MFAN ATTACK ON GRAPH NEURAL NETWORKS

MULTIFACETED ANCHOR NODES ATTACK ON GRAPH NEURAL NETWORKS: A BUDGET-EFFICIENT APPROACH

BY
HUANZHANG ZHU, B.CompSc.

A Thesis
submitted to the Computing and Software
and the School of Graduate Studies
of McMaster University
in partial fulfilment of the requirements
for the degree of
Master of Science

Master of Science (2024) McMaster University
(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Multifaceted Anchor Nodes Attack on Graph Neural Networks: A Budget-efficient Approach
AUTHOR:	Huanzhang Zhu B.CompSc. (Computer Science), Concordia University, Montreal, Canada
SUPERVISOR:	Dr. Lingyang Chu
NUMBER OF PAGES:	xiv, 33

Lay Abstract

This study introduces a new and cost-effective method for launching attacks on graph neural networks (GNNs), which are widely used in applications like social media and recommendation systems. Traditional attacks on GNNs focus on altering the connections between nodes to disrupt the model, but they often require control over many nodes, making them expensive and easier to detect. Our approach improves on this by using multiple small sets of "anchor nodes" that work together with an assignment network to choose the best set for each attack. This method achieves high attack success while keeping costs low, since fewer nodes need to be controlled. Experiments on real-world data show that our method is highly effective and efficient.

Abstract

Structural adversarial attack methods, which attack graph neural networks (GNNs) by perturbing the edges of the input graph, are well-recognized for their high effectiveness. However, most existing structural attacks prioritize maximizing attack performance while neglecting the significant budget required to control (i.e., acquire or hijack) the nodes (e.g., user accounts in a social network) necessary for executing such attacks in real-world networks. Classic anchor node attacks are comparatively more budget-efficient, as they rely on controlling a small set of anchor nodes to conduct all attacks. Nevertheless, their attack efficacy is constrained by the limitation of using a single set of anchor nodes. In this work, we propose a strong and budget-efficient multifaceted anchor nodes attack on GNNs, with the core innovation lies in the simultaneous training of multiple sets of anchor nodes and an assignment network, enabling the assignment network to select the most optimal set of anchor nodes for each new attack. This approach significantly enhances attack effectiveness while maintaining a minimal budget for node control. Extensive experiments across five real-world datasets demonstrate the superior performance of the proposed method.

Acknowledgements

I would like to take this opportunity to thank all the people that helped me so much during my gradaute study.

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Chu, for accepting me as a master student. His wisdom and patience were the cornerstones of my academic progress. His ability to steer me through challenges while granting me the freedom to explore has been invaluable to my development as a researcher. For that, I am truly grateful.

A special thank you goes to my lab colleagues, who made this experience both collaborative and fulfilling. Your encouragement, shared knowledge, and countless brainstorming sessions were essential to overcoming the roadblocks along the way. I would specially thank Shaoxin, for his assistance during my graduate research project.

Additionally, I also extend my appreciation to the professors who have guided me throughout my graduate courses. Your teachings and insights have expanded my academic horizons and left a lasting impact on my approach to research.

Lastly, to my family: thank you for being my foundation. Your constant support, encouragement, and belief in me have been my greatest source of strength. Without your love and understanding, I wouldn't be where I am today.

Contents

Lay Abstract **iii**

Abstract **iv**

Acknowledgements **v**

Notation, Definitions, and Abbreviations **x**

Declaration of Academic Achievement **xiv**

1 Introduction **1**

2 Related Work **3**
2.1 Global Structural Attacks 3
2.2 Targeted Structural Attacks 4
2.3 Anchor Nodes Attacks 4
2.4 Graph Attacks with Other Strategies 5

3 Preliminary **6**
3.1 Graph 6
3.2 Graph Convolutional Network 6
3.3 Victim Model Types 7
3.4 Attacking Types 7

4 Problem Definition **9**
4.1 MFAN Task 9
4.2 Modelling the MFAN Attack 10
4.3 Formulating the Problem 11

5 Solution **13**
5.1 Solving the Problem 13
5.2 Algorithms 15

6 Experiments **17**
6.1 Experimental Setting 17
6.2 Fooling Ratio under Different Budgets of Controlled Nodes . . 19
6.3 Effectiveness of Transfer Attack 21
6.4 The Effects of K, λ, the Assignment Network, and Simulated Annealing 22
6.5 Time Analysis 25

7 Future Work **28**

8 Conclusion **29**

List of Figures

4.1 MFAN attack procedure 10
6.1 Fooling ratio (FR) v.s. budget per target node (BPT, ξ). . . . 20
6.2 Fooling ratio (FR) v.s. budget for all target nodes (BFA, δ). . 20
6.3 The curves of total loss, original loss, SPT and λ 23

List of Tables

6.1 Model configuration of the assignment network. 18
6.2 Dataset statistics. 18
6.3 FR of non-transfer and transfer attacks. 22
6.4 FR of MFAN with different K values. 23
6.5 FR of MFAN using λ growth rates $v \in \{2.5, 5.0, 10.0\}$. 24
6.6 The effect of the assignment network on FR. 24
6.7 The effect of simulated annealing on FR and QE. 25
6.8 Average attacking time in milliseconds. 26
6.9 Training time complexity. 26
6.10 Average training time in milliseconds. 27

Notation, Definitions, and Abbreviations

Notation

G	The unweighted graph
V	The set of vertices in a graph
E	The set of edges in a graph
A	The adjacency matrix of a graph
X	The feature matrix of a graph
N	The number of nodes in a graph
d	The dimension of the feature of a node
C	The number of classes of node labels
v_i	The i-th node in a graph
I	The identity matrix
$\hat{A}$	The normalized adjacency matrix
$\tilde{D}$	The diagonal degree matrix
H^l	The l-th hidden graph convolutional layer of a GCN
W^l	The model parameters at the l-th layer of a GCN
f	The GCN
Z	The output of a GCN f
Z_{ij}	The probability of the i-th node being classified into the j-th class

K	The number of sets of anchor nodes
$\mathcal{Q}$	The sets of anchor nodes
Q	The set of anchor nodes
Q_k	The k-th set of anchor nodes
g_θ	The assignment network
θ	The model parameters of the assignment network
ξ	The budget on the number of controlled nodes for each target node
δ	The budget on the number of controlled nodes for all target nodes
$\mathcal{P}$	The set of perturbation vectors
$\mathbf{p}$	The perturbation vector
$\mathbf{p}_k$	The k-th perturbation vector
$\mathbf{1}$	The matrix with all entries equal to one
$\mathbf{1}_0$	The matrix with all entries equal to one except for the diagonal entries being zeros
P	The matrix with i-th row and i-th column replaced by $\mathbf{p}_k$ and the other entries set to zero
G'	The perturbed graph
A'	The perturbed adjacency matrix
λ	The weight hyperparameter of the penalty term
v	The growth rate of λ
$\mathbf{w}_k$	The weight vector calcuated from $\mathbf{p}_k$
$\mathbf{p}_k^h$	The h-th entry in $\mathbf{p}_k$
$\mathbf{w}_k^{min}$	The smallest value among all entries in $\mathbf{w}_k$
T	The "temperature" hyperparameter of the simulated annealing

$\mathcal{L}$	The original loss function
$\mathcal{L}^*$	The final loss function
V_T	The training set of node in V
η_1	The learning rate for training $\mathcal{P}$
η_2	The learning rate for training δ
ρ	The perturbation function
σ	The annealing function
ϕ	The quantization function
max	The max function (maxima)
min	The min function (minima)
arg max	The arg max function (arguments of the maxima)
arg min	The arg min function (arguments of the minima)
`ReLU`	The ReLU (rectified linear unit) activation function
`softmax`	The softmax activation function
`CE`	The cross entropy loss function
`clip`	The clipping function
`quantize`	The quantizing (i.e., binarizing) function

Definitions

Flip The operation that either adds non-existing edge or removes an existing edge.

Anchor node The node in a graph anchored where the attack flips edges between it and the target nodes.

Perturbation The attacking operation to the victim GNN model that flips edges between each anchor node and the target nodes.

Assignment network The model that selects the most suitable set of anchor nodes to attack a given target node.

Abbreviations

GNN	Graph neural network
GCN	Graph convolutional network
GAT	Graph attention network
FR	Fooling ratio
BPT	Budget per target node
BFA	Budget for all target nodes
SPT	Sum of penalty terms

Declaration of Academic Achievement

I, Huanzhang Zhu, hereby declare that the academic achievement presented in the thesis is the result of own original efforts and was carried out under the supervision of Dr. Lingyang Chu.

The work has been accepted by the 27th International Conference on Pattern Recognition (ICPR 2024).

Chapter 1

Introduction

Graph neural networks (GNNs) are extensively utilized across various real-world application domains, such as social network analysis [4, 25], recommendation systems [6, 14] and drug discovery [9], which have been demonstrated substantial efficacy in practice. In recommendation systems, a user can be suggested commodities that align with their preferences, which are inferred from patterns shared by similar users encoded within the graph structure. Similarly, in social networks, recommendations for users are also generated based on this underlying graph representation. GNNs have also proven to be highly valuable in drug discovery as well, enabling the prediction of drug effects or side effects by modeling drug-protein and protein-protein interactions through graph edges.

Despite their widespread applicability, GNNs are inherently susceptible to adversarial attacks [11, 19, 29, 31, 34, 41], which can result in significant negative societal implications. In the financial domain, a money laundering account may evade detection by engaging in transactions that appear legitimate when interacting with other verified accounts. In social networks, malicious entities such as spam bots, phishing accounts, and scam profiles may form ostensibly legitimate connections to circumvent account verification protocols. Similarly, falsified articles on platforms like Wikipedia can enhance their credibility by strategically altering their network of links [21].

