0.35	0.40	0.41	0.44	0.45
FGA	0.37	0.47	0.52	0.53	0.56
SGA	**0.47**	**0.57**	<u>0.58</u>	<u>0.62</u>	<u>0.64</u>

11.1 Attack Results on GCN

We present the performance of various adversarial attack techniques on GCN. Tables 8 to 12 display the misclassification rates for various datasets with 1 to 5 perturbed edges, respectively. In summary,

Table 8: Misclassification rate (↑) on Cora with budget $\Delta = 1$ to 5: GOttack achieves performance in 2 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.22 ± 0.049	0.3 ± 0.078	0.34 ± 0.049	0.41 ± 0.058	0.49 ± 0.068
Nettack	0.34 ± 0.06	0.5 ± 0.068	0.58 ± 0.057	0.66 ± 0.029	0.7 ± 0.011
GOttack	0.41 ± 0.052	0.54 ± 0.049	0.62 ± 0.033	0.66 ± 0.042	$\underline{0.71 \pm 0.052}$
1819 orbit attack	0.32 ± 0.063	0.45 ± 0.073	0.54 ± 0.076	0.6 ± 0.051	0.65 ± 0.025
1519 orbit attack	0.37 ± 0.029	0.47 ± 0.08	0.57 ± 0.062	0.62 ± 0.074	0.64 ± 0.072
1922 orbit attack	0.34 ± 0.078	0.43 ± 0.074	0.5 ± 0.075	0.57 ± 0.08	0.58 ± 0.069
FGA	0.32 ± 0.057	0.44 ± 0.08	0.53 ± 0.033	0.58 ± 0.048	0.61 ± 0.042
SGA	0.41 ± 0.058	$\underline{0.59 \pm 0.05}$	$\underline{0.63 \pm 0.07}$	$\underline{0.69 \pm 0.02}$	0.69 ± 0.06
UGBA	$\underline{0.57 \pm 0.011}$	0.49 ± 0.14	0.34 ± 0.06	0.45 ± 0.16	0.3 ± 0.07
GRBCD	$\mathbf{0.64 \pm 0.029}$	$\mathbf{0.73 \pm 0.027}$	$\mathbf{0.85 \pm 0.025}$	$\mathbf{0.89 \pm 0.042}$	$\mathbf{0.95 \pm 0.064}$

Table 9: Misclassification rate (↑) on Citeseer with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 5 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.34 ± 0.057	0.41 ± 0.058	0.5 ± 0.074	0.52 ± 0.073	0.57 ± 0.031
Nettack	0.46 ± 0.045	0.61 ± 0.038	0.68 ± 0.027	0.74 ± 0.021	0.76 ± 0.014
GOttack	$\underline{0.46 \pm 0.034}$	0.63 ± 0.037	$\underline{0.72 \pm 0.054}$	$\underline{0.76 \pm 0.063}$	$\underline{0.78 \pm 0.042}$
1819 orbit attack	0.44 ± 0.058	0.56 ± 0.038	0.67 ± 0.045	0.71 ± 0.029	0.74 ± 0.033
1519 orbit attack	0.45 ± 0.05	$\underline{0.63 \pm 0.031}$	0.68 ± 0.018	0.73 ± 0.037	0.76 ± 0.021
1922 orbit attack	0.4 ± 0.033	0.55 ± 0.041	0.67 ± 0.033	0.7 ± 0.033	0.72 ± 0.041
FGA	0.31 ± 0.107	0.48 ± 0.04	0.62 ± 0.068	0.64 ± 0.045	0.68 ± 0.069
SGA	0.41 ± 0.06	0.6 ± 0.05	0.63 ± 0.068	0.69 ± 0.022	0.69 ± 0.057
GRBCD	$\mathbf{0.56 \pm 0.045}$	$\mathbf{0.765 \pm 0.058}$	$\mathbf{0.835 \pm 0.052}$	$\mathbf{0.85 \pm 0.073}$	$\mathbf{0.91 \pm 0.06}$

Table 10: Misclassification rate (↑) on Polblogs with budget $\Delta = 1$ to 5: GOttack achieves performance in 1 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.34 ± 0.021	0.43 ± 0.078	0.46 ± 0.074	0.48 ± 0.087	0.51 ± 0.076
Nettack	0.38 ± 0.04	0.43 ± 0.058	0.46 ± 0.063	0.5 ± 0.082	0.51 ± 0.072
GOttack	$\mathbf{0.41 \pm 0.086}$	0.46 ± 0.08	0.51 ± 0.089	0.52 ± 0.089	0.55 ± 0.077
1819 orbit attack	0.26 ± 0.193	0.35 ± 0.1	0.36 ± 0.183	0.41 ± 0.152	0.46 ± 0.089
1519 orbit attack	0.3 ± 0.198	0.3 ± 0.203	0.3 ± 0.164	0.32 ± 0.179	0.3 ± 0.178
1922 orbit attack	0.22 ± 0.184	0.25 ± 0.149	0.25 ± 0.156	0.26 ± 0.08	0.26 ± 0.038
FGA	0.31 ± 0.098	0.4 ± 0.078	0.45 ± 0.097	0.45 ± 0.089	0.48 ± 0.087
SGA	0.37 ± 0.075	$\mathbf{0.56 \pm 0.051}$	$\mathbf{0.56 \pm 0.037}$	$\mathbf{0.57 \pm 0.068}$	$\mathbf{0.65 \pm 0.02}$

Table 11: Misclassification rate (↑) on BlogCatalog with budget $\Delta = 1$ to 5: GOttack achieves performance in 2 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.12 ± 0.092	0.17 ± 0.071	0.2 ± 0.097	0.25 ± 0.085	0.3 ± 0.082
Nettack	0.2 ± 0.027	0.25 ± 0.085	$\mathbf{0.37 \pm 0.072}$	$\mathbf{0.4 \pm 0.077}$	$\mathbf{0.45 \pm 0.065}$
GOttack	0.22 ± 0.040	0.25 ± 0.050	0.35 ± 0.062	0.37 ± 0.083	0.45 ± 0.075
FGA	0.1 ± 0.047	0.2 ± 0.083	0.27 ± 0.050	0.35 ± 0.031	0.37 ± 0.018
SGA	$\mathbf{0.245 \pm 0.03}$	$\mathbf{0.28 \pm 0.05}$	0.32 ± 0.05	0.375 ± 0.04	0.41 ± 0.07

