# Derivative-free optimization adversarial attacks for graph convolutional networks

**Runze Yang[1] and Teng Long[1]**

[1]**School of Information Engineering, China University of Geosciences (Beijing), Beijing, China**

Corresponding author:
Teng Long[1]

Email address: longteng@cugb.edu.cn

## ABSTRACT

In recent years, Graph Convolutional Networks (GCNs) have emerged rapidly due to their excellent performance in graph data processing tasks. However, recent researches show that GCNs are vulnerable to adversarial attacks. An attacker can maliciously modify edges or nodes of the graph to mislead the model's classification of the target nodes, or even cause a degradation of the model's overall classification performance. In this paper, we first propose a black-box adversarial attack framework based on derivative-free optimization (DFO) to generate graph adversarial samples without using gradient and apply advanced DFO algorithms conveniently. Second, we implement a direct attack algorithm (DFDA) using *Nevergrad* library based on the framework. Additionally, we overcome the problem of large search space by redesigning the perturbation vector using constraint size. Finally, we conducted a series of experiments on different datasets and parameters. The results show that DFDA outperforms *Nettack* in most cases, and it can achieve an average attack success rate of more than 95% on the Cora dataset when perturbing at most 8 edges. This demonstrates that our framework can fully exploit the potential of DFO methods in node classification adversarial attacks.

## INTRODUCTION

Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) are one of the most popular Graph Neural Networks (GNNs) (Scarselli et al., 2009). They are widely used in recommendation systems (He et al., 2020), molecular chemistry (Ryu et al., 2019) and knowledge graphs (Wang et al., 2019). Like traditional neural networks, GCNs are also vulnerable to adversarial attacks. The adversary can modify structure and node features to make the model misclassify the target nodes (Zügner et al., 2018; Dai et al., 2018; Ma et al., 2019), or even cause the overall classification performance to degrade (Zügner and Günnemann, 2019). Therefore, the study of GCNs' robustness has received increasingly widespread attention (Sun et al., 2018).

Gradient-based adversarial attacks in the continuous domain have had a series of related works on both images (Goodfellow et al., 2015; Papernot et al., 2016) and audio (Carlini and Wagner, 2018). Unlike adversarial attacks in the continuous domain, graph data are often discrete: the structural information (adjacency matrix) of the graph is

40 discontinuous and the features may also have discrete values. This makes it difficult to
41 use gradient information to attack (Zügner et al., 2018), especially under a black-box
42 condition where only classification output vector can be obtained.

43 Derivative-free optimization (DFO) algorithms (Conn et al., 2009) are a class of
44 algorithms that do not compute the gradient but only use the value of the objective
45 function to optimize. These algorithms are often used in cases where the derivative of
46 the objective function is undefined, or where it is difficult to obtain a reliable value of
47 the derivative. It has been successfully applied to attack traditional deep neural networks
48 (Ughi et al., 2020). There is also some DFO attack work on GNNs. For example, Dai
49 et al. (2018) implemented a black-box GCN adversarial attack algorithm based on a
50 genetic algorithm by setting the population, fitness, selection, crossover and mutation
51 in detail. Chen et al. (2019) proposed a community detection attack algorithm with
52 a genetic algorithm and verified their algorithm has good transferability. However,
53 without a uniform framework, these works usually have to implement custom versions
54 of the algorithms for a certain problem. To the author's best knowledge, there is no
55 general framework that can quickly apply various DFO algorithms in the field of graph
56 adversarial attacks.

57 In this paper, we propose a black-box adversarial attack framework based on the idea
58 of DFO. It consists of three steps: *Input Setting* (design the loss function, perturbation
59 vector, constraints and so on), *Iterative Query* (generate perturbation vectors and query
60 the black-box GCN model iteratively) and *Final Perturbation* (modify graph data with
61 perturbation that minimize the loss function). As facing the difficulty of using gradient
62 and the inconvenience of applying and comparing DFO algorithms, the key idea insight
63 of our approach is (1) regarding graph adversarial attacks as a search problem in a
64 discrete solution space and using derivative-free optimizers (DFOers) to solve it; (2)
65 abstracting the specific task of graph adversarial attacks as an optimization problem
66 about a certain form perturbation vector in order to switch and compare various DFOers
67 conveniently.

68 Moreover, we use the *Nevergrad* (Rapin and Teytaud, 2018) library to implement
69 a black-box direct adversarial attack algorithm (called DFDA) on GCN-based node
70 classification tasks. Following the framework above, we set Attack Loss Function,
71 Perturbation Vector, Perturbation Constraint, Mapping Function and Derivative-Free
72 Optimizer separately.

73 We conducted a series of experiments on Cora, Citeseer and Polblogs. Without loss
74 of generality, we attack node 0 of the Cora dataset with five different DFOers to compare
75 the classification margin and comprehensive performance. Then, we randomly select 50
76 nodes to attack separately to study the average attack success rate of DFDA with different
77 iteration numbers, perturbation constraints and perturbation types. Finally, we compare
78 DFDA with a classical algorithm *Nettack* (Zügner et al., 2018) —a well-performing
79 greedy algorithm— under different defense models. *Nettack* is a well-performing
80 adversarial attack algorithm based on greedy approach. The results show that all the
81 selected DFOers can search for effective perturbations, and DFDA is superior to *Nettack*
82 in most cases.

83 The contributions of this paper are listed below:

84 • We have proposed a black-box adversarial attack framework in order to generate

85  graph adversarial samples without using the gradient. Specifically, we use DFO
86  methods to perform attacks effectively. Additionally, the uniform outputs of
87  DFOers (perturbation vectors) make it convenient to use and compare different
88  DFO algorithms.

89  • We have implemented a direct adversarial attack algorithm on node classification
90  tasks for GCNs based on the framework above. Facing the potential problem of
91  too large search space, we set the perturbation vector dimension to the constraint
92  size and set the elements of the perturbation vector as the pointers indicating the
93  perturbed position of the original matrix. This approach reduces the search space
94  from exponential level to power level and enables the perturbation vector to pass
95  the constraint check more easily.

96  • We have conducted a series of experiments under various conditions. The results
97  show that we can achieve an average attack success rate of more than 95% on the
98  Cora dataset when perturbing at most 8 edges. We compare our algorithm with the
99  classical algorithm *Nettack* under different defense models and find that DFDA
100 outperforms *Nettack* in most cases. During the experiments, instead of copying
101 an original graph at each perturbation iteration, we use inverse perturbation to
102 restore the perturbed graph to its original state. This can effectively reduce the
103 computation cost under a large number of iterations.

104 This paper is organized as follows. "Preliminaries" gives the basic concepts of
105 GCN-based node classification and adversarial attacks. "Derivative-Free Adversarial
106 Attack on GCNs" describes our framework and algorithms, followed by the experimental
107 results in "Evaluation". "Related Work" introduced the related work, followed by some
108 concluding remarks in "Conclusion".