To enhance the security and robustness of GNNs, numerous studies have been conducted to identify potential vulnerabilities in trained GNN models by developing powerful adversarial attacks. Among the various adversarial attack strategies [5, 8, 11, 17, 19, 23, 24, 29, 31, 33, 34, 38, 39, 40, 41], the structural attack methods [5, 7, 12, 17, 23, 24, 33, 35, 38, 39, 40], which which compromise GNNs by perturbing (i.e., modifying) the edges of the input graph, are widely recognized for their strong effectiveness and the relative ease with which such attacks can be launched due to the simplicity of edge perturbation. However, most existing structural attack approaches primarily

focus on maximizing attack effectiveness, often overlooking the substantial budget required to control the nodes needed to alter edges. For instance, in a social network, modifying an edge requires the adversary to control at least one of the connected nodes, which typically involves either purchasing or hijacking the account, and if numerous edges are to be perturbed, the adversary must control a large number of nodes, which not only incurs a significant financial cost but also heightens the risk of detection.

The challenge of executing a potent yet budget-efficient attack on GNNs represents a novel problem that has not been systematically explored in the existing literature. As outlined later in Chapter 2, both global structural attacks [1, 17, 24, 33, 40] and targeted structural attacks [5, 7, 12] necessitate control over a large number of nodes to achieve satisfactory attack performance, where the substantial resource requirement significantly undermines their cost-efficiency. In contrast, anchor nodes attacks [38, 39] offer more budget-efficient approachs, as they require control of only a small set of anchor nodes to target all nodes. However, due to the constrained effectiveness of relying on a single set of anchor nodes, existing anchor nodes attacks [38, 39] frequently fail to deliver strong attack performance.

In this work, we propose a budget-efficient Multi-Faceted Anchor Nodes (MFAN) attack against GNNs in the context of node classification tasks. The key concept is to train multiple sets of anchor nodes in conjunction with an assignment network, where each set of anchor nodes is specialized in effectively attacking a distinct subset of target nodes, and the assignment network then selects the optimal set of anchor nodes to attack each new target node. Through this "divide and conquer" strategy, MFAN achieves exceptional attack performance by targeting the collective union of nodes successfully compromised by each set of anchor nodes. Furthermore, MFAN demonstrates budget efficiency by requiring control over only a minimal number of anchor nodes. In summary, we make the following contributions.

- We introduce a novel adversarial attack task, which seeks to maximize attack effectiveness while minimizing the budget of controlled nodes.
- We successfully address the task by devising the Multifaceted Anchor Nodes (MFAN) attack, which exhibits both high strength and budget efficiency.
- We perform extensive experiments on five real-world datasets, empirically validating the exceptional performance of the proposed MFAN approach.

Chapter 2

Related Work

The multifaceted anchor nodes attack on graph neural networks presents a novel problem that has not been systematically addressed in prior research. Our approach, as it alters the edge structure of the input graph, aligns closely with structural attacks [1, 5, 7, 12, 17, 23, 35, 36, 38, 39, 40], which aim to degrade the classification performance of a victim graph neural network by perturbing (i.e., adding or removing) the edges within the input graph. The relationship between our work and existing literature is discussed in detail below.

2.1 Global Structural Attacks

Global structural attacks [1, 17, 23, 24, 33, 35, 40] seek to to diminish the overall classification accuracy of a victim model across all nodes in the graph by perturbing a substantial set of edges once and for all.

Meta-Self [43] leverages meta-learning to execute attacks by treating the graph's adjacency matrix as a hyper-parameter. [36] introduces two perturbation strategies utilizing first-order attack generation techniques. [17] conducts attacks on large-scale graphs by employing projected and greedy randomized block coordinate descent methods to sample the edges for perturbation. Several studies have also demonstrated effective global structural attacks by mitigating gradient bias [24], using Eigen decomposition [1], and employing a certified robustness-inspired framework [33].

These global structural attacks typically necessitate perturbing a considerable number of edges to attain strong attack performance. However, these approaches are not budget-efficient, as altering a large set of edges requires controlling a significant number of nodes, thereby imposing substantial budgetary demands.

2.2 Targeted Structural Attacks

Targeted structural attacks [5, 7, 12] aim to degrade the classification accuracy of the victim model on a single node by perturbing a subset of edges within the graph. FGA [7] exploits iterative gradient information between pairwise nodes obtained from a pre-trained graph convolutional neural network to generate edge perturbations.

GF-Attack [5] employs graph embedding techniques in conjunction with a corresponding graph filter to produce edge perturbations. RL-S2V [12] utilizes hierarchical reinforcement learning to induce edge perturbations, thereby significantly reducing the prediction accuracy of the victim model.

These targeted structural attacks generate distinct edge perturbations for each specific target node. As perturbing edges necessitates control over the nodes connected to those edges, initiating a new attack by altering a different set of edges requires gaining control over a new set of nodes, and given that controlling each additional node incurs a proportional budgetary cost, these targeted structural attacks are not budget-efficient. Consequently, when the budget is constrained, these methods are unable to control a sufficient number of new nodes to effectively attack a large number of target nodes.

2.3 Anchor Nodes Attacks

Anchor nodes attacks [38, 39] represent a distinct category of structural attacks. These methods first identify a fixed set of nodes, referred to as anchor nodes, and subsequently degrade the classification performance of the victim model on each target node by flipping edges between the anchor nodes and the target node, where flipping an edge refers to either adding a non-existent edge or removing an existing one.

As the pioneering work in this domain, GUA [39] iteratively identifies a set of anchor nodes through a minimum perturbation strategy. The follow-up work, GUAP [38], introduces a set of new nodes, utilizing them as anchor nodes to launch the attack.

Both GUA and GUAP are budget-efficient, as they require control over only a small, fixed set of anchor nodes to attack all target nodes. However, their attack efficacy is significantly constrained by the limitation of relying on a single set of anchor nodes for all attacks. Since victim graph neural network models make predictions for a target node based on its multi-hop neighbors, numerous target nodes may fall outside the effective attack range of the anchor nodes if they do not reside within close proximity in the network. Our proposed multifaceted anchor node attack distinguishes itself from GUA and GUAP by smoothly integrating multiple sets of anchor nodes, alongside a well-trained assignment network, to substantially enhance attack effectiveness

while maintaining a minimal budget of controlled nodes.

2.4 Graph Attacks with Other Strategies

Other than solely changing the graph structure, there are works that our approach is fundamentally distinct from, including works that focus on modifying node features [2, 8, 30, 42] or injecting malicious nodes [11, 19, 29, 31, 34, 41].

Nettack[42], the pioneering work on adversarial attacks on graphs, generates perturbations on both node features and graph structure. GAALV[2], in contrast, keeps the graph structure intact and focuses exclusively on perturbing node features. PoisonProbe[30] targets the victim node by selecting nodes within a 2-hop radius and perturbing their features. TUA[13], which aims to misclassify victim nodes into a specific target class, randomly selects nodes with the target class label, referred to as attack nodes, followed by generating a set of fake nodes with the same target class label and connects the victim node to the attack nodes during the attack. G-NIA[31] performs its attacks by injecting a single node into the graph and linking it to other nodes within the graph. Greedy-GAN[34] introduces fake nodes into the original graph and connects them to existing nodes without altering any existing edges or features.

Chapter 3

Preliminary

3.1 Graph

Let $G = (V, E, A, X)$ represent an unweighted graph, where V denotes the set of nodes, E the set of edges, and $N = |V|$ represents the total number of nodes in G. The adjacency matrix $A \in \{0, 1\}^{N \times N}$ encodes the edge structure of G. Each node in G is associated with a d-dimensional feature vector, and the feature vectors of all nodes are collectively represented by the feature matrix $X \in \mathbb{R}^{N \times d}$, where the i-th row of X corresponds to the feature vector of the i-th node, v_i, in G. Furthermore, each node in G is assigned a class label from one of C possible classes.

3.2 Graph Convolutional Network

A graph neural network, denoted by $f(X, A)$, takes as input the feature matrix X and the adjacency matrix A of graph G and outputs predictions for the class labels of the nodes in G.

In accordance with literature [7, 36, 38, 39, 42, 43], we target a well-known graph neural network, the graph convolutional network (GCN) [20], as the victim model for our attack. A typical GCN comprises one or more hidden graph convolutional layers, followed by a softmax layer to generate the final predictions. In the well-established node classification GCN model that we seek to attack, the hidden graph convolutional layer is denoted as

$$H^{(l+1)} = \sigma(\hat{A} H^{(l)} W^{(l)}), \tag{3.2.1}$$

where l is the number of convolution layers, $\sigma(\cdot)$ is the activation function ReLU [16], and $\hat{A} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$ is a normalized adjacency matrix with $\tilde{A} = A + I$ and diagonal degree matrix $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. With the addition of the final softmax layer, a typical GCN with a single hidden graph convolutional

layer is represented by

$$f(X, A) = \texttt{softmax}(\hat{A}\ \texttt{ReLU}(\hat{A}XW^{(0)})\ W^{(1)}), \tag{3.2.2}$$

where $W^{(0)}$ and $W^{(1)}$ are the model parameters of the GCN.

For simplicity, we denote $f(X, A)$ as f when the context is clear. The output of the GCN f, denoted by

$$Z = f(X, A), \tag{3.2.3}$$

is a matrix with dimensions of N rows and C columns, where the entry in the i-th row and j-th column, denoted by Z_{ij}, represents the probability of the i-th node being classified into the j-th class. Additionally, the i-th row of Z can be denoted as $f(X, A)_i$ for simplicity.

3.3 Victim Model Types

In the field of adversarial attacks, there are two distinct types of victim models characterized by different levels of access an attacker has to the model's internal parameters.

- *White-Box Model*. The attacker has full access to the model's architecture, parameters, and internal mechanisms, including gradients, weights, and biases, which allows the attacker to exploit the model's vulnerabilities by directly manipulating its parameters or using gradient-based methods to generate adversarial examples.