Table 12: Misclassification rate (↑) on Pubmed with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 3 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.15 ± 0.022	0.3 ± 0.037	0.45 ± 0.092	0.47 ± 0.027	0.47 ± 0.082
Nettack	0.5 ± 0.045	0.62 ± 0.037	$\mathbf{0.67 \pm 0.052}$	0.72 ± 0.032	0.75 ± 0.025
GOttack	0.57 ± 0.012	0.6 ± 0.037	0.65 ± 0.020	$\mathbf{0.72 \pm 0.012}$	$\mathbf{0.75 \pm 0.017}$
FGA	0.32 ± 0.025	0.48 ± 0.037	0.51 ± 0.012	0.50 ± 0.075	0.57 ± 0.050
SGA	$\mathbf{0.575 \pm 0.04}$	$\mathbf{0.655 \pm 0.067}$	0.65 ± 0.053	0.695 ± 0.021	0.725 ± 0.031
UGBA	0.53 ± 0.03	0.53 ± 0.074	0.64 ± 0.1	0.61 ± 0.21	0.53 ± 0.1

Table 13: Misclassification rate (↑) on OGB-Arxiv with budget $\Delta = 1$ to 5: GOttack achieves performance in 2 out of 5 tasks (GCN model).

Budget →	1	2	3	4	5
Random	0.69 ± 0.48	0.74 ± 0.3741	0.915 ± 0.045	0.835 ± 0.208	0.855 ± 0.195
Nettack	ERROR	ERROR	ERROR	ERROR	ERROR
GOttack	$\mathbf{0.915 \pm 0.052}$	$\mathbf{0.93 \pm 0.057}$	$\underline{0.935 \pm 0.068}$	$\underline{0.93 \pm 0.065}$	$\underline{0.925 \pm 0.064}$
FGA	OOR	OOR	OOR	OOR	OOR
SGA	$\underline{0.89 \pm 0.207}$	$\underline{0.895 \pm 0.181}$	$\mathbf{0.97 \pm 0.027}$	$\mathbf{0.94 \pm 0.084}$	$\mathbf{0.945 \pm 0.074}$
UGBA	0.46 ± 0.07	0.63 ± 0.16	0.66 ± 0.14	0.66 ± 0.05	0.65 ± 0.08
GRBCD	0.16 ± 0.029	0.19 ± 0.029	0.205 ± 0.082	0.245 ± 0.084	0.285 ± 0.08

Table 14: Misclassification rate (↑) on Cora with budget $\Delta = 1$ to 5: GOttack achieves performance in 2 out of 5 tasks (RGCN model).

Budget →	1	2	3	4	5
Random	0.22 ± 0.04	0.25 ± 0.08	0.35 ± 0.09	0.32 ± 0.08	0.35 ± 0.06
Nettack	$\underline{0.27 \pm 0.06}$	$\underline{0.42 \pm 0.06}$	$\mathbf{0.52 \pm 0.07}$	$\mathbf{0.6 \pm 0.09}$	$\mathbf{0.67 \pm 0.01}$
GOttack	0.25 ± 0.05	$\mathbf{0.45 \pm 0.04}$	$\underline{0.52 \pm 0.08}$	$\underline{0.57 \pm 0.04}$	$\underline{0.67 \pm 0.02}$
FGA	0.27 ± 0.11	0.35 ± 0.14	0.37 ± 0.06	0.47 ± 0.16	0.55 ± 0.07
SGA	$\mathbf{0.27 \pm 0.01}$	0.4 ± 0.04	0.47 ± 0.09	0.52 ± 0.17	$\mathbf{0.67 \pm 0.01}$

the proposed GOttack method outperforms all baseline adversarial techniques in 13 out of 25 tasks. SGA and Nettack achieve the performance in 7 and 5 tasks out of 25, respectively. It is noteworthy that the SGA model could not run on the large BlogCatalog dataset, indicating SGA is not scalable for large datasets.

11.2 Attack Results on GraphSAGE

The results of adversarial techniques on GraphSAGE are presented in Tables 15 to 18. In summary, the proposed GOttack outperforms all baseline adversarial techniques by achieving the highest misclassification rate in 14 out of 20 tasks. In addition, SGA and Nettack accomplish the second and third highest performance in 5 and 3 tasks, respectively (ties are counted extra).

11.3 Attack Results on GIN

In this subsection, we discuss the performance of proposed GOttack along with various adversarial attack techniques on GIN, as shown in Tables 19 to 22. Results show that GOttack outperforms all baseline adversarial techniques in 9 out of 20 tasks, while SGA, Nettack and FGA achieve the performance in 8, 3 and 1 tasks, respectively (ties are counted extra).

Table 15: Misclassification rate (↑) on Cora with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 4 out of 5 tasks (GraphSAGE model).

Budget →	1	2	3	4	5
Random	0.55 ± 0.037	0.65 ± 0.031	0.7 ± 0.094	0.73 ± 0.06	0.77 ± 0.06
Nettack	0.58 ± 0.045	0.66 ± 0.042	0.7 ± 0.067	0.74 ± 0.054	0.77 ± 0.027
GOttack	0.59 ± 0.055	**0.78 ± 0.054**	**0.86 ± 0.014**	**0.88 ± 0.029**	**0.92 ± 0.033**
1819 orbit attack	0.5 ± 0.053	0.52 ± 0.08	0.56 ± 0.135	0.52 ± 0.083	0.53 ± 0.08
1519 orbit attack	0.52 ± 0.031	0.54 ± 0.014	0.58 ± 0.037	0.59 ± 0.076	0.54 ± 0.065
1922 orbit attack	0.52 ± 0.045	0.59 ± 0.049	0.55 ± 0.04	0.57 ± 0.085	0.59 ± 0.045
FGA	0.54 ± 0.089	0.57 ± 0.082	0.68 ± 0.072	0.7 ± 0.108	0.71 ± 0.029
SGA	**0.61 ± 0.06**	0.68 ± 0.06	0.67 ± 0.09	0.78 ± 0.04	0.73 ± 0.08
UGBA	0.59 ± 0.068	0.54 ± 0.165	0.6 ± 0.142	0.64 ± 0.095	0.74 ± 0.074

Table 16: Misclassification rate (↑) on Citeseer with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 4 out of 5 tasks (GraphSAGE model).