## RELATED WORK

110 There has been some classical work on node classification adversarial attacks. The
111 concept of graph adversarial attacks was first introduced by Zügner et al. (2018). They
112 proposed a gray-box attack algorithm *Nettack* based on the greedy approach. In this
113 algorithm, the attacker can obtain training labels to train the surrogate model. The
114 adversary can generate an adversarial sample by attacking the surrogate model and
115 migrate it to the target model for attacks. Subsequently, they proposed *Mettack* (Zügner
116 and Günnemann, 2019), an attack algorithm that can reduce the global classification
117 accuracy. This algorithm attacks based on the gradient information of the adjacency
118 matrix. Dai et al. (2018) proposed a black-box attack algorithm named *RL-S2V* based on
119 reinforcement learning. This method performs attacks by injecting nodes into the graph.
120 *Mettack* is a global attack algorithm that attacks to decline the classification accuracy
121 of a whole graph instead of a target node. Black-box queries in *RL-S2V* setting only
122 return the prediction classes rather than class probability vectors. Therefore we do not
123 compare with the above algorithms. We compare DFDA with *Nettack* because the attack
124 settings are more similar: both of them can perform direct perturbations of edges and
125 features of a target node. For the sake of reasonableness, we unify *Nettack*'s settings in
126 terms of node selection and constraints with DFDA.

DFO methods like genetic algorithms are used in some graph adversarial attack tasks. For example, Dai et al. (2018) applied genetic algorithms to node classification adversarial attacks and adapted it as a baseline algorithm for their main algorithm RL-S2V. Experiments in their paper demonstrate that genetic algorithms are effective for combinatorial optimization problems like graph adversarial attacks. Chen et al. (2019) proposed a community detection attack algorithm based on genetic algorithms and verified that the algorithm has good transferability. In this paper, we propose an adversarial attack framework to make it convenient to apply and compare different DFO algorithms.

*Nevergrad* (Rapin and Teytaud, 2018) is a Python3 based open-source framework developed by Facebook, which provides a large number of implementations of DFO algorithm optimizers, such as, differential evolution algorithms, fast genetic algorithms, covariance matrix adaptive algorithms, particle swarm optimization algorithms, etc. When DFO algorithms need to be applied, researchers usually need to implement custom versions of algorithms for real problems, which consume a lot of time and do not facilitate the comparisons among algorithms. The algorithm DFDA we proposed is based on *Nevergrad*, using which the strengths and weaknesses of various DFO algorithms can be easily compared. It can help researchers find suitable algorithms that deserve to be further customized.

## PRELIMINARIES

In this section, we introduce the notions about GCN-based node classification and the general form of adversarial attack on graph.

### Node Classification with GCN

The classification task on graph data mainly consists of two cases, one is graph-level classification, where the graph $G$ is a whole with a label, and the other is node-level classification, where each node in the graph $G$ belongs to a class in the label set $\boldsymbol{Y}$. In this paper, we focus on node-level semi-supervised classification. Semi-supervised node classification is a task in which the labels of unknown nodes are derived by training under the condition that the training set labels are known.

In the following we give the mathematical definition of semi-supervised node classification. For an undirected graph $G$, given the adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N \times N}$, the node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ (the feature of the node $i$ is $\boldsymbol{x}_i \in \mathbb{R}^d$), the label of the $i$-th node $y_i \in \{1,2,...,k\}$, a set of labeled nodes $V_L$ (training set, where $|V_L| = N_L$), and a set of unlabeled nodes $V_U$ (test set, where $|V_U| = N_U$). The objective is to train a model $f_\theta(G)$ on the graph $G$ with training parameter $\theta$ that predicts the label of each node in $V_U$.

The general idea of semi-supervised learning model training is to minimize the loss function of the model on the training set as much as possible, i.e.,

$$\min \mathscr{L}_{train}(f_\theta(G)) = \sum_{v_i \in V_L} \ell(f_\theta(\boldsymbol{A},\boldsymbol{X})_i, y_i) \tag{1}$$

where $f_\theta(\boldsymbol{A},\boldsymbol{X})_i$ denotes the class probilities of node $v_i$ and $y_i$ denotes the true label of node $v_i$. The loss function $\ell(\cdot,\cdot)$ denotes the cross-entropy error.

We use the 2-layer GCN proposed by (Kipf and Welling, 2017) as the target model. The 2-layer GCN is one of the victim models commonly used in adversarial attack experiments. It is defined as follows:

$$f(\boldsymbol{A}, \boldsymbol{X}) = \text{Softmax}\left(\hat{\boldsymbol{A}} \, \text{ReLU}\left(\hat{\boldsymbol{A}} \boldsymbol{X} \boldsymbol{W}^{(1)}\right) \boldsymbol{W}^{(2)}\right) \tag{2}$$

where $\hat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and $\widetilde{\boldsymbol{D}}_{ii} = \sum_j \widetilde{\boldsymbol{A}}_{ij}$. Each row vector $f(\boldsymbol{A}, \boldsymbol{X})_i \in [0,1]^k$ of the output matrix represents the class probability vector of node $v_i$.

### General Forms of Adversarial Attack

Given a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ and a set of attack target nodes $V_t \subseteq V$. Let $y_i$ denote the true class of node $v_i$. For node $v_i$ in the test set, since we do not know its true label, we can adopt the method of self-learning (Zügner and Günnemann, 2019), that is, regarding the model output on the clean graph as the true labels of unknown nodes. The attack objective is to find a perturbed graph $G' = (\boldsymbol{A}', \boldsymbol{X}')$, which makes the attack target function $\mathscr{L}_{atk}$ minimum, i.e.,

$$\begin{aligned}
\min \mathscr{L}_{atk}\left(f_\theta\left(G'\right)\right) &= \sum_{v_i \in V_t} \ell_{atk}\left(f_{\theta^*}\left(G'\right)_i, y_i\right) \\
\text{s.t.,} \quad \theta^* &= \arg\min_\theta \mathscr{L}_{\text{train}}\left(f_\theta\left(\hat{G}\right)\right)
\end{aligned} \tag{3}$$

where $\ell_{atk}$ is the loss function of the attack. $\hat{G}$ can be chosen as either the original graph $G$ or the perturbed graph $G'$, corresponding to the poisoning attack scenario and the escape attack scenario, respectively. Poisoning attack means that the GCN will be retrained with the perturbed graph while evasion attack represents the cases that the GCN will not be retrained (Sun et al., 2018).

In particular, for an adversarial attack with a certain single node $v_i$, the objective function can be formulated as:

$$\min \mathscr{L}_{atk}\left(f_\theta\left(G'\right)\right) = \ell_{atk}\left(f_{\theta^*}\left(G'\right)_i, y_i\right). \tag{4}$$

However, the perturbation of $G'$ is constrained and does not allow unrestricted modification on the graph. A realistic assumption is that the attack needs to generate as small and indistinguishable perturbations as possible, i.e., $G \in \Phi(G)$, where $\Phi(G)$ denotes the constraint domain. If a perturbation upper bound $\Delta$ is given, a typical perturbation constraint can be expressed as

$$\left\|\boldsymbol{A} - \boldsymbol{A}'\right\|_0 + \left\|\boldsymbol{X} - \boldsymbol{X}'\right\|_0 \leq \Delta. \tag{5}$$