- *Black-Box Model*. The attacker has no direct access to the model's internal structure or parameters, and it can only interact with the model by observing its input-output behavior, such as querying the model with inputs and observing the corresponding outputs.

We adopt the same white-box setting in [38, 39] wherein an adversary has complete access to the architecture and parameters of the victim model. However, as demonstrated later in Section 6.3, under the black-box setting, the attack model trained by MFAN on a white-box victim model can be easily transferred to effectively attack other black-box models.

3.4 Attacking Types

To launch an adversarial attack, modifications can be made to the graph to mislead the victim GNN model's prediction. Depending on the stage at which

the graph attacks occur, these attacks can be categorized into two distinct types.

- *Evasion Attack*. Graphs are modified after the victim model has been trained. At this stage, the parameters of the victim model are fixed and cannot be altered by the attacker. The attacker's objective is to degrade the classification performance of the victim model or induce it to predict labels that differ from its original predictions.
- *Poisoning Attack*. Graphs are modified (poisoned) before the victim model is trained. Thus, the parameters of the victim model is trained and tuned on the modified graph. The attacker's objective is to degrade the classification performance of the victim model.

We employ the evasion attack scenario in our setting, with the objective of inducing the victim model to produce outputs that deviate from its original predictions.

Chapter 4

Problem Definition

4.1 MFAN Task

In the task of multifaceted anchor nodes (MFAN) attack, our objective is to target a victim model, graph convolutional network (GCN) f, which is trained on an unweighted graph G. We conduct the attack by perturbing (i.e., modifying) the edges in G to create a modified graph G', such that the label of a target node v_i, predicted by the victim GCN f on G', differs from the label predicted by f on the original graph G.

In line with the approaches of [38, 39], the perturbation on G is a set of edge modifications, (i.e., additions or deletions of edges), induced by a set of nodes in G, named **anchor nodes**. Denote by $Q \subset V$ a set of anchor nodes, a **perturbation** induced by Q to attack f's prediction on a target node v_i is to flip the edges between each node in Q and v_i. Here, **flip** refers to either adding a non-existent edge or removing an existing edge. Furthermore, for brevity, we refer to "change f's prediction on a target node v_i" as "attack the target node v_i".

In the framework of MFAN, our objective is to train K distinct sets of anchor nodes, represented by the collection $\mathcal{Q} = \{Q_1, \ldots, Q_K\}$, where each set of anchor nodes $Q_k \in \mathcal{Q}$ optimized to effectively attack a substantial subset of target nodes in V. In conjunction with the training of $\mathcal{Q}$, we also train an **assignment network**, denoted by g_θ, which is responsible for selecting the most appropriate set of anchor nodes from $\mathcal{Q}$ to attack a given target node v_i.

The MFAN task is formally defined as follows.

Definition 1. *Given an integer budget $\xi > 0$ on the number of controlled nodes for each perturbation and a victim GCN f trained on an unweighted graph G, the task of **multifaceted anchor nodes (MFAN) attack** is to train an attack model, composed by K sets of anchor nodes $\mathcal{Q} = \{Q_1, \ldots, Q_K\}$ and an assignment network g_θ, such that*

1. *the size of each set of anchor nodes is not larger than ξ;*

2. *each target node $v_i \in V$ is attacked by the perturbation induced by the set of anchor nodes $Q_k \in \mathcal{Q}$ that is selected by the assignment network g_θ; and*

3. *our attack model can successfully attack most of the nodes in G.*

In contrast to classic anchor node attacks [38, 39], the MFAN attack is multifaceted, as it utilizes multiple sets of anchor nodes. The assignment network, g_θ, takes the graph G and the target node v_i as input to select the best-suited set of anchor nodes, maximizing the likelihood of successfully attacking v_i. Since each set of anchor nodes is specialized in targeting different subsets of nodes, the MFAN attack effectively implements a "divide and conquer" strategy, allowing it to attack the union of target nodes that are individually targeted by each set, which substantially enhances the attack's overall effectiveness. The budget ξ constrains the number of controlled nodes for each set of anchor nodes, and since the same sets of anchor nodes are used to attack all target nodes in G, the number of nodes that need to be controlled remains minimal, rendering MFAN highly budget-efficient.

4.2 Modelling the MFAN Attack

We present in this section the modelling of the perturbation and the assignment network g_θ. A example of the attack procedure of MFAN is shown in Figure 4.1.

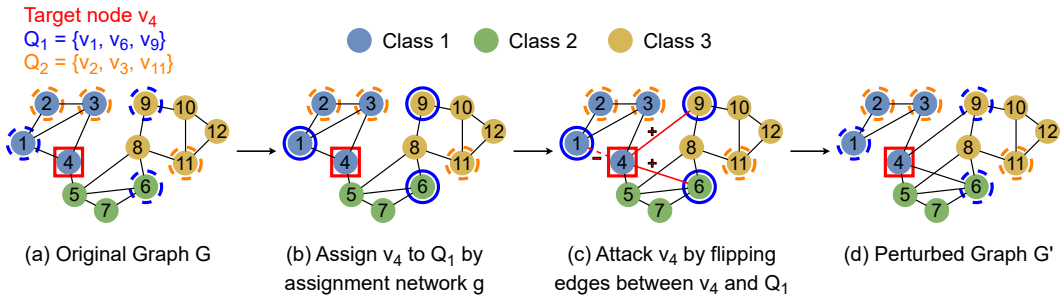


Figure 4.1: An example of MFAN attack. To attack the target node v_4 in the original graph G, the assignment network selects the anchor nodes set $Q_1 = \{v_1, v_6, v_9\}$. Then it flips the edges between v_4 and Q_1, generating the perturbed graph G'.

4.2.1 Modelling of the Perturbation

Given a set of anchor nodes $Q_k \in \mathcal{Q}$, MFAN attacks the target node v_i by introducing a perturbation to G, which involves flipping the edges between v_i and each anchor node in Q_k. To mathematically formalize this perturbation, we represent the set of anchor nodes in Q_k by a perturbation vector $\mathbf{p}_k \in \{0,1\}^N$, where the i-th element of $\mathbf{p}_k$ being equal to 1 indicates the i-th node v_i in G is an anchor node in Q_k. Thus, the sets of anchor nodes in $\mathcal{Q}$ are modeled by the set of perturbation vectors $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_K\}$. Following the approaches [38, 39], we model the **perturbation** induced by a perturbation vector $\mathbf{p}_k \in \mathcal{P}$ to attack v_i as

$$\rho(v_i, \mathbf{p}_k) = (\mathbf{1} - P) \circ A + P \circ (\mathbf{1}_0 - A), \tag{4.2.1}$$

where ρ is the perturbation function, $\circ$ is element-wise multiplication, $\mathbf{1}$ is an N-by-N matrix with all entries equal to one, $\mathbf{1}_0$ is an N-by-N matrix of all entries equal to one except for the diagonal entries being zeros, and P is an N-by-N matrix with i-th row and i-th column replaced by $\mathbf{p}_k$ and the other entries set to zero.

The output of $\rho(v_i, \mathbf{p}_k)$, denoted by $A' = \rho(v_i, \mathbf{p}_k)$, represents the **perturbed adjacency matrix** of the **perturbed graph** G', in which the edges between v_i and each anchor node in Q_k in the original graph G have been flipped.

4.2.2 Modelling of the Assignment Network

The **assignment network** is modeled as a GCN, denoted by $g_\theta(X, A)$, which predicts the probabilities of selecting each of the K sets of anchor nodes in $\mathcal{Q}$ to attack a given target node v_i in G. The output of $g_\theta(X, A)$ is an N-by-K matrix, where the entry in the i-th row and the k-th column, denoted by $g_\theta(X, A)_{ik}$, represents the probability of selecting the anchor nodes set $Q_k \in \mathcal{Q}$ to attack the target node v_i in G.

4.3 Formulating the Problem

The loss of the MFAN attack is mathematically formulated as

$$\mathcal{L}(\mathcal{P}, \theta) = -\sum_{i=1}^{N} \sum_{k=1}^{K} g_\theta(X, A)_{ik} \cdot \texttt{CE}(f(X, A')_i, f(X, A)_i), \tag{4.3.1}$$

where $\texttt{CE}(\cdot, \cdot)$ is the cross entropy loss, $A' = \rho(v_i, \mathbf{p}_k)$ is the perturbed adjacency matrix induced by the perturbation vector $\mathbf{p}_k \in \mathcal{P}$, f is the victim model, and

$f(X, A')_i$ and $f(X, A)_i$ are the predicted class distributions of the target node v_i on the perturbed graph G' and the original graph G, respectively. The cross entropy loss term quantifies the difference between $f(X, A')_i$ and $f(X, A)_i$ when $\mathbf{p}_k$ is used to attack the target node v_i. Given that $g_\theta(X, A)_{ik}$ represents the probability of selecting $\mathbf{p}_k$ to attack v_i, the summation term $\sum_{k=1}^{K}$ computes the expectation of the difference between $f(X, A')_i$ and $f(X, A)_i$. According to the task defined in Definition 1, our objective is to successfully attack most of the nodes in G, which is accomplished by maximizing the expected difference between between $f(X, A')_i$ and $f(X, A)_i$ across all the nodes in G.

Therefore, the MFAN attack is further formulated as the following optimization problem.

$$\min_{\mathcal{P},\theta} \mathcal{L}(\mathcal{P}, \theta) \quad \text{s.t. } \forall \mathbf{p}_k \in \mathcal{P}, ||\mathbf{p}_k||_1 \leq \xi, \mathbf{p}_k \in \{0, 1\}^N, \tag{4.3.2}$$

where the constraint $||\mathbf{p}_k||_1 \leq \xi$ ensures that the number of anchor nodes in each set $Q_k \in \mathcal{Q}$ does not exceed the specified budget ξ.