Budget →	1	2	3	4	5
Random	0.64 ± 0.119	0.7 ± 0.051	0.73 ± 0.074	0.71 ± 0.049	0.74 ± 0.052
Nettack	**0.66 ± 0.091**	0.68 ± 0.123	0.72 ± 0.069	0.77 ± 0.065	0.74 ± 0.082
GOttack	0.61 ± 0.093	**0.83 ± 0.062**	**0.92 ± 0.029**	**0.95 ± 0.047**	**0.97 ± 0.041**
1819 orbit attack	0.52 ± 0.029	0.56 ± 0.052	0.62 ± 0.043	0.6 ± 0.057	0.65 ± 0.047
1519 orbit attack	0.55 ± 0.085	0.55 ± 0.12	0.62 ± 0.107	0.59 ± 0.038	0.63 ± 0.057
1922 orbit attack	0.5 ± 0.06	0.62 ± 0.06	0.54 ± 0.06	0.64 ± 0.042	0.6 ± 0.083
FGA	0.6 ± 0.113	0.65 ± 0.079	0.74 ± 0.072	0.76 ± 0.052	0.76 ± 0.082
SGA	0.6 ± 0.06	0.68 ± 0.06	0.67 ± 0.09	0.78 ± 0.037	0.73 ± 0.08

Table 17: Misclassification rate (↑) on Polblogs with budget $\Delta = 1$ to 5: GOttack achieves performance in 1 out of 5 tasks (GraphSAGE model).

Budget →	1	2	3	4	5
Random	0.31 ± 0.052	0.34 ± 0.048	0.4 ± 0.078	0.45 ± 0.068	0.48 ± 0.089
Nettack	0.29 ± 0.029	0.34 ± 0.078	0.36 ± 0.074	0.39 ± 0.068	0.38 ± 0.115
GOttack	0.29 ± 0.038	0.36 ± 0.054	0.44 ± 0.072	0.49 ± 0.084	**0.54 ± 0.099**
1819 orbit attack	0.21 ± 0.091	0.25 ± 0.077	0.29 ± 0.042	0.3 ± 0.113	0.3 ± 0.066
1519 orbit attack	0.23 ± 0.112	0.24 ± 0.102	0.24 ± 0.08	0.22 ± 0.053	0.24 ± 0.07
1922 orbit attack	0.2 ± 0.074	0.2 ± 0.074	0.21 ± 0.045	0.24 ± 0.08	0.25 ± 0.093
FGA	0.22 ± 0.081	0.28 ± 0.071	0.32 ± 0.084	0.33 ± 0.096	0.37 ± 0.11
SGA	**0.35 ± 0.057**	**0.38 ± 0.065**	**0.51 ± 0.093**	**0.52 ± 0.079**	0.52 ± 0.03

Table 18: Misclassification rate (↑) on Pubmed with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 5 out of 5 tasks (GraphSAGE model).

Budget →	1	2	3	4	5
Random	0.47 ± 0.08	0.52 ± 0.05	0.52 ± 0.09	0.52 ± 0.06	0.75 ± 0.05
Nettack	**0.52 ± 0.07**	0.6 ± 0.05	0.65 ± 0.07	**0.77 ± 0.07**	0.52 ± 0.06
GOttack	**0.52 ± 0.08**	**0.67 ± 0.06**	**0.7 ± 0.08**	**0.77 ± 0.09**	**0.77 ± 0.05**
FGA	0.42 ± 0.03	0.55 ± 0.05	0.60 ± 0.07	0.62 ± 0.06	0.70 ± 0.04
SGA	0.3 ± 0.00	0.475 ± 0.00	0.575 ± 0.00	0.55 ± 0.00	0.55 ± 0.00

Table 19: Misclassification rate (↑) on Cora with budget $\Delta = 1$ to 5: GOttack achieves performance in 0 out of 5 tasks (GIN model).

Budget →	1	2	3	4	5
Random	0.43 ± 0.088	0.45 ± 0.037	0.53 ± 0.052	0.57 ± 0.082	0.6 ± 0.057
Nettack	0.46 ± 0.108	0.55 ± 0.116	0.62 ± 0.084	**0.64 ± 0.091**	**0.66 ± 0.049**
GOttack	0.37 ± 0.037	0.48 ± 0.076	0.54 ± 0.074	0.59 ± 0.101	0.64 ± 0.076
1819 orbit attack	0.34 ± 0.089	0.32 ± 0.055	0.38 ± 0.063	0.36 ± 0.084	0.4 ± 0.112
1519 orbit attack	0.36 ± 0.029	0.36 ± 0.048	0.36 ± 0.091	0.37 ± 0.041	0.4 ± 0.069
1922 orbit attack	0.35 ± 0.085	0.34 ± 0.091	0.36 ± 0.065	0.38 ± 0.121	0.38 ± 0.096
FGA	0.4 ± 0.095	0.46 ± 0.058	0.52 ± 0.027	0.57 ± 0.027	0.64 ± 0.033
SGA	**0.57 ± 0.06**	**0.63 ± 0.06**	**0.61 ± 0.065**	0.57 ± 0.09	0.61 ± 0.04

Table 20: Misclassification rate (↑) on Citeseer with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 5 out of 5 tasks (GIN model).