## DERIVATIVE-FREE ADVERSARIAL ATTACK ON GCNS

In this section, we introduce a black-box adversarial attack framework for GCNs based on the idea of DFO. And then, we implement a direct attack algorithm on the GCN node classification task using the *Nevergrad* algorithm library. Finally, we optimize the attack algorithm to solve the problem of excessive dimensionality encountered during the algorithm implementation.
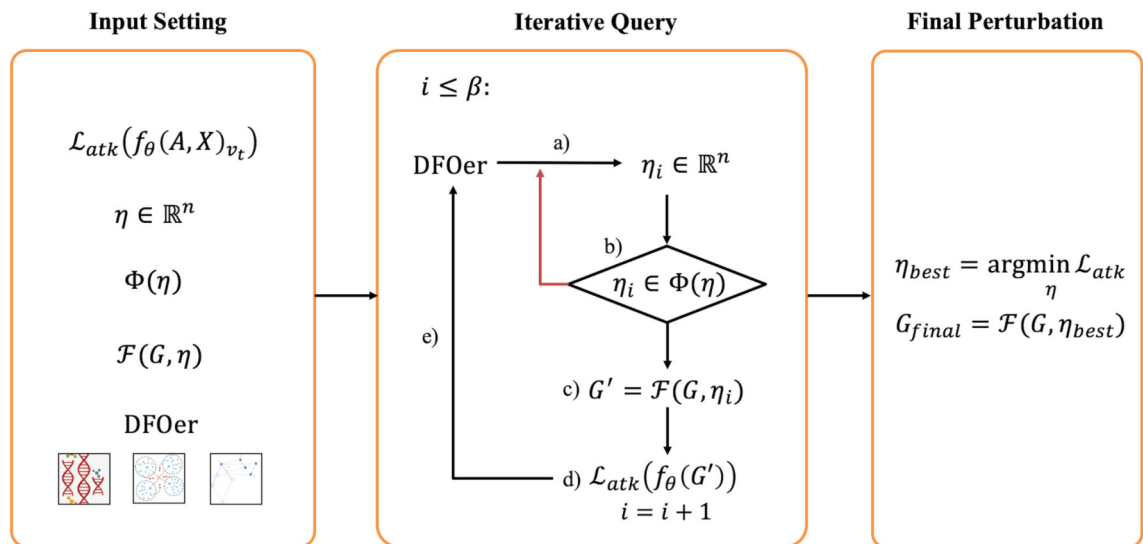
**Figure 1.** Derivative-Free Black-Box Attack Framework

### Derivative-Free Black-Box Attack Framework

The core idea of the framework is mainly twofold: (1) we treat graph adversarial attacks as a search problem in a discrete solution space and use derivative-free optimizers (DFOers) to solve it; (2) we abstract the specific task of graph adversarial attacks as an optimization model about perturbation vectors. The former mainly copes with the problem of difficult gradient utilization in black-box graph adversarial attacks. The DFOers can optimize without computing the gradient. The latter reflects the generality of the framework. The framework unifies the output of the DFOer into a certain form perturbation vector, which provides a basis for the rapid switching and comparison of various DFOers.

The main elements of the framework are described below. This framework contains three main steps: *Input Setting*, *Iterative Query* and *Final Perturbation*. The schematic diagram of the framework is shown in Figure 1.

#### *Input Setting*

In this step, we need to define the attack loss function $\mathcal{L}_{atk}$, perturbation vector $\eta$, perturbation constraint $\Phi(\eta)$, mapping function $\mathcal{F}$ and the DFOer $Opt$. In the black-box scenario, we only know the adjacency matrix, the feature matrix and the black-box query result in the form of class probability vectors. So the attack loss function is usually constructed from the black-box query results. It is the target of our optimization. In general, the smaller the function value, the better the attack effect. The perturbation vector is the interface between the DFOer and the loss function. The uniform perturbation vector ensures the feasibility of switching the DFOer for comparison experiments. The perturbation constraint defines the restrictions on the perturbation vector generated by the DFOer. The mapping function defines how the perturbation is imposed on the graph $G$. The DFOer can return a perturbation vector that tends to make the loss function as small as possible by learning the value of the loss function obtained from the query.

### Iterative Query

After defining the input, we will perform $\beta \in \mathbb{N}$ iterations to generate the candidate perturbation vectors. Within the number of iterations $\beta$ (also called resource budget), repeat: a) generate a perturbation vector with the DFOer; b) check whether the perturbation vector violates the constraints and regenerate the vector that violates them; c) impose the perturbation to the original graph $G$ to generate the perturbed graph $G'$; d) input the perturbation graph $G'$ into the GCN model and query the attack loss function value; e) return it to the optimizer. In this step, all perturbation vectors and the corresponding loss values are recorded.

### Final Perturbation

At the end of the iterative query, the perturbation vector that minimizes the loss function will be imposed to the graph $G$ to generate $G_{final}$. At this point, the whole adversarial sample generation process is finished. Subsequently, we put $G_{final}$ in (3) as $G'$ for poisoning attacks or evasion attacks.

## Derivative-Free Direct Attack (DFDA)

The Derivative-Free Direct Attack (DFDA) algorithm is a black-box adversarial attack method on GCN node classification tasks. In this algorithm, the adversary can directly modify the connections and features of the target node to mislead GCNs to misclassify the node as a chosen class (called target class). DFDA can perform structural perturbation (modifying the adjacency matrix) and feature perturbation (modifying the feature matrix).

First, we set up the five subsections of *Input Setting* of the framework.

**Attack Loss Function:** Considering the success rate of the attack, the second most probable class originally predicted by the clean graph is selected as the target class. The loss function is designed as,

$$\mathcal{L}_{atk} = f(\boldsymbol{A}, \boldsymbol{X})_{v_t, c_1} - f(\boldsymbol{A}, \boldsymbol{X})_{v_t, c_2} \tag{6}$$

$f(\boldsymbol{A}, \boldsymbol{X})_{v_t}$ is the output of the black-box query of the target node $v_t$ as a class probability vector. $c_1$ and $c_2$ denote the first and second largest probability classes of the black-box query before the attack. When the loss function decreases, the target class probability increases and the correct class probability decreases.

**Perturbation Vector:** For structural perturbations, the perturbation vector needs to be set to $\{0, 1\}^{N-1}$ to describe all possible perturbations that can be generated, since the target node may have edges with all other $N-1$ nodes. The size of the search space for this problem is $O(2^N)$, which is of exponential level. Since the number of edges that an attacker can modify (usually set to $\Delta$) is very small to ensure the invisibility of the perturbation in practical situations, the perturbation constraint is usually very tight. The vast majority of perturbations are not qualified. To reduce the search space and improve the probability of passing the constraint check, we define the structural perturbation vector as $\eta_A \in \{x \in \mathbb{N} \mid x < N\}^{\Delta}$. As is shown in Figure 2, each element $u \in \eta_A$ like a position pointer represents that the connection between the target node and the node $v_u$ is changed. This method reduces the search space significantly (down to $O(N^{\Delta})$). Similarly, we set the feature perturbation vector to $\eta_X \in \{x \in \mathbb{N} \mid x < d\}^{\Delta}$. Here, we only consider the case where the feature matrix is of the form $\boldsymbol{X} = \{0, 1\}^{N \times d}$.
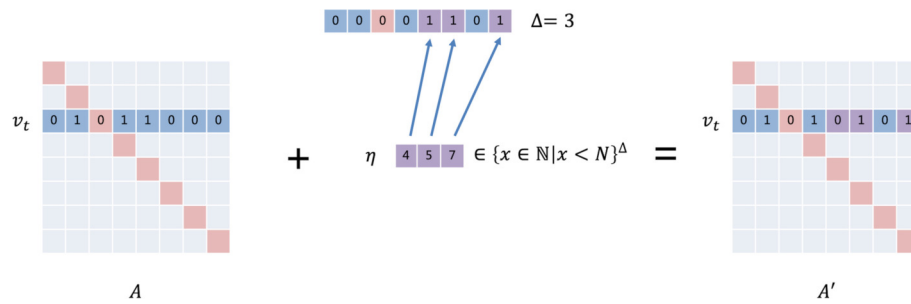
**Figure 2.** The setting of Perturbation Vector. The upper vector is in the form of $\{0,1\}^N$. 0 means that the connection status remains unchanged, and 1 means that the edge is added or removed. We design the lower vector which uses the position pointers as elements to decline the dimension.