Chapter 5

Solution

5.1 Solving the Problem

The original optimization problem in Equation (4.3.2) is a constrained integer programming problem, which is NP-hard and cannot be directly solved using gradient-based methods. We present the solution in steps.

5.1.1 Relaxation

Each integer-valued constraint $\mathbf{p}_k \in \{0,1\}^N, \mathbf{p}_k \in \mathcal{P}$ is relaxed to a real-valued constraint $\mathbf{p}_k \in [0,1]^N$, thereby transforming the original optimization problem in Equation (4.3.2) to

$$\min_{\mathcal{P},\theta} \mathcal{L}(\mathcal{P},\theta) \quad \text{s.t. } \forall \mathbf{p}_k \in \mathcal{P}, ||\mathbf{p}_k||_1 \leq \xi, \mathbf{p}_k \in [0,1]^N. \tag{5.1.1}$$

5.1.2 Penalty

Following the standard penalty method [3, 15, 28], we incorporate each constraint $||\mathbf{p}_k||_1 \leq \xi$ as a penalty term, $\max(||\mathbf{p}_k||_1 - \xi, 0)$, within the loss function, thereby transforming the optimization problem in Equation (5.1.1) to

$$\min_{\mathcal{P},\theta} \mathcal{L}(\mathcal{P},\theta) + \lambda \sum_{\mathbf{p}_k \in \mathcal{P}} \max(||\mathbf{p}_k||_1 - \xi, 0) \quad \text{s.t. } \forall \mathbf{p}_k \in \mathcal{P}, \mathbf{p}_k \in [0,1]^N. \tag{5.1.2}$$

5.1.3 Simulated Annealing

Under the relaxed constraint $\mathbf{p}_k \in [0,1]^N$, the entries in each solution $\mathbf{p}_k \in \mathcal{P}$ may be real values that deviate significantly from 0 or 1, solving the optimization problem in Equation (5.1.2) often fails to find optimal K sets of anchor nodes. However, directly quantizing (i.e., binarizing) these entries in $\mathbf{p}_k$ to 0

or 1 introduces substantial quantization error, hence diminishing the quality of the final solution.

To address this challenge, we propose a **simulated annealing** technique that forces the real-valued entries in each solution $\mathbf{p}_k \in [0,1]^N$ to converge toward either 0 or 1, which effectively minimizes quantization error when quantizing a solution $\mathbf{p}_k$ to a binary vector in $\{0,1\}^N$, thereby enhancing the quality of the final solution. Specifically, we construct a **weighted penalty term** to reformulate Equation (5.1.2) as

$$\min_{\mathcal{P},\theta} \mathcal{L}(\mathcal{P},\theta) + \lambda \sum_{\mathbf{p}_k \in \mathcal{P}} \max(||\mathbf{w}_k \circ \mathbf{p}_k||_1 - \mathbf{w}_k^{min}\xi, 0) \quad \text{s.t. } \forall \mathbf{p}_k \in \mathcal{P}, \mathbf{p}_k \in [0,1]^N, \tag{5.1.3}$$

where $\circ$ is element-wise multiplication, $\mathbf{w}_k$ is an weight vector of size N calculated from $\mathbf{p}_k$, and $\mathbf{w}_k^{min}$ is the smallest value among all the entries in $\mathbf{w}_k$. The h-th entry of $\mathbf{w}_k$ is computed by the annealing function

$$\mathbf{w}_k^h = \sigma(\mathbf{p}_k^h) = \frac{1}{1 + e^{(\mathbf{p}_k^h - \delta)/T}}, \tag{5.1.4}$$

where $\mathbf{p}_k^h$ is the h-th entry in $\mathbf{p}_k$, δ is the mean of the ξ-th and $(\xi+1)$-th largest entry in $\mathbf{p}_k$, and T is a hyperparameter that controlls the "temperature" of the annealing process. A smaller value of T causes the annealing function σ to approximate a step function, assigning weights close to 1 to the top-ξ largest entries in $\mathbf{p}_k$ and weights close to 0 to the remaining entries. Consequently, by progressively reducing the value of T, we can drive the top-ξ largest entries in $\mathbf{p}_k$ to 1 while pushing the other entries toward 0.

Equation (5.1.3) is the formal form of our optimization problem, which can be solved using standard proximal gradient descent [26]. A feasible solution to Equation (5.1.3) is also a feasible solution to Equation (5.1.1), as demonstrated by the following rationale. During the minimization of the objective function in Equation (5.1.3) following [3, 15, 28], λ is gradually increased to push the penalty terms toward zero, which ensures that each feasible solution $\mathbf{p}_k \in \mathcal{P}$ to Equation (5.1.3) satisfies the constraint $||\mathbf{w}_k \circ \mathbf{p}_k||_1 \leq \mathbf{w}_k^{min}\xi$. We then have $||\mathbf{p}_k||_1 \leq \xi$, as demonstrated in Theorem 1, indicating that $\mathbf{p}_k$ is a feasible solution to Equation (5.1.1).

Theorem 1. *$\forall \mathbf{p}_k \in \mathcal{P}, \mathbf{p}_k \in [0,1]^N$, if $||\mathbf{w}_k \circ \mathbf{p}_k||_1 \leq \mathbf{w}_k^{min}\xi$, then $||\mathbf{p}_k||_1 \leq \xi$.*

Proof. Since $||\mathbf{w}_k \circ \mathbf{p}_k||_1 \leq \mathbf{w}_k^{min}\xi$, we have $\xi \geq ||\frac{\mathbf{w}_k}{\mathbf{w}_k^{min}} \circ \mathbf{p}_k||_1 \geq ||\mathbf{p}_k||_1$. □

As Equation (5.1.1) represents a relaxed version of the original optimization problem in Equation (4.3.2), a **final solution** to Equation (4.3.2) can be obtained by quantizing each feasible solution $\mathbf{p}_k \in \mathcal{P}$ into a binary vector in $\{0,1\}^N$. Since the simulated annealing trick forces the top-ξ largest entries

in $\mathbf{p}_k$ approach 1 and the other entries approach 0, the solution for $\mathbf{p}_k$ is ensured to approximate a binary vector, which minimizes the quantization error, therefore the final solution quantized from $\mathbf{p}_k$ satisfies the budget of controlled nodes.

5.2 Algorithms

5.2.1 Training Algorithm

Algorithm 1: Training $\mathcal{P}$ and θ

Input : ξ, f, G, and a training set of nodes $V_T \subset V$
Output: $\mathcal{P}$ and θ

1 $T \leftarrow 1$, $\lambda \leftarrow 0.1$ and $max\_epoch \leftarrow 120$
2 Randomly initialize $\mathcal{P} \leftarrow \mathcal{P}^{(0)}$ and $\theta \leftarrow \theta^{(0)}$
3 **while** *epoch* $< max\_epoch$ **do**
4 **for** *each mini-batch in* V_T **do**
5 **for** *each* $\mathbf{p}_k \in \mathcal{P}$ **do**
6 $\mathbf{p}_k \leftarrow \mathbf{p}_k - \eta_1 \nabla_{\mathbf{p}_k} \mathcal{L}^*(\mathcal{P}, \theta)$
7 $\mathbf{p}_k \leftarrow \texttt{clip}(\mathbf{p}_k, 0, 1)$
8 **end**
9 $\theta \leftarrow \theta - \eta_2 \nabla_\theta \mathcal{L}^*(\mathcal{P}, \theta)$
10 **end**
11 Update $\lambda \leftarrow \lambda \times 5$ for every 20 epochs when *epoch* $\leq max\_epoch/2$.
12 Update $T \leftarrow T/2$ for every 5 epochs when *epoch* $> max\_epoch/2$.
13 **end**
14 $\mathcal{P} \leftarrow \texttt{quantize}(\mathcal{P}, \xi)$
15 **return** $\mathcal{P}$ *and* θ

We describe the process for training $\mathcal{P}$ and θ by solving the optimization problem presented in Equation (5.1.3). Algorithm 1 summarizes the details to train $\mathcal{P}$ and θ, where $\mathcal{L}^*(\mathcal{P}, \theta)$ denotes the objective function in Equation (5.1.3).

In Line 6 and line 9, standard gradient steps are conducted to update $\mathbf{p}_k$ and θ, respectively, where η_1 and η_2 denote learning rates. In line 7, the function $\texttt{clip}(\mathbf{p}_k, 0, 1)$ is applied, which performs a proximal projection of standard proximal gradient descent [26], ensuring that any entry in $\mathbf{p}_k$ falling outside the range $[0, 1]$ is clipped back to its nearest boundary value within $[0, 1]$. In Line 11, the weight λ of the penalty term is gradually increased. In Line 12, the simulated annealing makes the annealing function σ approximate

a step function by decreasing the temperature parameter T. In line 14, the perturbation vector $\mathbf{p}_k$ is quantized to obtain the final solution.

5.2.2 Attacking Algorithm

Algorithm 2: Attacking a target node v_i in G

Input : G, v_i, $\mathcal{P}$, and g_θ
Output: A perturbed adjacency matrix A'

1 $k^* \leftarrow \arg\max_k g_\theta(X, A)_{ik}$
2 $A' \leftarrow \rho(v_i, \mathbf{p}_{k^*})$
3 **return** A'

Once $\mathcal{P}$ and θ are trained, they can be used to launch the MFAN attacks. Algorithm 2 summarizes the details to attack a target node v_i in G.

In line 1, the assignment network g_θ is employed to select the most suitable set of anchor nodes to attack v_i, denoted as $\mathbf{p}_{k^*}$. In line 2, the selected $\mathbf{p}_{k^*}$ is used to attack v_i by flipping the edges between v_i and each of the anchor nodes represented by $\mathbf{p}_{k^*}$. Since g_θ is trained in conjunction with $\mathcal{P}$, it effectively generalizes to select the most suitable set of anchor nodes, thereby significantly enhancing the success rate of the attacks.