Budget →	1	2	3	4	5
Random	0.54 ± 0.058	0.6 ± 0.102	0.63 ± 0.078	0.62 ± 0.085	0.73 ± 0.076
Nettack	**0.57 ± 0.049**	0.6 ± 0.084	0.64 ± 0.033	0.7 ± 0.035	0.74 ± 0.065
GOttack	**0.57 ± 0.049**	**0.6 ± 0.077**	**0.66 ± 0.099**	**0.74 ± 0.104**	**0.76 ± 0.074**
1819 orbit attack	0.42 ± 0.136	0.45 ± 0.064	0.47 ± 0.094	0.43 ± 0.08	0.5 ± 0.033
1519 orbit attack	0.39 ± 0.14	0.44 ± 0.119	0.43 ± 0.063	0.48 ± 0.055	0.44 ± 0.089
1922 orbit attack	0.45 ± 0.066	0.47 ± 0.091	0.39 ± 0.101	0.42 ± 0.014	0.46 ± 0.114
FGA	0.44 ± 0.102	0.57 ± 0.104	0.56 ± 0.084	0.64 ± 0.108	0.64 ± 0.054
SGA	0.57 ± 0.060	0.62 ± 0.061	0.61 ± 0.067	0.56 ± 0.090	0.6 ± 0.040

Table 21: Misclassification rate (↑) on Polblogs with budget $\Delta = 1$ to 5: GOttack achieves performance in 0 out of 5 tasks (GIN model).

Budget →	1	2	3	4	5
Random	0.12 ± 0.045	0.16 ± 0.038	0.2 ± 0.033	0.22 ± 0.042	0.24 ± 0.078
Nettack	0.13 ± 0.029	0.2 ± 0.048	0.29 ± 0.055	0.32 ± 0.048	0.38 ± 0.061
GOttack	0.15 ± 0.086	0.23 ± 0.048	0.28 ± 0.011	0.32 ± 0.042	0.34 ± 0.029
1819 orbit attack	0.15 ± 0.048	0.16 ± 0.022	0.15 ± 0.037	0.14 ± 0.049	0.16 ± 0.065
1519 orbit attack	0.12 ± 0.053	0.16 ± 0.068	0.18 ± 0.056	0.16 ± 0.052	0.18 ± 0.045
1922 orbit attack	0.15 ± 0.031	0.12 ± 0.066	0.19 ± 0.052	0.17 ± 0.048	0.18 ± 0.04
FGA	0.14 ± 0.029	0.14 ± 0.048	0.15 ± 0.031	0.17 ± 0.037	0.2 ± 0.040
SGA	**0.35 ± 0.080**	**0.35 ± 0.070**	**0.37 ± 0.120**	**0.4 ± 0.076**	**0.5 ± 0.097**

Table 22: Misclassification rate (↑) on Pubmed with budget $\Delta = 1$ to 5: GOttack achieves highest performance in 4 out of 5 tasks (GIN model).

Budget →	1	2	3	4	5
Random	0.5 ± 0.09	0.65 ± 0.07	0.62 ± 0.09	0.52 ± 0.08	0.6 ± 0.08
Nettack	0.47 ± 0.02	0.6 ± 0.08	0.6 ± 0.07	0.57 ± 0.07	0.52 ± 0.06
GOttack	**0.55 ± 0.05**	0.6 ± 0.04	**0.67 ± 0.06**	**0.67 ± 0.08**	0.67 ± 0.07
FGA	0.52 ± 0.01	**0.67 ± 0.06**	0.62 ± 0.03	0.52 ± 0.01	0.52 ± 0.02
SGA	0.475 ± 0.018	0.605 ± 0.011	0.66 ± 0.022	0.67 ± 0.021	**0.685 ± 0.014**

12 Additional proof of GOttack's impact

Node's features. Another interesting property of the proposed attacks can be seen in Table 23 and Table 28, in which we observe the change in the target node characteristics after adding or removing an edge between different orbit types. More precisely, we consider degree centrality, closeness centrality, betweenness centrality, and clustering coefficient as node feature metrics. The results show that adding/removing an edge between the target node and the connecting node with frequent orbit type causes a significant shift node feature, especially betweenness centrality of the target nodes.

Table 23: Changes in the node attribute of the target node, averaged over all target nodes (CORA dataset). A positive value indicates an increase after the attack compared to before.

Attack Orbit Type	Degree Centrality	Closeness Centrality	Betweenness Centrality	Clustering Coefficient
GOttack (1518)	+15.18	+6.5	+20.32	-3.46
1922	+15.19	+17.01	+25.54	+1.35
1519	+15.21	+4.58	+23.6	-3.43
1819	+15.22	+9.34	+23.61	-3.42

Table 24: Shortest path length changes of the target node to the candidate node, with seven rows each representing a different label node group. These are mean values calculated over target test nodes. LS denotes same label with the target node, LD denotes different label with the target node. A negative value indicates a decrease after the attack compared to before.

	1518		1922		1519		1819	
	LS	LD	LS	LD	LS	LD	LS	LD
Label-0	-0.06	-0.08	-0.04	-0.01	0.0	0.0	-0.01	0.0
Label-1	-0.01	-0.01	-0.07	-0.03	-0.01	-0.01	0.0	0.0
Label-2	-0.02	-0.02	-0.03	-0.02	-0.05	-0.03	-0.03	-0.02
Label-3	-0.05	-0.08	-0.05	-0.04	-0.06	-0.04	-0.07	-0.04
Label-4	0.00	-0.01	-0.01	-0.01	-0.04	-0.02	0.0	-0.01
Label-5	0.0	0.0	-0.02	0.0	-0.05	-0.03	-0.02	-0.01
Label-6	-0.03	-0.02	-0.02	-0.01	-0.03	-0.01	-0.01	-0.01

Table 25: Changes in the nodes' labels within two-hop neighbors (CORA dataset) of the target node, with seven rows each representing a different label node group. LS denotes same label with the target node, LD denotes different label with the target node. These are mean values calculated over target test nodes. A positive value indicates an increase after the attack compared to before.

	1518		1922		1519		1819	
	LS	LD	LS	LD	LS	LD	LS	LD
Label-0	+1.25	+3.7	+0.63	+4.75	+1.28	+3.88	+1.6	+3.5
Label-1	+1.27	+3.58	+0.5	+4	+0.58	+3.73	+0.48	+4.05
Label-2	+1.3	+2.65	+2.36	+3.53	+0.78	+3.15	+0.55	+3.75
Label-3	+1.0	+3.5	+0.43	+4.53	+1.28	+3.58	+1.08	+3.88
Label-4	+0.525	+3.2	+0.68	+3.8	+0.5	+3.5	+0.65	+3.6
Label-5	+0.125	+3.55	0.28	+4.1	+0.33	+3.75	+0.45	+3.95
Label-6	+0.975	+3.4	+1.88	+3.8	+1.63	+3.43	+1.48	+3.83

Shortest path scores. In this experiment, we aim to observe the change in shortest path scores before and after attacks, calculating the minimum traveling cost from the source node to the destination node. Table 24 presents the results for different orbit types and node labels in the Cora dataset. Here, the notation **LS** denotes the score when connecting edges between nodes with the same label, while **LD** denotes the score when connecting edges between nodes with different labels. It is worth noting that connecting edges between different labels results in higher or equal changes in shortest path scores compared to connections between nodes with the same label. The results also indicate that connecting nodes with different or the same labels belonging to the 1518 orbit types results in higher changes compared to other orbit types.