**Perturbation Constraint:** Based on the design of the above perturbation vector, we define the constraints as: *a)* no duplicate elements in $\eta_A$ and $\eta_X$; and *b)* $v_t \notin \eta_A$. Constraint *a* guarantees that no duplicate modifications will be made to a particular edge while *b* guarantees that the graph structure will not generate self-loops.

**Mapping Function:** The perturbation mapping function is defined as follows. For each element $i$ in $\eta_A$,

$$A_{v_t i} := 1 - A_{v_t i}. \tag{7}$$

For each element $j$ in $\eta_F$,

$$X_{v_t j} := 1 - X_{v_t j}. \tag{8}$$

**Derivative-Free Optimizer:** We choose OnePlusOne, DiscreteOnePlusOne, DoubleFastGAOnePlusOne, DE and RandomSearch in *Nevergrad* as DFOers to generate the perturbation vectors.

After defining the output, we follow the framework for *Iterative Query* and *Final Perturbation*. In the *Iterative Query* step, our approach is slightly different from the framework. In each iteration, we let the DFOer generate three perturbation vectors simultaneously, perform constraint checking and loss query separately, and finally select the perturbation vector with the smallest attack loss as the output of this iteration. These can make the optimization process more stable.

## EVALUATION

We have implemented our algorithm on DeepRobust (Li et al., 2020), an adversarial attack algorithm library developed on PyTorch. As DeepRobust integrates classical attack and defense algorithms on the image and graph domains, it can support the comparison between our algorithm and other algorithms.

We consider the following three research questions:

- **RQ1:** Which DFOer is the most suitable in the setting of DFDA?

- **RQ2:** To what extent does parameters like constraint size affect the attack results?

---

**Algorithm 1** The Derivative-Free Direct Attack (DFDA) algorithm

---

**Input:** Original clean graph $G = (\boldsymbol{A}, \boldsymbol{X})$, target node $v_t \in V_t$, resource budget $\beta$, perturbation constraint $\Delta$, black-box target model $\mathscr{M}$

**Output:** Perturbed adversarial sample $G = (\boldsymbol{A'}, \boldsymbol{X'})$

1: $p_{v_t} \leftarrow \text{Query}(\mathscr{M}, v_t)$
2: Get the most probable class $c_1$ and the second most probable class $c_2$ in $p_{v_t}$
3: $t \leftarrow 0$
4: **while** $t < \beta$ **do**
5:     **repeat**
6:         Generate perturbation vector $\eta_A^{(t)} \in \{x \in \mathbb{N} \mid x < N\}^\Delta, \eta_X^{(t)} \in \{x \in \mathbb{N} \mid x < d\}^\Delta$
        with a chosen DFOer
7:     **until** $v_t \notin \eta_A^{(t)}$ **and** $\eta_A^{(t)}, \eta_X^{(t)}$ does not contain duplicate elements
8:     $\boldsymbol{A}^{(t)} \leftarrow \boldsymbol{A}, \boldsymbol{X}^{(t)} \leftarrow \boldsymbol{X}$
9:     **for** each element $i$ in vector $\eta_A^{(t)}$ **do**
10:         $\boldsymbol{A}^{(t)}[v_t][i] \leftarrow 1 - \boldsymbol{A}[v_t][i]$
11:     **end for**
12:     **for** each element $j$ in vector $\eta_X^{(t)}$ **do**
13:         $\boldsymbol{X}^{(t)}[v_t][i] \leftarrow 1 - \boldsymbol{X}[v_t][i]$
14:     **end for**
15:     $p_{v_t} \leftarrow \text{Query}(\mathscr{M}, v_t)$
16:     $loss^{(t)} \leftarrow p_{v_t}^{(t)}[c_1] - p_{v_t}^{(t)}[c_2]$
17:     Inform the optimizer of $loss$
18: **end while**
19: $\boldsymbol{A'} \leftarrow \boldsymbol{A}, \boldsymbol{X'} \leftarrow \boldsymbol{X}$
20: Select $\eta_A^{(m)}, \eta_X^{(m)}$ that minimizes the $loss$
21: **for** each element $i$ in vector $\eta_A^{(m)}$ **do**
22:     $\boldsymbol{A'}^{(t)}[v_t][i] \leftarrow 1 - \boldsymbol{A}[v_t][i]$
23: **end for**
24: **for** each element $j$ in vector $\eta_X^{(m)}$ **do**
25:     $\boldsymbol{X'}^{(t)}[v_t][i] \leftarrow 1 - \boldsymbol{X}[v_t][i]$
26: **end for**
27: **return** $G' = (\boldsymbol{A'}, \boldsymbol{X'})$

---

278     • **RQ3:** Compare with *Nettack*, how does our method perform under different
279       defense models and scenarios?

280     To answer these questions, we design three kinds of experiments in the next section.
281     To answer RQ1, we conducted a single-node experiment, attacking only one node, to
282 analyze the attack results. Without loss of generality, we compare the effect of different
283 DFOers on node 0 in Cora. In this experiment, we select the most appropriate optimizer
284 for subsequent experiments.
285     To answer RQ2, we conducted multi-node experiments in which all nodes in a target
286 node set were attacked separately. The percentage of "successful" and "misleading"
287 nodes were counted. In the multi-node experiments, we investigate the effects of resource
288 budget $\beta$, perturbation constraint $\Delta$, and perturbation type on the success rate of the
289 attack.
290     To answer RQ3, we conducted comprehensive attack experiments on different
291 datasets and different defense models. In these experiments, we investigate the attack
292 effect of DFDA under three defense models in evasion attack and poisoning attack
293 scenarios. We also compare the DFDA with the greedy algorithm *Nettack* in poisoning
294 attack cases.

295 **Dataset and Settings**
296 **Dataset** The commonly used datasets in the field of graph adversarial attacks are Cora,
297 Citeseer and Polblogs (Rossi and Ahmed, 2015). We present the statistics for each
298 dataset in the following Table 1. Among them, Cora and Citeseer are attribute graphs,
299 i.e., each node in the graph has a specific dimensional attribute; the Polblogs dataset is
300 a directed weighted graph with no node features. Cora and Citeseer are more sparse,
301 while Polblogs is denser and has only two categories. Due to their strong representation,
302 we perform attacks based on these three datasets.

**Table 1.** Dataset Statistics

| Dataset | Nodes | Edges | Features | Classes |
|---------|-------|-------|----------|---------|
| Cora | 2708 | 5429 | 1433 | 7 |
| Citeseer | 3327 | 4732 | 3703 | 6 |
| Polblogs | 1490 | 19025 | - | 2 |

303     **Setting** We describe some of the parameters and their range of values during the
304 experiment in Table 2. The poisoning attack considered in this paper refers to the case
305 where the model is not retrained at query time and is retrained at test time after the
306 attack. In this case, our approach is equivalent to generating an adversarial sample with
307 the training parameters of the original model and then transferring this sample to the
308 model with the new parameters after retraining.
309     We define "original class" as the maximum probability class of the target node's
310 black-box query before the attack. We define "successful attack" as the case that the
311 maximum probability class after the attack is different from the original class. We
312 define "misleading success" as the maximum probability class of the node after the
313 attack is the same as the selected target class. "Classification margins" here are defined

as $Z_{v,c} - \max_{c' \neq c} Z_{v,c'}$ where $c$ is the original class, $Z_{v,c}$ is the probability of the class $c$ given to the node $v$ by the attacked model. The lower the classification margins, the better the attack performance.