Chapter 6

Experiments

We conduct extensive experiments in this chapter to compare our method against six baseline methods for adversarial graph structural attacks, including GUA [39][1], GUAP [38][2], PGD [36], DICE [35], Meta-Self [43] and FGA [7][3]. We utilize the publicly available source code of these baselines, with their default parameter configurations employed in all experiments. Our code is accessible at the following link[4].

6.1 Experimental Setting

In accordance with existing literature [7, 36, 38, 39, 42, 43], we employ the classic GCN[20], introduced in Chapter 3, as the default victim model for the attacks. Additionally, we use GAT [32] and Node2Vec [18] as the black-box victim models for the transfer attacks.

Each experiment is independently repeated 5 times to report the average performance. All experiments are conducted on a server equipped with an NVIDIA RTX 3090 GPU, 64 GB of RAM, and an Intel(R) Core(TM) i9-10900K CPU @ 3.70GHZ.

6.1.1 Implementation Details

For MFAN, the assignment network g_θ is implemented using the GCN, as defined in Equation (3.2.2). The model configuration of g_θ is detailed in Table 6.1, where the graph convolution layer performs mean aggregation, and the "Input Dim.", "Weight Dim." and "Output Dim." correspond to the dimensions of $H^{(l)}$, $W^{(l)}$ and $H^{(l+1)}$ for each layer l, as defined in Equation (3.2.1).

[1]Code: `https://github.com/chisam0217/Graph-Universal-Attack`

[2]Code: `https://anonymous.4open.science/r/ffd4fad9-367f-4a2a-bc65-1a7fe23d9d7f/`

[3]Code for PGD, DICE, Meta-Self and FGA: `https://github.com/DSE-MSU/DeepRobust`

[4]Code for MFAN: `https://github.com/zhz0108/mfan`

Table 6.1: Model configuration of the assignment network.

Layer	Type	Input Dim.	Weight Dim.	Output Dim.	Activation
1	Graph Convolution	$N \times d$	$d \times 16$	$N \times 16$	ReLU
2	Graph Convolution	$N \times 16$	$16 \times K$	$N \times K$	Softmax

In the implementation of Algorithm 1 of MFAN, we use a batch size of 32, with η_1 set to 0.01 and η_2 set to 0.2 for the Facebook dataset (to be discussed later) and 0.005 for the other datasets. We set $\xi = 5$ and $K = 2$ by default, unless otherwise specified. The impact of K on the attack effectiveness of MFAN will be thoroughly examined in Section 6.4.1.

6.1.2 Datasets

Table 6.2: Dataset statistics.

Dataset Statistics	Cora	Citeseer	Facebook	Wiki	Pubmed
#Nodes	2,708	3,327	4,039	2,405	19,717
#Edges	5,278	4,676	88,234	17,981	44,324
#Features	1,433	3,703	1,283	4,973	500
#Classes	7	6	193	17	3

The experiments are conducted on five widely utilized benchmark datasets for node classification tasks, as detailed in Table 6.2. Cora [27], Citeseer [27] and Pubmed [27] are scientific publication networks, Facebook [22] is a social network, and Wiki [10] is a network of web pages with their hyperlinks as edges.

For the largest dataset, Pubmed, a sampled subgraph comprising 2,000 nodes rather than full graph is extracted where the training is conducted exclusively, while the remaining nodes are designated as target nodes for testing. For the other datasets, we adhere to the setting of [5, 7, 29, 30, 42], where the training is performed on the entire graph, with 20% of the nodes allocated for training and the remaining 80% of the nodes designated as target nodes for testing.

FGA was unable to complete the attack on all testing target nodes on Pubmed within 72 hours. Consequently, we cannot provide the corresponding results for this method.

6.1.3 Evaluation Metrics

The attack performance on misclassification is evaluated by *fooling ratio* (FR), as defined by

$$\text{FR} = \frac{\#\text{ of misclassified nodes in test set}}{\#\text{ of nodes in test set}} \tag{6.1.1}$$

Given the diverse attack paradigms employed by the baseline methods, we propose two distinct measures of the *budget of controlled nodes* to provide a comprehensive evaluation of their performance.

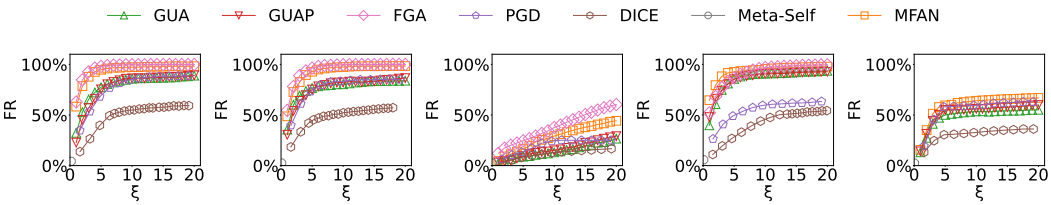
- The first type of budget, namely *budget per target node* (BPT), denoted by ξ, represents the number of controlled nodes used in attacking a single target node. For GUA and GUAP, since they use the same set of anchor nodes to attack each target node, ξ corresponds to the number of anchor nodes. For MFAN, since we use only one set of anchor nodes to attack each target node, ξ precisely aligns with the budget in Definition 1. For the remaining baselines, ξ is the average number of nodes connected to a target node by a perturbed edge, either prior to or following the attack.
- The second type of budget, namely *budget for all target nodes* (BFA), denoted by δ, represents the total number of all controlled nodes used to attack all the target nodes in the testing dataset. For GUA and GUAP, $\delta = \xi$. For MFAN, $\delta = K \times \xi$. For FGA, δ denotes the size of the union of nodes connected to perturbed edges during each attack. For the remaining baselines, δ represents the total number of nodes connected to a target node by a perturbed edge, either prior to or following the attack.

6.2 Fooling Ratio under Different Budgets of Controlled Nodes

In this section, we perform an analysis of the FR of all compared methods under different budgets of controlled nodes.

6.2.1 Fooling Ratio under Budget Per Target Node

Figure 6.1 illustrates how the FR of each method changes when using different BPT (i.e., ξ). The FR of all methods increases as ξ grows, since a larger BPT allows more edges to be perturbed in each attack, thus enhancing the likelihood of success. FGA achieves the highest FR on Cora, Citeseer, and Facebook when using the same BPT as the other methods due to FGA tailored approach, where it meticulously trains a unique set of controlled nodes for each new

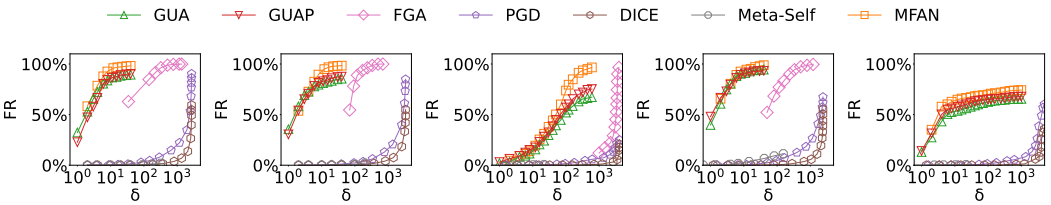


(a) Cora (b) Citeseer (c) Facebook (d) Wiki (e) Pubmed

Figure 6.1: Fooling ratio (FR) v.s. budget per target node (BPT, ξ).

target node, significantly bolstering the success rate of its attacks. However, as we will explore later, this strategy comes at the cost of a substantially higher BFA, as each new target requires the orchestration of a distinct set of controlled nodes. The other baseline methods, by contrast, cannot rival the FR of FGA because they rely on a single set of controlled nodes for all attacks, a static configuration not fine-tuned for each specific target. Interestingly, while MFAN does not custom-train anchor nodes for every new target, its FR still comes closest to that of FGA across all datasets. This strongly attests to MFAN's remarkable efficacy and the sophistication of its attack strategy, nearly matching the precision of FGA despite its more generalized approach.

6.2.2 Fooling Ratio under Budget For All Target Nodes



(a) Cora (b) Citeseer (c) Facebook (d) Wiki (e) Pubmed

Figure 6.2: Fooling ratio (FR) v.s. budget for all target nodes (BFA, δ).

Figure 6.2 illustrates how the FR of each method changes when using different BFA (i.e., δ). While FGA leads in performance when using BPT as the budget for controlled nodes, its FR is significantly lower compared to the anchor node attacks (i.e., GUA, GUAP, and MFAN) when BFA is used as the budget. The minimum BFA for FGA by setting $\xi = 1$ is still much higher than the BFA of anchor node attacks because FGA trains a unique set of controlled nodes for each new target node, requiring control over a large number of nodes to attack the thousands of target nodes in the test dataset. Similarly, global structural attacks (i.e., DICE, PGD, and Meta-Self) also incur a significantly

high BFA because they perturb a large number of edges in one go for all attacks. In contrast, the anchor nodes attacks are extremely efficient in BFA since they only use a small constant set(s) of anchor nodes to attack all the target nodes. Notably, MFAN performs best across all datasets, thanks to its effective multifaceted attack approach with successful application of the "divide and conquer" principle.

In conclusion, when the total number of controlled nodes is constrained by available resources (e.g., financial costs, personnel, etc.), MFAN demonstrates superior FR performance by effectively employing the "divide and conquer" strategy. Additionally, MFAN attacks are considerably more covert compared to non-anchor node methods, owing to its minimal BFA, further enhancing its stealthiness.

6.3 Effectiveness of Transfer Attack

In this section, we evaluate the effectiveness of the transfer attacks across all the baseline methods and MFAN.

For each method, we first train its attack model on the white-box victim model GCN f. Afterward, we apply the trained attack model to attack two additional black-box victim models, namely GAT [32] and Node2Vec [18], which are trained on the same dataset as the white-box victim model GCN f.