Nodes' label in two hops neighbors changes. As we know, GNN classification depends not only on the node itself but also on its neighbors. Given two nodes in the network, if they belong to the same class, their embeddings will exhibit high similarity. Considering this, we investigate the number of nodes that share the same label (LS) as the target node (v) and the number of nodes with different labels (LD) before and after applying the attack.

It is noteworthy that when we apply an attack on two convolution layers of GNN variants, the embeddings of nodes in the two-hop neighborhood significantly contribute to updating the embedding of the target node. Table 25 shows the changes in the two-hop neighbors of target nodes for Cora dataset. From the results, we can observe that the target node is connected to LD nodes, which belong to 1518 orbit types. The neighbor change is significant, with an average increase of $+3.36$ for the GOttack method. Furthermore, the results of the same experiment with the same settings conducted on Citeseer dataset are shown in the Appendix, Table 29.

Important edge. After adding a new edge to the target node, the embedding of the target nodes is updated with information aggregated through the new edge. The more important the edge in the two-hop neighbor of the target node, the more noise is added to the target node that eventually causes miss-classification.

Table 26: Edge betweenness of newly added edges between 1518 nodes and target nodes.

Newly added edge to	Edge betweeness
Misclassified nodes	$\mathbf{0.44 \pm 0.18}$
Correctly classified nodes	0.22 ± 0.14

The importance of edge can be assessed based on edge betweenness centrality, we conducted 1518 GOttack on GCN, using the Cora dataset once to get the list of the target node and corresponding connecting nodes chosen by our algorithm. We separate them into two classes, one is the list of target nodes that are miss-classified and the other is the list of correct-classified. Table 26 shows that the average edge betweenness of edges added to miss-classified nodes is 0.44, which is twice compared to the average edge betweenness of edges added to correct-classified nodes, 0.22. It proves that our attack harms node classification accuracy by discovering and adding important edges, which connect the target node to a faraway region in the graph and bring the noise to the target nodes' embedding.

Table 27: Time Experiment Results in Seconds (↓).

	GOttack	Nettack	FGA	SGAttack	Random
Cora	66.69 ± 3.34	82.62 ± 3.34	86.57 ± 3.4	$\mathbf{57.49 \pm 0.88}$	57.64 ± 3.42
Citeseer	$\mathbf{75.57 \pm 3.52}$	92.56 ± 2.08	128.52 ± 1.52	98.48 ± 3.32	76.64 ± 1.95
Polblogs	123.08 ± 4.27	143.81 ± 10.56	$\mathbf{93.6 \pm 3.4}$	94.35 ± 4.72	89.91 ± 8.71
Blogcatalog	1735.16 ± 18.72	2003.69 ± 29.17	$\mathbf{1298.8 \pm 24.58}$	$> 24h$	1181.88 ± 19.92
Pubmed	$\mathbf{1347.9 \pm 16.72}$	1582.35 ± 27.59	$2.84h$	$> 24h$	1485.42 ± 34.02

Table 28: Changes in the node attribute of the target node, averaged over all target nodes (POLBLOGS dataset). A positive value indicates an increase after the attack compared to before.

Orbit type	LSame-2 hop	LDiff-2hop	Degree Centrality(10^{-3})	Closeness Centrality(10^{-2})	Betweenness Centrality(10^{-3})	Cluster
GOttack (1518)	8.3 ± 1.06	1.58 ± 0.3	0.819 ± 0.0	0.35 ± 0.14	0.11 ± 0.02	-0.042 ± 0.034
1922	8.89 ± 1.14	1.51 ± 0.2	0.819 ± 0.0	0.43 ± 0.17	0.12 ± 0.03	-0.038 ± 0.036
1519	9.25 ± 0.83	0.975 ± 0.46	0.819 ± 0.0	0.43 ± 0.17	0.12 ± 0.03	-0.038 ± 0.036
1819	9.64 ± 1.18	$4.29 \pm 0.6.32$	0.95 ± 0.0	0.47 ± 0.2	0.12 ± 0.05	-0.052 ± 0.038

12.1 Subgraph-based Explanations

GNNs integrate node features information and graph structure by recursively passing messages along the edges of the graph to update the embedding and make predictions, therefore, this complex integration leads to the models that are challenging to explain in terms of their predictions.

In this subsection, we leverage two different renowned explainers (subgraph-based) to see the changes that the GOttack method poses to graphs that lead to misclassification.

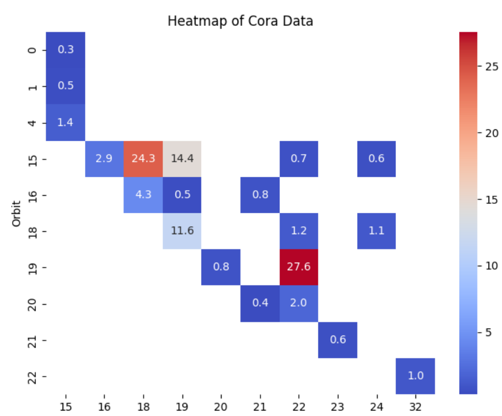
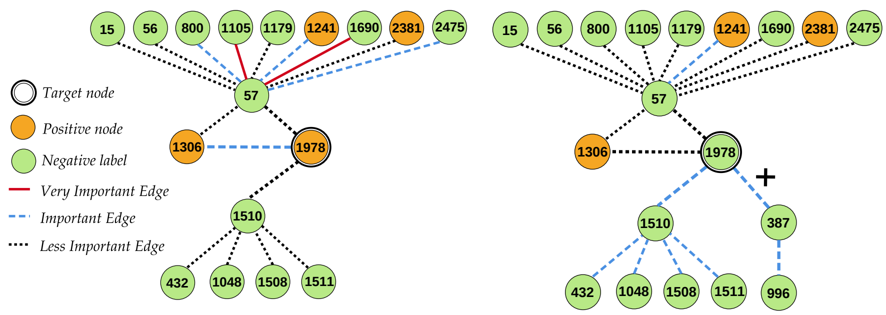


Figure 8: Matrix of co-occurrence percentages for node orbits: visualizing intersections between the first and second orbit.