**Table 2.** Description of experimental parameters

| Parameters | Meaning | Range |
| --- | --- | --- |
| dataset | Dataset used to train the graph neural network | Cora, Citeseer and Polblogs |
| target model[1] | The graph neural network model that will be attacked | 2-layer GCN, GCN-Jaccard and GCN-SVD |
| node ID | Target nodes attacked in single-node experiments | $0 \sim N - 1$(only test set nodes are taken) |
| number of target nodes | Number of target nodes for multi-node experiments | $0 \sim N_U - 1$ |
| resource budget $\beta$ | The maximum number of iterations of the gradient-free optimizer. This parameter controls the number of computational resources. | Positive integers |
| perturbation type | Perturbation of structure or of features | structure, feature, both |
| constraint size $\Delta$ | The maximum number of edges or features that can be modified for each node manipulated by the attacker; this parameter controls the strength of the perturbation. | For structural perturbations: $1 \sim N$ For feature perturbations: $1 \sim d$ |
| scenario | Control whether the final perturbation is injected before training(poisoning) or after training(escape) | Poisoning or Evasion |
| DFOer | Indicates which derivative-free optimizer in Nevergrad is selected | OnePlusOne, DiscreteOnePlusOne, DoubleFastGA, etc. |

[1] The same hyperparameters are chosen for the target model 2-layer GCN, GCN-Jaccard and GCN-SVD: hidden layer dimension is 16, dropout rate is 0.5, learning rate is 0.1, and weight decay is $5 * 10^{-4}$.

Besides DeepRobust, we also use packages include: Python 3.7; PyTorch 1.8.1; Nevergrad 0.4.3 post2; H5py 3.2.1, etc. In terms of hardware, the processor used for the experiment was a 2.6 GHz hexa-core Intel Core i7.

**Single-node Experiments (RQ1)**

In this part, we attack just one node with different DFOers. We conducted Experiment 1 in order to find an appropriate DFOer by comparing their attack loss curves and

323 classification margins. In this experiment, different DFOers are taken to attack node 0
324 of Cora to let the model classify the node as class 6 instead of its original class 5.
325    **Experiment 1: Attack by 5 different DFOers on node 0 of Cora**

**Table 3.** Parameter settings of Experiment 1

| Dataset | Model[1] | Node ID | $\beta$ | Type[2] | $\Delta$ | Scenario |
|---------|----------|---------|---------|---------|----------|----------|
| Cora | 2-layer GCN | 0 | 100 | Structure | 5 | Poisoning |

[1,2] Refers to "target model" and "perturbation type".

**Table 4.** Results of Experiment 1

| DFOer | Origin Class | Target Class | New Class | Result | Margin |
|-------|--------------|--------------|-----------|--------|--------|
| OnePlusOne | 5 | 6 | 6 | MS[1] | -0.67 |
| DiscreteOnePlusOne | 5 | 6 | 6 | MS | -0.75 |
| DoubleFastGA[2] | 5 | 6 | 6 | MS | -0.70 |
| DE | 5 | 6 | 6 | MS | -0.65 |
| RandomSearch | 5 | 6 | 6 | MS | -0.67 |

[1] Means "misleading success".
[2] The full name is "DoubleFastGAOnePlusOne".

326    Table 3 and 4 show the parameter settings and results of Experiment 1 respectively.
327 Figure 3 to 7 show the loss curves and class probability results of the 5 different
328 DFOers. The figures with mark "A" show the attack loss curves of different DFOers.
329 The horizontal coordinate is the number of iterations while the vertical coordinate is
330 the value of the loss function. The smaller the loss function value, the greater the
331 difference between the target class probability value and the original class probability
332 value, the better the attack effect. The figures marked "B" and "C" show the class
333 probabilities before and after the attacks. The horizontal coordinate is class labels, and
334 the vertical coordinate refers to the probability that the node belongs to a certain class.
335 The figures marked "B" shows the classification probabilities on unperturbed graphs and
336 the figures marked "C" shows the probabilities on perturbed graphs. In this experiment,
337 the greater the probability of class 6 and the smaller the probability of class 5, the better
338 the experiment result is.
339    From the experimental result figures, we can see that various DFOers can all success-
340 fully mislead the model in our algorithm. As shown in Figure 3A, Figure 6A, and Figure
341 7A, OnePlusPne, DE, and RandomSearch can all obtain minimum values of -0.4~-0.6,
342 which can achieve less than -0.65 of the classification margin. However, judging from
343 the oscillation degree of the attack loss curves, they do not have the potential of continu-
344 ous optimization and cannot continuously reduce the value of the loss function with the
345 increase of iteration number.
346    From Figure 4A and Figure 5A, it can be seen that DiscreteOnePlusOne and Dou-
347 bleFastGA can quickly find the appropriate optimization direction and optimize the loss
348 function stably, and can reach a minimum value of about -0.8, corresponding to less
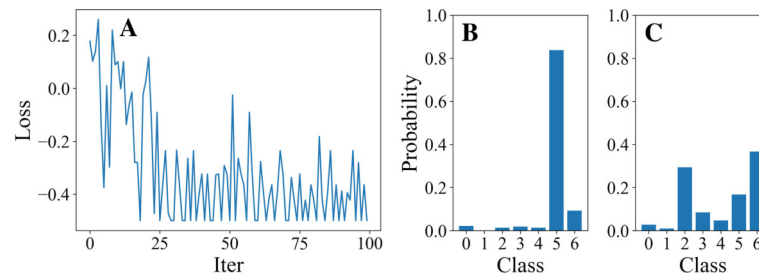
**Figure 3.** OnePlusOne
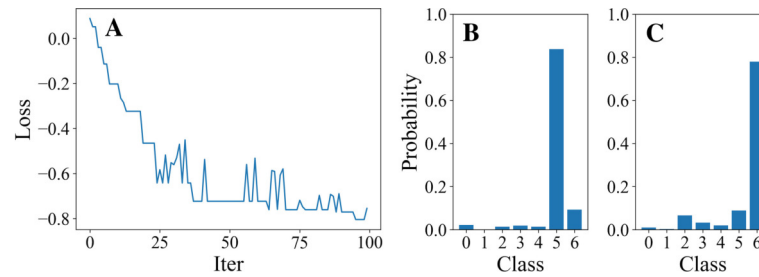
**Figure 4.** DiscreteOnePlusOne
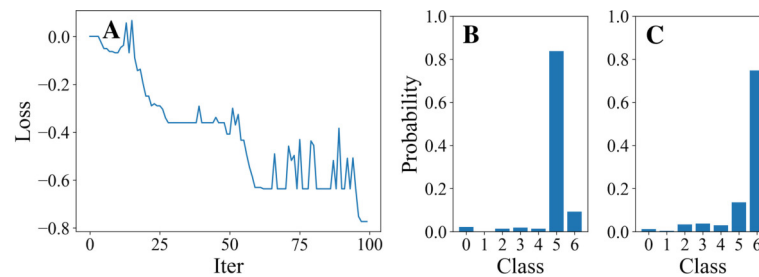
**Figure 5.** DoubleFastGA
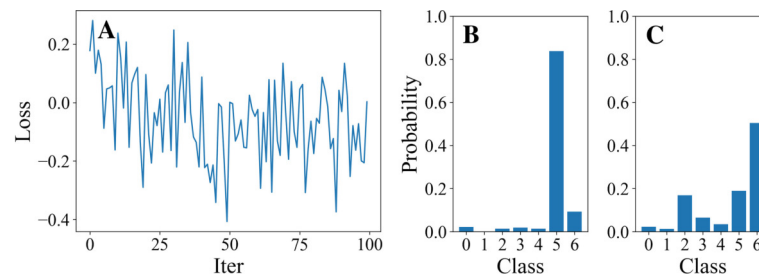
**Figure 6.** DE
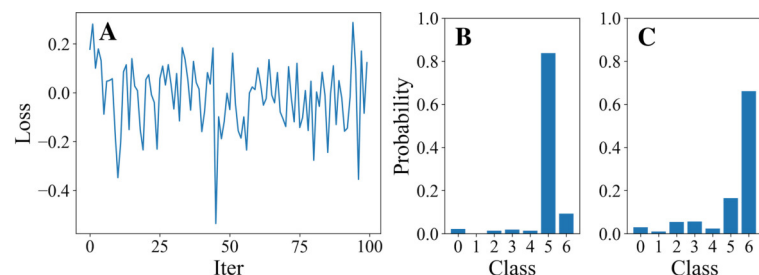
**Figure 7.** RandomSearch