Table 6.3 presents the FR of transfer attacks on the five datasets, respectively, with the best FR highlighted in bold and the second-best underlined. The performance of Meta-Self is not reported with the given BPT values (i.e., ξ) since it could not be executed due to the requirement of high memory usage. FGA failed to generate meaningful results with small BFA (i.e., δ), hence its corresponding results are also not provided.

It can be observed that MFAN delivers the best performance in most cases, demonstrating its exceptional capability in transfer attacks against black-box victim models. We attribute this strong performance to the following reasons.

- Each set of anchor nodes performs effectively in transfer attacks. Because each set of anchor nodes is trained to successfully attack a large group of target nodes, it tends to exploit common defect patterns shared by many of these nodes. These defect patterns are prevalent across various target nodes, meaning that they can be learned by a new GNN model trained on the same dataset. As a result, the anchor nodes remain effective in carrying out transfer attacks against the new model.

- The assignment network also performs effectively in transfer attacks. As the input graph remains unchanged, the output of the assignment network stays consistent. Consequently, the same set of anchor nodes is

Table 6.3: FR of non-transfer and transfer attacks.

Dataset	Model	GCN (white-box, non-transfer)				GAT (black-box, transfer)				Node2Vec (black-box, transfer)			
Cora	Budget	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$
	GUA	77.37%	85.75%	88.83%	89.96%	81.39%	86.19%	88.74%	90.03%	76.92%	84.85%	89.33%	88.60%
	GUAP	75.32%	86.20%	89.25%	90.25%	65.56%	85.78%	86.54%	91.89%	71.64%	85.86%	87.04%	87.11%
	PGD	71.88%	85.33%	0.74%	2.07%	74.72%	83.86%	0.93%	1.80%	72.04%	70.89%	15.53%	15.77%
	DICE	44.30%	54.99%	0.05%	0.10%	51.03%	51.77%	0.11%	0.35%	79.88%	80.24%	15.83%	16.21%
	Meta-Self	-	-	0.02%	0.06%	-	-	0.59%	1.17%	-	-	15.84%	16.04%
	FGA	**95.00%**	**98.63%**	-	-	**85.44%**	92.95%	-	-	62.81%	82.14%	-	-
	MFAN	93.79%	94.30%	**94.30%**	**96.11%**	84.67%	**93.04%**	**93.04%**	**94.59%**	**85.14%**	**92.17%**	**92.17%**	**96.01%**
Citeseer	Budget	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$
	GUA	76.63%	80.35%	83.83%	84.49%	74.63%	81.51%	84.10%	85.12%	86.41%	86.56%	86.70%	86.94%
	GUAP	75.17%	82.20%	86.45%	87.65%	66.87%	79.69%	83.19%	89.24%	76.89%	81.40%	83.18%	84.13%
	PGD	75.67%	83.80%	0.16%	0.34%	77.92%	80.42%	0.64%	1.33%	79.58%	79.45%	37.14%	37.45%
	DICE	45.30%	48.96%	0.03%	0.05%	46.73%	48.90%	0.16%	0.33%	79.06%	79.79%	37.80%	37.89%
	Meta-Self	-	-	0.04%	0.06%	-	-	0.44%	0.72%	-	-	37.22%	38.05%
	FGA	**96.27%**	**98.70%**	-	-	84.45%	92.72%	-	-	73.93%	86.29%	-	-
	MFAN	92.37%	97.31%	**97.31%**	**98.45%**	**90.21%**	**95.78%**	**95.78%**	**97.86%**	**91.79%**	**93.09%**	**93.09%**	**94.65%**
Facebook	Budget	$\xi = 10$	$\xi = 20$	$\delta = 200$	$\delta = 400$	$\xi = 10$	$\xi = 20$	$\delta = 200$	$\delta = 400$	$\xi = 10$	$\xi = 20$	$\delta = 200$	$\delta = 400$
	GUA	13.12%	26.60%	60.44%	66.02%	13.45%	16.63%	50.78%	58.23%	10.24%	15.47%	54.19%	61.77%
	GUAP	15.12%	29.10%	68.69%	73.67%	16.59%	26.68%	64.35%	70.23%	14.96%	20.44%	59.40%	65.23%
	PGD	24.12%	24.96%	3.24%	5.47%	31.16%	33.67%	2.74%	5.45%	16.85%	19.43%	10.25%	10.68%
	DICE	12.99%	16.90%	0.26%	0.66%	15.24%	19.05%	0.74%	1.21%	19.05%	25.93%	10.07%	10.11%
	Meta-Self	-	-	1.33%	2.16%	-	-	1.98%	4.16%	-	-	10.10%	10.33%
	FGA	**37.68%**	**49.40%**	-	-	30.33%	**49.14%**	-	-	21.23%	34.87%	-	-
	MFAN	27.36%	39.53%	**88.37%**	**94.94%**	**33.49%**	38.50%	**76.54%**	**85.53%**	**38.50%**	**58.53%**	**83.39%**	**87.52%**
Wiki	Budget	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$
	GUA	85.58%	90.85%	93.68%	94.35%	42.70%	45.90%	50.06%	56.72%	30.81%	44.99%	54.47%	74.18%
	GUAP	85.40%	90.65%	92.73%	93.11%	39.85%	48.76%	50.81%	66.59%	21.87%	43.08%	51.52%	67.44%
	PGD	52.01%	60.01%	0.81%	2.06%	56.37%	**67.92%**	0.68%	1.11%	42.55%	46.15%	14.74%	15.20%
	DICE	30.78%	44.94%	0.13%	0.35%	39.58%	63.42%	0.26%	0.31%	46.63%	62.37%	14.64%	15.45%
	Meta-Self	-	-	4.66%	6.69%	-	-	0.70%	1.27%	-	-	15.10%	15.71%
	FGA	86.74%	**96.84%**	-	-	52.81%	63.24%	-	-	44.23%	**72.45%**	-	-
	MFAN	**92.55%**	95.52%	**95.52%**	**97.82%**	**62.93%**	67.07%	**67.07%**	**73.31%**	**47.78%**	54.97%	**54.97%**	**82.70%**
Pubmed	Budget	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$	$\xi = 5$	$\xi = 10$	$\delta = 20$	$\delta = 40$
	GUA	50.04%	53.50%	55.11%	59.42%	47.33%	48.54%	51.03%	55.45%	41.00%	43.27%	44.36%	48.31%
	GUAP	54.97%	56.13%	59.88%	63.14%	49.98%	56.97%	60.08%	65.30%	45.19%	47.33%	50.87%	**54.33%**
	PGD	58.02%	59.47%	0.20%	0.33%	58.06%	**63.44%**	1.02%	1.99%	47.59%	**52.98%**	20.14%	24.77%
	DICE	30.46%	32.55%	0.03%	0.05%	36.68%	38.40%	0.73%	0.92%	32.32%	37.65%	18.00%	18.78%
	Meta-Self	-	-	0.26%	0.38%	-	-	0.80%	1.14%	-	-	19.56%	22.17%
	FGA	-	-	-	-	-	-	-	-	-	-	-	-
	MFAN	**59.48%**	**64.04%**	**64.04%**	**66.88%**	**60.09%**	62.95%	**62.95%**	**65.63%**	**49.64%**	51.34%	**51.34%**	53.34%

chosen to attack the target node during transfer attacks. If the new GNN model learns the same defect pattern of the target node, the selected set of anchor nodes can then effectively execute the transfer attack on the target node.

6.4 The Effects of K, λ, the Assignment Network, and Simulated Annealing

In this section, we discuss the effects of K, λ, the assignment network g_θ, and the simulated annealing technique.

6.4.1 The Effect of K

We investigate the effect of K by comparing the FR of MFAN with different K values when $\xi = 5$.

Table 6.4: FR of MFAN with different K values.

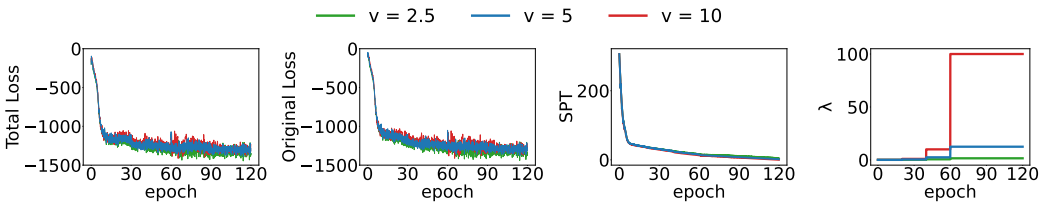
K	Cora	Citeseer	Facebook	Wiki
1	81.36%	79.14%	8.33%	83.43%
2	94.68%	92.77%	16.38%	93.56%
3	95.23%	92.77%	16.38%	93.56%
4	95.23%	92.77%	16.38%	93.87%

As displayed in Table 6.4, FR increases significantly when K rises from 1 to 2 on the four datasets. This is because, with $K = 1$, MFAN relies on a single set of anchor nodes, which is insufficient to successfully attack all target nodes. While $K = 2$, MFAN employs two sets of anchor nodes, each tailored to effectively attack different subsets of target nodes. Furthermore, with the help of the assignment network that selects the optimal set of anchor nodes for each attack, the success rate of each attack is improved. In this way, MFAN follows a "divide and conquer" approach, where the final set of successfully attacked target nodes is nearly the union of the target nodes attacked by each of the K sets of anchor nodes. When K exceeds 2, FR quickly converges due to the well-known "diminishing marginal effect": most target nodes attacked by the third set of anchor nodes have already been attacked by the first two, so adding a third set yields little improvement in FR. Thus, using two sets of anchor nodes is sufficient for MFAN to achieve excellent FR.