GNNExplainer. Battaglia et al. [3], Zhou et al. [48] and Zhang et al. [46] summarizes the update of GNN model in three core computations. (i) For every pair of node (v_i, v_j), the message is computed from the representations $\mathbf{h_i^{l-1}}$, $\mathbf{h_j^{l-1}}$ of each node in the previous layer and their relation r_{ij}, given by $m_{ij}^l = \mathbf{MSG}(\mathbf{h_i^{l-1}}, \mathbf{h_j^{l-1}}, r_{ij})$. (ii) Secondly, an aggregated message $\mathbf{M}_i$ is computed for each node v_i by aggregating messages from all v_i's neighborhood. (iii) Finally, the representation, also known as embedding, $\mathbf{h_i^l}$ of node v_i at layer l is calculated from the embedding of the node v_i in the previous layer and aggregated message $\mathbf{M}_i$. This demonstrates that the neighbour of all node v_i contributes to formulating the final embedding of v_i. Ying et al. defined the subgraph of all neighbours of the node v_i as a computation graph, denoted as $\mathcal{G}_c$ [42].

Table 29: Changes in the nodes' labels within two-hop neighbors (CITESEER dataset) of the target node, with six rows each representing a different label node group. LS denotes same label with the target node, LD denotes different label with the target node. These are mean values calculated over target test nodes. A positive value indicates an increase after the attack compared to before.

	1518		1922		1519		1819	
	LS	LD	LS	LD	LS	LD	LS	LD
Label-0	+0.45	+3.78	+0.95	+4.5	+0.15	+3.43	+0.05	+4.15
Label-1	+0.58	+3.58	+2.5	+3.35	+0.33	+3.3	+1.53	+3.08
Label-2	+2	+3.03	+1.48	+4.18	+2.8	+2.28	+0.63	+3.8
Label-3	+0.15	+3.73	+0.38	+4.18	+0.48	+3.23	+0.65	+3.5
Label-4	+2.55	+2.95	+1.7	+3.3	+1.13	+3.28	+1.78	+2.88
Label-5	+1.28	+3.12	0.48	+4.18	+1.65	+3.38	+0.65	+3.93



(a) Computation graph before attack

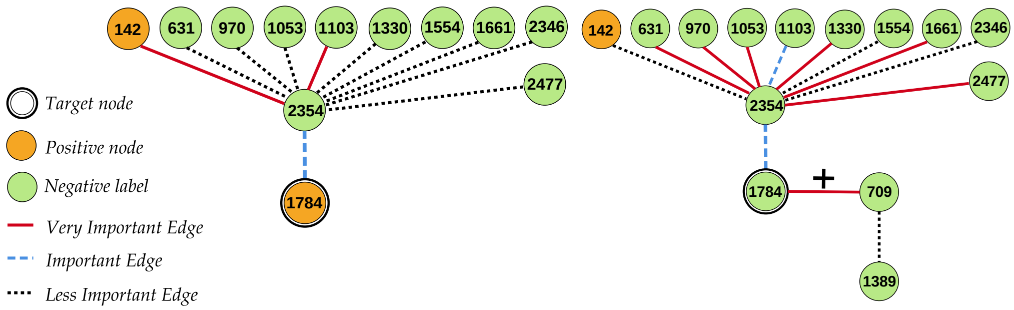
(b) Computation graph after attack

Figure 9: The computation graph for the targeted node 1978 from the CORA datasets, as identified by PGExplainer [26] . The edge (1978, 387) is added during the successful attack. Edge importances change considerably after the attack and negative class gains importance due to the newly added nodes.

In particular, v's computation graph tells the GNN how to generate v's embedding z. The computation graph of node v is crucial, as it fully determines all the information GNN uses to generate prediction $\hat{y}_v$ at node v. Among nodes and edges in $\mathcal{G}_c$, we are interested in $\mathcal{G}_s \subseteq \mathcal{G}_c$ that are important for the GNN's prediction on node v, in which removing of either a node or an edge in $\mathcal{G}_s$ strongly decrease the probability of prediction $\hat{y}_v$. By solving the optimization of the conditional entropy $H(Y|\mathcal{G} = \mathcal{G}_s \ , \mathcal{X} = \mathcal{X}_s)$, GNNExplainer returns an explanation for the prediction $\hat{y}_v$ as $(\mathcal{G}_s, \mathcal{X}_s^F)$, where $\mathcal{G}_s$ is a small subgraph of the computation graph, $\mathcal{X}_s^F$ is the associated feature of $\mathcal{G}_s$, and $\mathcal{X}_s^F$ is a small subset of node features that are most important for explaining $\hat{y}_v$ [42].