349 than -0.7 of the attack power. However, during the running stage, DiscreteOnePlusOne
350 often generates perturbations that violate the constraints, which is very inefficient. Dou-
351 bleFastGA can combine speed and accuracy, so this optimizer is selected for the later
352 experiments.

### Multi-node Experiments (RQ2)

354 In this part, we choose a set of target nodes from the test set and attack them separately.
355 Every time we attack a node, we will test and record whether the attack was successful.
356 We define Success Rate (SR) as the ratio of the "successful attack" number to the total
357 attacks number, Misleading Rate (MR) as the ratio of the "misleading success" number
358 to the number of the total attacks. In Experiment 2 & 3, we try to find the effect of
359 different parameter settings on SR and MR.

360     **Experiment 2: The effect of different resource budget $\beta$**

**Table 5.** Parameter settings of Experiment 2

| Dataset | Number[1] | Type | Δ | Model | Scenario | DFOer |
|---------|-----------|------|---|-------|----------|-------|
| Cora | 50 | Structure | 5 | 2-layer GCN | Poisoning | DoubleFastGA |

[1] Number of the target nodes.

**Table 6.** Results of Experiment 2

| $\beta$ | SR(MR) | MR/SR | Running Time | Running Time (Improved) |
|---------|--------|-------|--------------|-------------------------|
| 10 | 0.68(0.54) | 79.41% | 5m3s | 3m39s |
| 25 | 0.82(0.72) | 87.80% | 9m15s | 5m28s |
| 50 | 0.80(0.72) | 90.00% | 18m33s | 10m30s |
| 100 | 0.94(0.86) | 91.49% | 37m11s | 18m37s |
| 200 | 0.94(0.86) | 91.49% | 1h14m37s | 33m35s |

361     The parameter settings and results of Experiment 2 are shown in Table 5 and 6. The
362 SR directly reflects the attack effectiveness of the algorithm: the higher the SR, the
363 better the effectiveness. The MR reflects the directional misleading ability of the attack:
364 the higher the ratio of MR to SR, the more directional the attack is.

365     The SR gradually increases with the increase of resource budget (i.e., the number
366 of iterations), but the time consumption is proportional to the number of iterations.
367 The ratio of MR to SR increases with the resource budget, which indicates that the
368 DoubleFastGA is capable to find the appropriate optimization direction to make it
369 more probable to classify nodes to the target class. When $\beta = 100$, the attack reaches
370 the balance of efficiency and performance. It can be seen that even with the smallest
371 resource budget ($\beta = 10$), 68% of the nodes are attacked successfully and 54% are
372 misled successfully. When $\beta \geq 25$, the SR can exceed 80%.

373     During the experiment, we found that each iteration needs to perturb at the original
374 image, which requires many deep copy operations on the original image which consumes

a lot of computational time. We use "inverse perturbation" to solve this problem. During
each iteration, we directly perturb the original graph data, and then "inversely perturb"
the graph after each query to restore it to the original data. This approach greatly reduces
the computational time of deep copies for a large number of iterations. As shown in the
last two columns of Table 6, the time consumption after improvement is about 50% of
that before.

**Experiment 3: Influence of constraint Δ and perturbation type on attack effect**

**Table 7.** Parameter settings of Experiment 3

| Dataset | Number | Δ | Model | Scenario | DFOer |
|---------|--------|---|-------|----------|-------|
| Cora | 50 | 5 | 2-layer GCN | Poisoning | DoubleFastGA |

**Table 8.** Results of Experiment 3

| Δ | Feature | Structure | Both[1] |
|---|---------|-----------|---------|
| 2 | 0.10(0.08)[2] | 0.60(0.56) | 0.70(0.68) |
| 4 | 0.14(0.14) | 0.88(0.84) | 0.92(0.88) |
| 6 | 0.22(0.20) | 0.96(0.86) | 0.94(0.94) |
| 8 | 0.24(0.24) | 0.92(0.80) | 0.98(0.80) |
| 10 | 0.30(0.30) | 0.98(0.92) | 0.98(0.94) |

[1] When structure perturbations and feature pertur-
bations are carried out simultaneously, each type
of perturbations has a constraint of Δ.
[2] Represents "SR(MR)".

Table 7 shows the parameter settings of Experiment 3. It can be seen from Tabel
8 that the structure perturbation effect is better than the feature perturbation. The
combination of structure perturbation and feature perturbation can achieve a stronger
attack effect but requires a more tolerant constraint (the sum of structural constraint and
feature constraint). Only structure perturbations were used in subsequent experiments.

As the perturbation constraint size increases, the optimizer can search in a larger
space. Obviously, a larger constraint size leads to a better attack effect.

## Comprehensive Experiments (RQ3)

In this part we conduct Experiment 4 & 5 to find how DFDA performs under 3 different
models: base model 2-layer GCN mentioned in Equation (2), GCN-Jaccard (Wu et al.,
2019) and GCN-SVD (Entezari et al., 2020). Experiment 4 is conducted in an evasion
scenario where the model doesn't retrain, while Experiment 5 is in a poisoning scenario
that needs retraining before attacks. Moreover, we compare DFDA with classic method
*Nettack* in Experiment 5. We randomly attack 30 test set nodes with DFDA (using the
DoubleFastGA optimizer) and *Nettack* respectively. We repeated the process five times
and calculated the mean and standard deviation of the SRs to make the comparison more
plausible. Both Experiment 4 & 5 are conducted on Cora, Citeseer and Polblogs.