Since the BFA of MFAN is $\delta = K \times \xi$, we set $K = 2$ by default in the other experiments to save our BFA while achieving outstanding FR.

6.4.2 The Effect of λ

We examine the effect of hyperparameter λ by comparing the performance of MFAN with different *growth rates*, denoted by v, which serves as the multiplier for λ when increasing its value in line 10 of Algorithm 1.



(a) $\mathcal{L}^*(\mathcal{P}, \theta)$ (b) $\mathcal{L}(\mathcal{P}, \theta)$ (c) SPT (d) λ

Figure 6.3: The curves of the total loss $\mathcal{L}^*(\mathcal{P}, \theta)$, the original loss $\mathcal{L}(\mathcal{P}, \theta)$, the sum of penalty terms (SPT) $\sum_{\mathbf{p}_k \in \mathcal{P}} \max\big(||\mathbf{w}_k \circ \mathbf{p}_k||_1 - \mathbf{w}_k^{min}\xi, 0\big)$ and λ on the dataset Cora. Each subfigure shows three curves using different λ growth rates.

Figure 6.3 shows that the loss curves, using different values of v while training, stay comparable. This indicates that the training is resilient to variations in the λ growth rate. When zooming in on Figure 6.3(c), we observe that the SPT with a larger v decreases slightly faster than with a smaller v, as the faster increase in λ accelerates the push of the SPT toward zero. However, since the influence of v on MFAN's training is minimal, it has little impact on the FR.

Table 6.5: FR of MFAN using λ growth rates $v \in \{2.5, 5.0, 10.0\}$.

Dataset	$\xi = 5$			$\xi = 10$			$\xi = 20$		
	$v = 2.5$	$v = 5.0$	$v = 10.0$	$v = 2.5$	$v = 5.0$	$v = 10.0$	$v = 2.5$	$v = 5.0$	$v = 10.0$
Cora	93.69%	93.79%	93.97%	94.27%	94.30%	94.19%	95.85%	96.11%	95.85%
Citeseer	91.79%	92.37%	92.34%	97.24%	97.31%	97.33%	98.45%	98.45%	98.38%
Facebook	15.01%	15.55%	15.14%	25.85%	25.97%	25.66%	38.29%	38.84%	38.84%
Wiki	91.26%	92.55%	92.62%	94.61%	95.52%	95.33%	97.79%	97.82%	97.62%
Pubmed	59.48%	59.48%	58.98%	63.97%	64.04%	63.77%	66.79%	66.88%	66.31%

Consequently, as shown in Table 6.5, the FR of MFAN remains comparable across different λ growth rates.

6.4.3 The Effect of the Assignment Network

To assess the effect of the assignment network g_θ, we compare the FR of the standard MFAN (ST), in which the assignment network is used, with an ablated MFAN (AB) that selects anchor nodes uniformly at random. Both methods utilize the same sets of anchor nodes trained by the standard MFAN.

Table 6.6: The effect of the assignment network on FR.

Dataset	Model	GCN (white-box, not transfer)			GAT (black-box, transfer)			Node2Vec (black-box, transfer)		
		$\xi = 5$	$\xi = 10$	$\xi = 20$	$\xi = 5$	$\xi = 10$	$\xi = 20$	$\xi = 5$	$\xi = 10$	$\xi = 20$
Cora	MFAN (AB)	78.32%	83.43%	83.46%	75.15%	82.41%	83.60%	50.96%	69.34%	77.98%
	MFAN (ST)	**93.79%**	**94.30%**	**96.11%**	**84.67%**	**93.04%**	**94.59%**	**85.14%**	**92.17%**	**96.01%**
Citeseer	MFAN (AB)	77.41%	80.92%	81.70%	75.86%	80.40%	80.61%	80.16%	81.61%	82.81%
	MFAN (ST)	**92.37%**	**97.31%**	**98.45%**	**90.21%**	**95.78%**	**97.86%**	**91.79%**	**93.09%**	**94.65%**
Facebook	MFAN (AB)	13.33%	21.51%	29.46%	15.05%	27.81%	32.21%	12.98%	23.64%	29.76%
	MFAN (ST)	**16.25%**	**27.36%**	**39.53%**	**18.22%**	**33.49%**	**38.50%**	**20.45%**	**38.50%**	**58.53%**
Wiki	MFAN (AB)	75.77%	85.55%	90.48%	57.99%	63.72%	70.95%	44.49%	50.28%	78.05%
	MFAN (ST)	**92.55%**	**95.52%**	**97.82%**	**62.93%**	**67.07%**	**73.31%**	**47.78%**	**54.97%**	**82.70%**
Pubmed	MFAN (AB)	46.67%	48.26%	52.99%	40.26%	42.97%	45.94%	40.47%	44.70%	47.66%
	MFAN (ST)	**59.48%**	**64.04%**	**66.88%**	**60.09%**	**62.95%**	**65.63%**	**49.64%**	**51.34%**	**53.34%**

As shown in Table 6.6, the FR of the standard MFAN is significantly better than that of the ablated MFAN, demonstrating the effectiveness of the assignment network.

6.4.4 The Effect of Simulated Annealing

To investigate the effect of the simulated annealing trick, we compare the FR and quantization error of the perturbation vectors generated by two versions of MFAN. The first version is the standard MFAN (ST), which solves Equation (5.1.3), where the simulated annealing technique is incorporated. The second version is an ablated MFAN (AB), which solves Equation (5.1.2) without applying the simulated annealing technique. The *quantization error* (QE) is calculated using

$$\text{QE} = \sum_{\mathbf{p}_k \in \mathcal{P}} ||\mathbf{p}_k - \phi(\mathbf{p}_k, \xi)||_1, \tag{6.4.1}$$

where $\phi(\mathbf{p}_k, \xi)$ outputs the quantized $\mathbf{p}_k$, setting the top-ξ largest entries to 1 and the remaining entries to 0.

Table 6.7: The effect of simulated annealing on FR and QE.

Dataset	Model	FR ↑			QE ↓		
		$\xi = 5$	$\xi = 10$	$\xi = 20$	$\xi = 5$	$\xi = 10$	$\xi = 20$
Cora	MFAN (AB)	91.44%	92.53%	94.99%	8.68	16.01	28.42
	MFAN (ST)	**93.79%**	**94.30%**	**96.11%**	**1.37**	**2.98**	**4.07**
Citeseer	MFAN (AB)	90.76%	96.02%	96.88%	16.77	29.10	47.77
	MFAN (ST)	**92.37%**	**97.31%**	**98.45%**	**1.57**	**3.12**	**7.72**
Facebook	MFAN (AB)	14.88%	25.47%	38.12%	51.21	104.45	172.56
	MFAN (ST)	**16.25%**	**27.36%**	**39.53%**	**15.30**	**27.29**	**34.84**
Wiki	MFAN (AB)	90.69%	94.94%	96.75%	8.24	12.51	22.00
	MFAN (ST)	**92.55%**	**95.52%**	**97.82%**	**2.20**	**2.61**	**4.61**
Pubmed	MFAN (AB)	57.66%	60.98%	63.55%	9.84	20.14	27.10
	MFAN (ST)	**59.48%**	**64.04%**	**66.88%**	**2.85**	**3.43**	**5.01**

As shown in Table 6.7, the standard MFAN achieves a lower QE and a higher FR than the ablated MFAN. This highlights the effectiveness of the simulated annealing trick in reducing quantization error and enhancing attack performance.

6.5 Time Analysis

6.5.1 Attacking Time

We analyze the average attacking time for each method, representing the average time required to attack each target node in the test dataset.

We measured the actual average attacking time for the methods, as shown in Table 6.8, with $\xi = 10$ for FGA and $\delta = 10$ for the other approaches.

Table 6.8: Average attacking time in milliseconds.

Methods	Cora	Citeseer	Facebook	Wiki
GUA	146	233	-	112
GUAP	176	259	-	139
PGD	≈ 0	≈ 0	≈ 0	≈ 0
DICE	≈ 0	≈ 0	≈ 0	≈ 0
Meta-Self	≈ 0	≈ 0	≈ 0	≈ 0
FGA	468,370	1,008,467	1,539,644	357,708
MFAN	154	240	375	131

FGA incurs the longest attacking time because it requires training a new set of controlled nodes from scratch for each attack. This training cannot be done offline and only begins once FGA starts attacking a new target node, meaning its attack time is equivalent to its training time. In contrast, the anchor node methods achieve much faster attacking time than FGA, as they only need to flip a small number of edges when attacking a new target node, with the anchor nodes being trained offline prior to launching any attack. Our method has a slightly longer attack time than GUA, as it requires a forward pass of the assignment network to select the optimal set of anchor nodes. GUAP also takes a bit more time, as it involves injecting new anchor nodes into the graph. Global structural attacks, like PGD, DICE, and Meta-Self, perform offline training to perturb a large number of edges in a single step. This approach requires no further action when attacking new target nodes, resulting in nearly zero attacking time. However, the effectiveness of the global structural attacks is significantly limited by the constrained number of controlled nodes.

6.5.2 Training Time

We begin by analyzing the time complexity of the training process for all the methods. Next, we assess and compare the actual average training time of our approach with the baselines.

Table 6.9: Training time complexity.