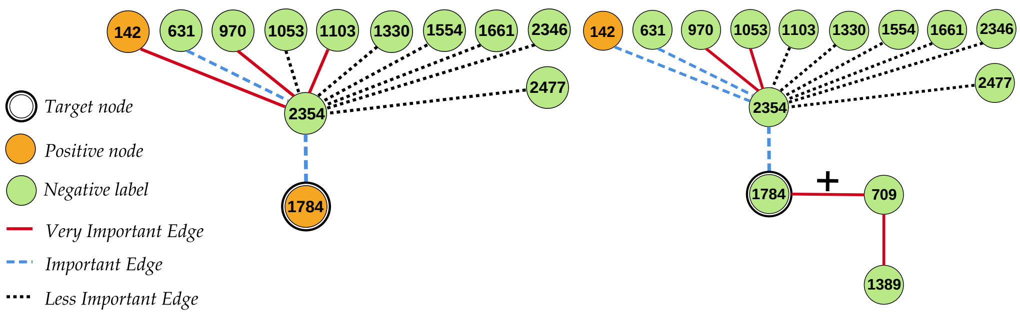
Figure 5 is the subgraph of the computation graph most influential for the GNN's prediction on node 1978, denoted as $\hat{y}_{1978}$. Before the attack, the prediction $\hat{y}_{1978}$ made by GNN on node 1978 is strongly affected by edges connecting to $1306, 1241$ and 2381 having the same label and 6 edges connecting to different label nodes. After the attack by adding an edge between the target node and 1518 node 387, there are two out of three edges connecting the node having the same label to 1978 is no longer having a strong impact on $\hat{y}_{1978}$, while there is an increase in the number of edges connecting different label nodes to 9. As a result, GNN is fooled into making a misprediction on 1978 after applying GOttack. In addition, we leverage GNNExplainer to explain another successful attack of GOttack on node 1784, described by Figure 10. Similarly, the newly added edge is considered as an important edge and there are significant changes in the importance of the remaining edge in the computational subgraph caused by the newly added edge $(1784, 709)$.

PGExplainer. PGExplainer [26] is a model-agnostic method designed to provide explanations for predictions made by GNNs. It is used to identify a subgraph and subset of node features crucial for a specific prediction. Let us consider a trained GNN model $f_\theta(\mathcal{G})$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$ is input graph. It aims to explain the prediction $\hat{y_v}$ for a target node $v \in \mathcal{V}$. It learns an edge mask $\mathbf{M}_e \in [0,1]^{|\mathcal{E}|}$ and a feature mask $\mathbf{M}_x \in [0,1]^{|\mathcal{X}|}$. The masks are optimized to maximize the mutual information between the GNN's predictions $\hat{y_v}$ and the subgraph $\mathcal{G}_s$, expressed as: $\mathbf{MI}(\hat{y_v}, \mathcal{G}_s) = H(\hat{y_v}) - H(\hat{y_v}|\mathcal{G} = \mathcal{G}_s)$



(a) Computation graph before attack (b) Computation graph after attack

Figure 10: The computation graph for the targeted node 1784 from the CORA datasets, as identified by GNNExplainer [26]. The edge $(1784, 709)$ is added during the successful attack. Edge importances change considerably after the attack and negative class gains importance due to the newly added nodes.



(a) Computation graph before attack (b) Computation graph after attack

Figure 11: The computation graph for the targeted node 1784 from the CORA datasets, as identified by PGExplainer [26] . The edge $(1784, 709)$ is added during the successful attack. Edge importances change considerably after the attack and negative class gains importance due to the newly added nodes.

However, optimizing this objective is infeasible due to the exponential number of possible subgraphs (i.e., 2^N), the problem is relaxed by assuming $\mathcal{G}_s$ is a Gilbert random graph where edge selections are conditionally independent. The probability of an edge e_{ij} being selected is modeled using a Bernoulli distribution: $P(\mathcal{G}) = \prod_{(i,j)\in\mathcal{E}} P(e_{ij})$, where $e_{ij} \in \mathcal{V} \times \mathcal{V}$ is a binary variable indicating whether the edge is selected, with $e_{ij} = 1$ if the edge (i, j) is selected, and 0 otherwise. The probability $P(e_{ij} = 1) = \theta_{ij}$ is the probability that edge (i, j) exists in $\mathcal{G}$. With this relaxation, the objective can be rewritten as [26]: $\min_{\mathcal{G}_s} H(\hat{y_v}|\mathcal{G} = \mathcal{G}_s) = \min_{\mathcal{G}_s} \mathbb{E}_{\mathcal{G}_s}[H(\hat{y_v}|\mathcal{G} = \mathcal{G}_s)] \approx \min_{\Theta} \mathbb{E}_{\mathcal{G}_s \sim q(\Theta)}[H(\hat{y_v}|G = \mathcal{G}_s)]$, where $q(\Theta)$ is the distribution of the explanatory graph parameterized by θ's.

Figure 9 shows the subgraph of the computation graph with the most influential edge connections. Before the attack, the prediction $\hat{y}_{1978}$ made by the GNN on node 1978 is strongly influenced by edges connecting to node 1306. However, after the attack, which involves adding an edge between the target node and node 387, the strong connection to node 1306 is no longer present. Instead, there is an increase in the number of edges connecting to different label nodes such as node 1510. Consequently, the GNN is misled into making an incorrect prediction for node 1978 after the application of GOttack. In addition, we leverage PGExplainer to explain another successful attack of GOttack on node 1784, described by Figure 11. Similarly, the newly added edge is considered as an important edge and there are significant changes in the importance of the remaining edge in the computational subgraph caused by the newly added edge $(1784, 709)$.

Table 30: The descriptions of symbols.

Symbol	**Descriptions**
$\mathcal{G}$	Graph representation
$\mathcal{V}$	Sets of vertices in $\mathcal{G}$
$\mathcal{E}$	Sets of edges in $\mathcal{G}$
$\lvert\mathcal{E}\rvert$	Number of edges
$\lvert\mathcal{V}\rvert$	Number of nodes
$\mathcal{A}$	Adjacency matrix of $\mathcal{G}$
$\mathcal{X}$	Node features
f_θ	Graph neural network model
v	Target node
$\mathcal{V}_T$	Set of target nodes
$\mathcal{N}(v)$	Set of adjacency nodes of v
$h(\cdot)$	Graph homophily ratio
$\mathcal{G}'$	Perturbed graph
$\mathcal{G}_c$	Computation graph
$\mathcal{G}_s$	Subgraph of computation graph
$\mathcal{A}'$	Perturbed adjacency matrix
$\hat{y}_v$	Predicted class
Δ	Attack budget
$\mathcal{G}_{gp}$	Graphlet
Aut($\mathcal{G}_{gp}$)	Automorphisms of a graphlet
GOV	Graphlets Orbit Vector
Orb_{max}	Largest orbit count value
Orb_{sec}	Second-largest count value

Table 31: Summary of the Literature review.