399 **Experiment 4: Effectiveness of DFDA under different defensive models in eva-**
400 **sion attack scenario**

**Table 9.** Parameter settings of Experiment 4

| Dataset | Number | Type | $\beta$ | Scenario | DFOer |
|---------|--------|------|---------|----------|-------|
| Cora | 50 | Structure | 100 | Evasion | DoubleFastGA |

**Table 10.** Results of Experiment 4

| $\Delta$ | 2-layer GCN | GCN-Jaccard | GCN-SVD |
|----------|-------------|-------------|---------|
| 2 | 0.46(0.46) | 0.63(0.53) | 0.63(0.60) |
| 4 | 0.86(0.80) | 0.83(0.76) | 0.80(0.73) |
| 6 | 0.86(0.83) | 0.93(0.83) | 0.90(0.76) |

401 Parameter settings of Experiment 4 are shown in Table 9. As we can see from Table
402 10, DFDA can achieve good results in the evasion scenario. In this scenario, both of
403 the defense models cannot effectively defend against adversarial samples. This may
404 be due to the fact that the defense mechanisms (e.g., graph purification mechanism of
405 GCN-Jaccard and low-rank approximation processing of GCN-SVD) are essentially
406 pre-processing of the input data. As the model doesn't retrain, the defense mechanisms
407 are bypassed in our evasion setup. This why the success rate on the defense models is
408 abnormally high.
409 **Experiment 5: Comparison of DFDA and *Nettack* in different defense models**
410 **under poisoning attack scenarios**

**Table 11.** Parameter settings of Experiment 5

| Number | Type | $\beta$ | Scenario | DFOer |
|--------|------|---------|----------|-------|
| 30 | Structure | 100 | Poisoning | DoubleFastGA |

411 Table 11 shows the parameter settings of Experiment 5. The results are shown
412 in Table 12 and are visualized in 9 line graphs from Figure 8 to 16. With $\beta = 100$
413 and $\Delta = 8$, DFDA achieves a maximum average SR of 95.3%, 98.0% and 70.0% on
414 Cora, Citeseer and Polblogs, respectively. DFDA performs more excellent on Cora and
415 Citeseer than on Polblogs. This is because there are only two class labels in Polblogs
416 and the edges are denser than Cora and Citeseer, which would make Polblogs more
417 difficult to attack.
418 The performance of DFDA on three models: 2-layer GCN > GCN-Jaccard > GCN-
419 SVD. The GCN-Jaccard is less defensive against DFDA. This may be because the
420 adversarial perturbations generated by the DFDA algorithm do not easily reduce the
421 Jaccard similarity between nodes. So it is not easy to be defended by GCN-Jaccard.
422 GCN-SVD is more defensive against DFDA because the variation of high-rank singular

**Table 12.** Results of Experiment 5

| Dataset | β | 2-layer GCN | | GCN-Jaccard | | GCN-SVD | |
|---|---|---|---|---|---|---|---|
| | | **DFDA** | **Nettack** | **DFDA** | **Nettack** | **DFDA** | **Nettack** |
| Cora | 2 | 62.7±9.2 | **69.3±3.7** | 39.3±7.2 | **42.7±14.0** | **20.0±7.8** | 10.0±6.7 |
| | 4 | **85.3±6.9** | 78.7±10.2 | 58.0±9.3 | **61.3±11.5** | **48.6±11.9** | 22.7±6.4 |
| | 6 | **95.3±6.9** | 88.0±3.8 | **73.3±9.4** | 72.0±12.2 | **68.0±5.6** | 42.7±9.5 |
| | 8 | **95.3±3.8** | 85.3±3.8 | **80.0±8.5** | 79.3±6.4 | **67.3±5.5** | 56.0±15.3 |
| Citeseer | 2 | **70.0±14.5** | 53.3±9.7 | **54.7±9.6** | 39.3±8.6 | **41.3±7.7** | 32.0±8.7 |
| | 4 | **83.3±7.5** | 73.3±6.2 | **72.0±7.7** | 61.3±5.1 | **65.3±10.2** | 54.7±9.0 |
| | 6 | **88.7±8.0** | 68.0±6.1 | **81.3±6.5** | 70.0±4.1 | **73.3±5.3** | 68.7±5.1 |
| | 8 | **98.0±1.8** | 74.0±6.4 | **88.0±8.0** | 77.3±9.8 | **78.7±6.5** | 72.0±6.9 |
| Polblogs | 2 | **51.3±5.1** | 28.0±9.6 | **41.3±21.0** | 31.3±10.2 | **7.3±4.3** | 1.3±1.8 |
| | 4 | **56.7±10.3** | 54.0±2.8 | **44.7±17.7** | 30.0±4.7 | **13.3±6.7** | 6.7±5.3 |
| | 6 | **69.3±2.8** | 58.0±3.8 | **41.3±21.9** | 32.0±3.8 | **23.3±10.3** | 14.0±6.0 |
| | 8 | **70.0±7.8** | 64.7±9.6 | **39.3±21.1** | 35.3±8.7 | **34.6±6.9** | 31.3±17.2 |

* The data represent the means and standard deviations of SRs. The bold indicates the average SR of the algorithm that performs better under the same conditions.
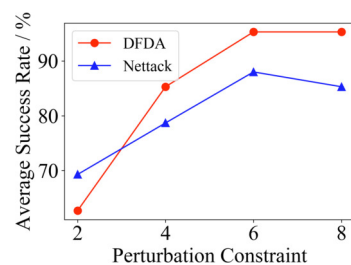
**Figure 8.** Average SR on 2-layer GCN & Cora
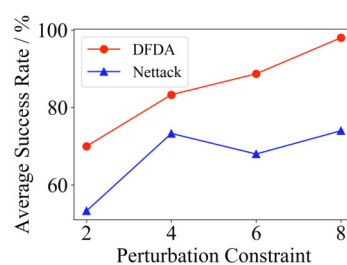
**Figure 9.** Average SR on 2-layer GCN & Citeseer
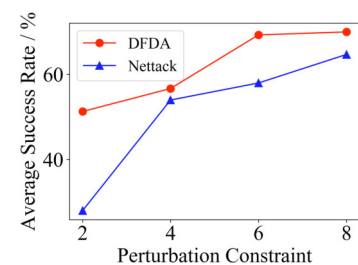
**Figure 10.** Average SR on 2-layer GCN & Polblogs
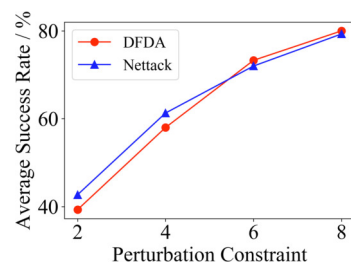
**Figure 11.** Average SR on GCN-Jaccard & Cora
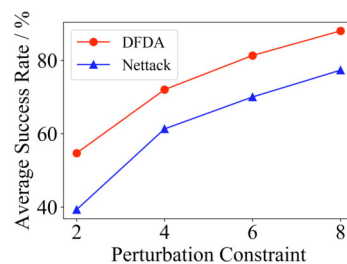
**Figure 12.** Average SR on GCN-Jaccard & Citeseer
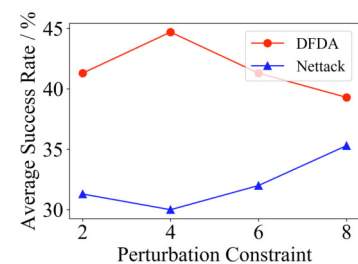
**Figure 13.** Average SR on GCN-Jaccard & Polblogs
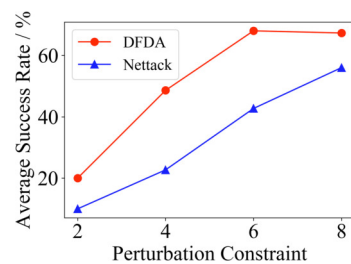
**Figure 14.** Average SR on GCN-SVD & Cora
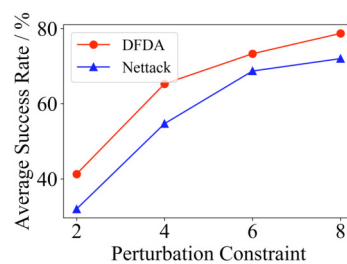
**Figure 15.** Average SR on GCN-SVD & Citeseer
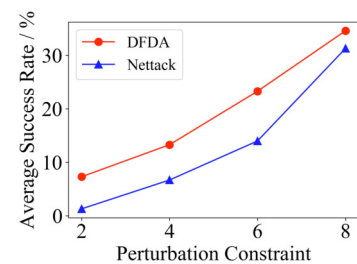
**Figure 16.** Average SR on GCN-SVD & Polblogs

⁴²³ components in the adversarial sample spectrum generated by DFDA is large. Therefore,
⁴²⁴ the perturbations generated by DFDA will be easily filtered by the matrix approximation
⁴²⁵ algorithm.