Methods	Time Complexity
GUA	$O(max\_epoch \cdot N_T \cdot max\_iter)$
GUAP	$O(max\_epoch \cdot N_T \cdot max\_iter)$
PGD	$O(T)$
DICE	$O(1)$
Meta-Self	$O(\delta)$
FGA	$O(N\xi)$
MFAN	$O(max\_epoch \cdot N_T K)$

The primary factor influencing training time is the execution of forward and backward passes through GCN, the victim model. With N_T representing the number of nodes in the training set, Table 6.9 provides a summary of the training time complexity for each method, based on the number of GCN passes. Both GUA and GUAP, as anchor node attacks, make GCN calls for each target node during every epoch. However, they continuously modify perturbation vectors to check for misclassification, with the number of checks capped by max_iter, resulting in a time complexity of $O(max\_epoch \cdot N_T \cdot max\_iter)$. For global methods like PGD and Meta-Self, the entire graph structure is updated, generating a single perturbed adjacency matrix per epoch. The time complexity of PGD is bounded solely by the number of iterations, $O(T)$, while Meta-Self performs perturbations δ times, resulting in a time complexity of $O(\delta)$. DICE, which modifies edges purely based on the topological structure, is not a gradient-based method, thus involving no GCN operation, giving it a constant time complexity of $O(1)$ regarding GCN passes. FGA, a targeted attack method, modifies one edge per target node based on the largest gradient over ξ iterations, making its time complexity $O(N\xi)$ for attacking all target nodes. Our method, which follows the optimization objective outlined in Equation (5.1.3), requires $N_T K$ GCN calls per epoch, thus leading to a time complexity of $O(max\_epoch \cdot N_T K)$.

Table 6.10: Average training time in milliseconds.

Methods	Cora	Citeseer	Facebook	Wiki
GUA	13,084,775	24,971,433	-	9,508,160
GUAP	9,422,330	18,956,390	-	7,440,657
PGD	14,257	19,360	30,404	9,617
DICE	165	193	412	140
Meta-Self	6,798	7,217	8,145	6,211
FGA	468,370	1,008,467	1,539,644	357,708
MFAN	588,663	1,021,496	1,845,367	450,552

We recorded the actual average training time for attacking all target nodes, as shown in Table 6.10, with $\xi = 10$ for FGA and $\delta = 10$ for the other approaches. Since max_iter is substantially larger than our K, our method proves to be more efficient, requiring only minutes of training time, compared to the hours needed by GUA and GUAP. The global methods — PGD, Meta-Self, and DICE — do not rely on per-node GCN passes, allowing them to run faster than our method by a factor of N_T, completing in seconds or milliseconds. FGA, which requires a total of $O(N\xi)$ GCN calls to generate attacks for all target nodes, has a training time comparable to ours.

Chapter 7

Future Work

Although our method achieves promising performance with the results discussed above, we admit there is still room for improvement. The challenge on the effect of the hyperparameter K remains to be further investigated.

As analyzed in Section 6.4.1, we are facing the issue of performance convergence in terms of FR due to the "diminishing marginal effect". The improvement in FR from $K = 2$ to $K = 3$ drops to less than 1% or even 0%, comparing to the rise of over 10% from $K = 1$ to $K = 2$ across the datasets. This could be the reason that the our graph datasets is limited in size, as there are potential uses of our method on larger datasets, such as graphs of millions or more nodes and edges, and in such senarios, investigation and experiments are to be conducted. We also assume that in graphs where nodes are more centralized within three or more components — characterized by strong connections within components and weak connections between them — a further increase in K may result in greater improvement in FR, making it worth studying. Thus, when it comes to larger graphs, selection of a larger K is probably a better choice. Natually, how to dynamically select K is another topic worth discussing, and the idea of incremental learning that gradually increases the K value while training until convergence may give us a good hint.

The expressiveness of our assignment network is another direction for future improvement. With a finer choice of the assignment network, it can potentially not only reduce the diminishing marginal effect with a larger K, but also helps better identify the sets of anchor nodes. Increasing the depth of convolutional layers, change of aggregation functions, or attaching classification heads could be means of enhancing expressiveness; utilizing other node classification models, such as GIN [37] and GAT [32], may also bring improvement similarly.

In summary, we hope the initial success of MFAN can draw some attention to the community, and the discussion and hypothesis above may give some directions to the future work.

Chapter 8

Conclusion

In this paper, we proposed and addressed a novel problem, termed multifaceted anchor nodes attack. The key idea involves simultaneously optimization of multiple sets of anchor nodes in conjunction with an assignment network. Each set of anchor nodes is specially tailored to effectively attack a different set of target nodes, while the assignment network proficiently allocates the best suitable set of anchor nodes to attack a new target node. Through this approach, we operationalize "divide and conquer" mechanism, enabling successful attacks on the collective union of the nodes attacked by each set of anchor nodes. Notably, by employing the same sets of anchor nodes across all target nodes, our method demonstrates significant budget efficiency, requiring the control of only a minimal number of nodes to achieve superior attack performance.

Bibliography

[1] Aleksandar Bojchevski and Stephan Günnemann. 2019. Adversarial attacks on node embeddings via graph poisoning. In *ICML*. 695–704.

[2] Avishek Joey Bose, Andre Cianflone, and William L Hamilton. 2019. Generalizable adversarial attacks with latent variable perturbation modelling. *arXiv:1905.10864* (2019).

[3] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[4] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. 2020. Popularity prediction on social platforms with coupled graph neural networks. In *WSDM*. 70–78.

[5] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. 2020. A restricted black-box adversarial framework towards attacking graph embedding models. In *AAAI*. 3389–3396.

[6] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential recommendation with graph neural networks. In *ACM SIGIR*. 378–387.

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

[8] Yang Chen, Zhonglin Ye, Haixing Zhao, Ying Wang, et al. 2023. Feature-based graph backdoor attack in the node classification task. *International Journal of Intelligent Systems* (2023).

[9] Mark Cheung and José MF Moura. 2020. Graph neural networks for covid-19 drug discovery. In *IEEE International Conference on Big Data*. 5646–5648.

[10] Silviu Cucerzan. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP-CoNLL*. 708–716.

[11] Enyan Dai, Minhua Lin, Xiang Zhang, and Suhang Wang. 2023. Unnoticeable backdoor attacks on graph neural networks. In *WWW*. 2263–2273.

[12] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *ICML*. 1115–1124.

[13] Jiazhu Dai, Weifeng Zhu, and Xiangfeng Luo. 2020. A targeted universal attack on graph convolutional network. *arXiv:2011.14365* (2020).

[14] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.

[15] Anthony V Fiacco and Garth P McCormick. 1990. *Nonlinear programming: sequential unconstrained minimization techniques*. SIAM.

[16] Kunihiko Fukushima. 1969. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics* 5, 4 (1969), 322–333.

[17] Simon Geisler, Tobias Schmidt, Hakan Şirin, Daniel Zügner, Aleksandar Bojchevski, and Stephan Günnemann. 2021. Robustness of graph neural networks at scale. In *NeurIPS*. 7637–7649.

[18] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*. 855–864.

[19] Mingxuan Ju, Yujie Fan, Chuxu Zhang, and Yanfang Ye. 2023. Let graph be the go board: Gradient-free node injection attack for graph neural networks via reinforcement learning. In *AAAI*. 4383–4390.

[20] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[21] Srijan Kumar, Robert West, and Jure Leskovec. 2016. Disinformation on the web: Impact, characteristics, and detection of wikipedia hoaxes. In *WWW*. 591–602.

[22] Jure Leskovec and Julian Mcauley. 2012. Learning to discover social circles in ego networks. In *NeurIPS*. 539–547.

[23] Kuan Li, Yang Liu, Xiang Ao, and Qing He. 2023. Revisiting graph adversarial attack and defense from a data distribution perspective. In *ICLR*.

[24] Zihan Liu, Yun Luo, Lirong Wu, Zicheng Liu, and Stan Z Li. 2022. Towards reasonable budget allocation in untargeted graph structure attacks via gradient debias. *NeurIPS* (2022), 27966–27977.

[25] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. 2021. STGSN—a spatial–temporal graph neural network framework for time-evolving social networks. *Knowledge-Based Systems* 214 (2021), 106746.

[26] R. Tyrrell Rockafellar. 1970. *Convex analysis*. Princeton University Press, Princeton, N. J.

[27] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[28] Alice E Smith, David W Coit, Thomas Baeck, David Fogel, and Zbigniew Michalewicz. 1997. Penalty functions. *Handbook of evolutionary computation* (1997).

[29] Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. 2020. Non-target-specific node injection attacks on graph neural networks: A hierarchical reinforcement learning approach. In *WWW*. 673–683.

[30] Tsubasa Takahashi. 2019. Indirect adversarial attacks via poisoning neighbors for graph convolutional networks. In *IEEE International Conference on Big Data*. 1395–1400.

[31] Shuchang Tao, Qi Cao, Huawei Shen, Junjie Huang, Yunfan Wu, and Xueqi Cheng. 2021. Single node injection attack against graph neural networks. In *CIKM*. 1794–1803.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[33] Binghui Wang, Meng Pang, and Yun Dong. 2023. Turning strengths into weaknesses: A certified robustness inspired attack framework against graph neural networks. In *CVPR*. 16394–16403.

[34] Xiaoyun Wang, Minhao Cheng, Joe Eaton, Cho-Jui Hsieh, and S Felix Wu. 2022. Fake node attacks on graph convolutional networks. *Journal of Computational and Cognitive Engineering* 1, 4 (2022), 165–173.

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

[36] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology attack and defense for graph neural networks: An optimization perspective. In *IJCAI*. 3961–3967.

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

[38] Xiao Zang, Jie Chen, and Bo Yuan. 2023. GUAP: Graph universal attack through adversarial patching. *arXiv:2301.01731* (2023).

[39] Xiao Zang, Yi Xie, Jie Chen, and Bo Yuan. 2020. Graph universal adversarial attacks: A few bad actors ruin graph learning models. In *IJCAI*. 3328–3334.

[40] Binchi Zhang, Yushun Dong, Chen Chen, Yada Zhu, Minnan Luo, and Jundong Li. 2024. Adversarial attacks on fairness of graph neural networks. In *ICLR*.

[41] Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. 2021. Tdgia: Effective injection attacks on graph neural networks. In *ACM SIGKDD*. 2461–2471.

[42] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *ACM SIGKDD*. 2847–2856.

[43] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. In *ICLR*.