Ref.	Article Name	Attack Type	Perturbation	Evasion/ Poisoning	Domain	Model	Baseline	Metrics	Dataset
[52]	Nettack	Target Attack	Structure Feature	Both	Node Classif.	GCN CLN DeepWalk	Random FGSM	Accuracy Classif. Margin	Cora-ML Citeseer Polblogs
[8]	FGA	Target Attack	Structure	Both	Node Classif.	GCN Grarep DeepWalk Node2vec Line GraphGAN	Random DICE Nettack	Success Rate AML	Cora-ML Citeseer Polblogs
[54]	Mettack	Global Attack	Structure Feature	Poisoning	Node Classif.	GCN CLN DeepWalk	DICE Nettack First-order attack	Accuracy Misclassif. Rate	Cora-ML Pubmed Citeseer Polblogs
[9]	RL-S2V	Target Attack	Structure	Evasion	Node Classif.	GNNs	Rnd. sampling Genetic algs.	Accuracy	Citeseer Finance Pubmed Cora
[5]	Node Embedding	Global Attack	Structure	Poisoning	Node Embedding	DW SVD DW SGNS Node2vec Spect. Embd Label Prop. GCN	Unknown	Accuracy Classif. Margin Loss	Cora Citeseer Polblogs
[39]	PGD, Min-max	Global Attack	Structure	Both	Node Classif.	GCN	DICE Greedy Meta-self	Misclassif. Rate	Cora Citeseer
[36]	DICE	Global Attack	Structure	Both	Node Classif.	GCN	DICE ROAM heuristic	Concealment	TerroristNet Facebook Twitter Google+ ScaleFree SmallWorld RandomGraph
[37]	IG-Attack	Target Attack	Structure Feature	Both	Node Classif.	GCN	JSMA IG-JSMA Nettack FGSM	Classif. Margin Accuracy	Cora Citeseer Polblogs
[32]	NIPA	Global Attack	Structure	Poisoning	Node Classif.	GCN	Random FGA Preferential attack	Accuracy Graph Statistics	Cora-ML Pubmed Citeseer
[24]	SGAttack	Target Attack	Structure	Poisoning	Node Classif.	GCN GAT SGC GraphSAGE ClusterGCN	Random DICE GradArgmaxNettack	Accuracy Classif. Margin	Citeseer Cora Pubmed Reddit
[27]	GC-RWCS	Target Attack	Structure	Evasion	Node Classif.	GCN JKNetConcat JKNetMaxpool	Random Degree Pagerank Betweenness RWCS GC-RWCS	Accuracy Loss	Citeseer Cora Pubmed
[44]	GNNGuard	Target Attack	Feature	Poisoning	Node Classif.	GCN GAT GIN JK-Net GraphSAINT	Nettack-Di NettackInMettack	Accuracy	Cora Citeseer ogbn-arxiv DP
[7]	GF-Attack	Global Attack	Feature	Evasion	Vertex Classif.	GCN SGC Cheby DW LINE	Random Degree RL-S2V A_class GF-Attack	Accuracy Execution Time	Cora Citeseer Pubmed
[49]	RGCN	Target Attack	Structure	Poisoning	Node Classif.	GCN GAT	Random RL-S2V Nettack	Accuracy	Cora Citeseer Pubmed
[34]	G-NIA	Target Attack	Structure Feature	Evasion	Node Classif.	GCN GAT APPNP	Radnom MostAttr. PrefEdge. NIPA AFGSM G-NIA	Misclassif. Rate	Reddit Citeseer ogbn-products
[31]	OPT-Attack	Target Attack	Structure	Poisoning	Node Embedding	GAE DeepWalk Node2vec LINE	Degree sum Shortest path Random PageRank	AP Similarity Score	Cora Citeseer Facebook
[17]	STRUCtack	Global Attack	Structure	Poisoning	Node Classif.	GCN	Random DICE Mettack PGD MinMax	Accuracy	Cora Citeseer Pubmed Cora-ML Polblogs
[38]	WT-AWP	Unknown	Feature	Poisoning Evasion	Node Classif. Graph Classif.	GCN GAT PPNP	DICE PGD Mettack	Accuracy Loss	Cora Citeseer Polblogs
[51]	TDGIA	Unknown	Feature	Evasion	Node Classif.	GCN	FGSM AFGSMSPEIT	Accuracy	KDD-CUP ogbn-arxiv Reddit
[35]	CANA	Unknown	Feature	Unknown	Unknown	COPOD PCA HBOS IForestAE	PGD TDGIA G-NIA	Accuracy Misclassif. Rate	ogbn-products redditogbn-arxiv
[47]	HGAttack	Target Attack	Unknown	Evasion	Node Embedding	GCN	FGA	Macro F1 Micro F1	ACM DBLP IMDB
[18]	PEH	Unknown	Structure	Unknown	Node Classif. Node Clustering	GCN GAT DGI RGCN	Nettack Mettack Random PGD	Accuracy Attention Score	Cora Citeseer Polblogs

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state that GOttack is designed to attack node classification models by perturbing graph orbits and demonstrate its effectiveness through empirical evaluations on multiple datasets. The scope and contributions align with these claims.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed the limitation of excluding node and edge features from GOttack since orbits cannot use them, which may restrict the generalizability of the method.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical results in the paper are accompanied by clear assumptions and detailed proofs, which are presented in both the main text and supplemental material.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides a comprehensive description of the datasets, model configurations, and hyperparameters used, ensuring that the experimental results are reproducible.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is shared on an anonymous GitHub repository, with detailed instructions provided in the supplemental material to ensure reproducibility.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all relevant details regarding the training and testing settings, including data splits, hyperparameters, optimizers, and other configuration settings.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We show results of five runs along with standard deviations, providing clear information on the statistical significance of the experiments.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper details the compute resources used for the experiments, including the type of hardware, memory, and time required for execution, ensuring transparency and reproducibility.

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research adheres to the NeurIPS Code of Ethics, with considerations for responsible AI practices and transparency in the methodologies and data used.

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We do not see a societal impact or ethical issue associated with the work performed in this paper.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper does not involve the release of models or data that have a high risk of misuse, and thus this question is not applicable.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper credits the creators of all assets used, including datasets and code, and adheres to the respective licenses and terms of use.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not introduce any new assets in this paper.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The research does not involve human subjects, and therefore, IRB approval is not applicable.