⁴²⁶     DFDA generally outperforms *Nettack* on three datasets and three target models.
⁴²⁷ *Nettack* uses training labels to train the surrogate model and generates adversarial
⁴²⁸ samples by attacking it. This algorithm uses greedy ideas in optimization and attacks
⁴²⁹ much faster, but it cannot optimize iteratively. DFDA can increase the number of
⁴³⁰ iterations to raise the SR by spending more time. The time complexity of *Nettack* is
⁴³¹ related to the constraint size ($O(\Delta)$), while the time complexity of DFDA is related to
⁴³² the number of iterations ($O(\beta)$).

⁴³³     Of the parameters involved in the above experiments, the three parameters - scenario,
⁴³⁴ perturbation type and target model - are used to distinguish between different types
⁴³⁵ of adversarial attacks, in decreasing order of importance. The scenario distinguishes
⁴³⁶ between poisoning attacks and evasion attacks, the perturbation type distinguishes
⁴³⁷ between structure and feature perturbations, and the target model distinguishes between
⁴³⁸ attacks under different defense models.

⁴³⁹     There are three numerical type parameters that influence the effectiveness of the
⁴⁴⁰ attack. In descending order of importance, they are constraint size, resource budget and
⁴⁴¹ number of target nodes. Constraint size determines the number of the edges or features
⁴⁴² that can be manipulated and directly controls the ease of the attack task. The resource
⁴⁴³ budget determines the number of iterations. A large resource budget can increase the
⁴⁴⁴ success rate of the attack to some extent. The number of target nodes determines the
⁴⁴⁵ stability of the success rate of the attack. The larger the nodes number, the less the
⁴⁴⁶ fluctuation in success rate.

## CONCLUSION

⁴⁴⁸ In this paper, we focus on the use of derivative-free optimization (DFO) ideas in
⁴⁴⁹ graph adversarial attacks. We first introduce a DFO-based black-box adversarial attack
⁴⁵⁰ framework against GCNs. Then we implement a direct attack algorithm (DFDA) using
⁴⁵¹ *Nevergrad* library, using which we can easily compare the performance of various
⁴⁵² derivative-free optimizers on node classification attack tasks. Moreover, we solve the
⁴⁵³ problem of large search space by declining the perturbation vector dimension. Finally,
⁴⁵⁴ we conducted three kinds of experiments on Cora, Citeseer and Polblogs. The results
⁴⁵⁵ show that DFDA outperforms *Nettack* in most cases. It can achieve an average success
⁴⁵⁶ rate of more than 95% on Cora when perturbing at most 8 edges, which demonstrates

that our method can fully exploit the potential of DFO methods in node classification adversarial attacks. In the future, we will focus on the application of DFO ideas to indirect attacks and global attacks.

## REFERENCES

Carlini, N. and Wagner, D. (2018). Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE.

Chen, J., Chen, L., Chen, Y., Zhao, M., Yu, S., Xuan, Q., and Yang, X. (2019). GA-Based Q-Attack on Community Detection. *IEEE Transactions on Computational Social Systems*, 6(3):491–503.

Conn, A. R., Scheinberg, K., and Vicente, L. N. (2009). *Introduction to derivative-free optimization*. SIAM.

Dai, H., Li, H., Tian, T., Xin, H., Wang, L., Jun, Z., and Le, S. (2018). Adversarial attack on graph structured data. In Dy, J. G. and Krause, A., editors, *35th International Conference on Machine Learning, ICML 2018*, volume 3 of *Proceedings of Machine Learning Research*, pages 1799–1808. PMLR.

Entezari, N., Al-Sayouri, S. A., Darvishzadeh, A., and Papalexakis, E. E. (2020). All you need is Low (rank): Defending against adversarial attacks on graphs. In Caverlee, J., Hu, X. B., Lalmas, M., and Wang, W., editors, *WSDM 2020 - Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 169–177. ACM.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., and Wang, M. (2020). Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 639–648, New York, NY, USA. Association for Computing Machinery.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. OpenReview.net.

Li, Y., Jin, W., Xu, H., and Tang, J. (2020). DeepRobust: A PyTorch Library for Adversarial Attacks and Defenses. *CoRR*, abs/2005.0.

Ma, Y., Wang, S., Derr, T., Wu, L., and Tang, J. (2019). Attacking Graph Convolutional Networks via Rewiring. *CoRR*, abs/1906.0.

Papernot, N., Mcdaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016). The limitations of deep learning in adversarial settings. In *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 372–387. IEEE.

Rapin, J. and Teytaud, O. (2018). Nevergrad - A gradient-free optimization platform. \url{https://GitHub.com/FacebookResearch/Nevergrad}.

Rossi, R. A. and Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In Bonet, B. and Koenig, S., editors, *Proceedings*

501    *of the National Conference on Artificial Intelligence*, volume 6, pages 4292–4293.
502    AAAI Press.

503 Ryu, S., Kwon, Y., and Kim, W. Y. (2019). A bayesian graph convolutional network for
504    reliable prediction of molecular properties with uncertainty quantification. *Chemical*
505    *science*, 10(36):8438–8446.

506 Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The
507    graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

508 Sun, L., Dou, Y., Yang, C., Wang, J., Yu, P. S., He, L., and Li, B. (2018). Adversarial
509    attack and defense on graph data: A survey. *arXiv preprint arXiv:1812.10528*.

510 Ughi, G., Abrol, V., and Tanner, J. (2020). An empirical study of derivative-free-
511    optimization algorithms for targeted black-box attacks in deep neural networks. *arXiv*
512    *preprint arXiv:2012.01901*.

513 Wang, H., Zhao, M., Xie, X., Li, W., and Guo, M. (2019). Knowledge graph convo-
514    lutional networks for recommender systems. In *The World Wide Web Conference*,
515    WWW '19, page 3307–3313, New York, NY, USA. Association for Computing
516    Machinery.

517 Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., and Zhu, L. (2019). Adversarial
518    examples for graph data: Deep insights into attack and defense. In Kraus, S., editor,
519    *IJCAI International Joint Conference on Artificial Intelligence*, volume 2019-Augus,
520    pages 4816–4823. ijcai.org.

521 Zügner, D., Akbarnejad, A., and Günnemann, S. (2018). Adversarial attacks on neural
522    networks for graph data. In Guo, Y. and Farooq, F., editors, *Proceedings of the ACM*
523    *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages
524    2847–2856. ACM.

525 Zügner, D. and Günnemann, S. (2019). Adversarial attacks on graph neural networks via
526    meta learning. In *7th International Conference on Learning Representations, ICLR*
527    *2019*. OpenReview.net.