Blindfolded Attackers Still Threatening: Strict Black-Box Adversarial Attacks on Graphs

Jiarong Xu[1], Yizhou Sun[2], Xin Jiang[2], Yanhao Wang[1], Chunping Wang[3], Jiangang Lu[1] and Yang Yang[1]*

[1]Zhejiang University, [2]University of California, Los Angeles, [3]FinVolution Group
{xujr, wangyanhao, lujg, yangya}@zju.edu.cn, yzsun@cs.ucla.edu, jiangxjames@ucla.edu, wangchunping02@xinye.com

Abstract

Adversarial attacks on graphs have attracted considerable research interests. Existing works assume the attacker is either (partly) aware of the victim model, or able to send queries to it. These assumptions are, however, unrealistic. To bridge the gap between theoretical graph attacks and real-world scenarios, in this work, we propose a novel and more realistic setting: *strict black-box graph attack*, in which the attacker has no knowledge about the victim model at all and is not allowed to send any queries. To design such an attack strategy, we first propose a *generic graph filter* to unify different families of graph-based models. The strength of attacks can then be quantified by the change in the graph filter before and after attack. By maximizing this change, we are able to find an effective attack strategy, regardless of the underlying model. To solve this optimization problem, we also propose a relaxation technique and approximation theories to reduce the difficulty as well as the computational expense. Experiments demonstrate that, even with no exposure to the model, the Macro-F1 drops 5.5% in node classification and 29.5% in graph classification, which is a significant result compared with existent works.

1 Introduction

Graph-based models, including graph neural networks (GNNs) (Kipf and Welling 2017; Veličković et al. 2018) and various random walk-based models (Lovász 1993; Perozzi, Al-Rfou, and Skiena 2014), have achieved significant success in numerous domains like social network (Pei, Chakraborty, and Sycara 2015), bioinformatics (Gilmer et al. 2017), finance (Paranjape, Benson, and Leskovec 2017), *etc.* However, these approaches have been shown to be vulnerable to adversarial examples (Jin et al. 2020), namely those that are intentionally crafted to fool the models through slight, even human-imperceptible modifications to the input graph. Therefore, adversarial attack presents itself as a serious security challenge, and is of great importance to identify the weaknesses of graph-based models.

Over recent years, extensive efforts have been devoted to studying adversarial attacks on graphs. According to the amount of knowledge accessible to the attacker, we summarize five categories of existing works in Table 1. As the first attempt, white-box attack assumes the attacker has full access to the victim model (Wu et al. 2019; Xu et al. 2019a; Chen et al. 2018; Wang et al. 2018). Then gray-box attack is proposed, in which the attacker trains a surrogate model based on limited knowledge of the victim as well as task-specific labeled data (Zügner, Akbarnejad, and Günnemann 2018; Zügner and Günnemann 2019). However, most real-world networks are unlabeled and it is arduously expensive to obtain sufficient labels, which motivates the study of (restricted) black-box attacks. With no access to the correct labels, restricted black-box attack shows its success but still needs limited knowledge of the victim model (Chang et al. 2020). Comparatively, the black-box model is not aware of the victim model, but has to query some or all examples to gather additional information (Dai et al. 2018; Yu et al. 2020).

Information accessible	White-box	Gray-box	Restricted black-box	Black-box	*Strict black-box*
Model parameters	√	×	×	×	×
Labels	√	√	×	×	×
Queries	√	×	×	√	×
Model structure	√	◯	◯	×	×

Table 1: Different graph adversarial attacks, categorized by whether the attacker has full (√), limited (◯), or no (×) access to the model.

Despite these research efforts, there still exists a considerable gap between the existing graph attacks and the real-world scenarios. In practice, the attacker cannot fetch even the basic information about the victim model (like whether the model is GNN-based or random-walk-based). Moreover, it is also unrealistic to assume that the attacker can query the victim model in real-world applications. Such querying attempts will be inevitably noticed and blocked by the defenders. For example, consider a credit risk model built upon the user lending network in a commercial bank. This model only serves the bank's internal personnel, but is completely unavailable to the public or credit card users. When malicious users want to improve their credit scores via some attack strategies, they have no knowledge of the victim model (including model structure and parameters) and do not even have access to queries or labels at all.

In view of the above limits, we propose a new attack strategy, *strict black-box graph attack (STACK)*, which has no knowledge of the victim model and no access to queries or

*Corresponding author.

labels at all. The design of such attack strategies is nontrivial due to the following challenges.

- Most existing graph attack strategies are model-specific, and are not extendable to the strict black-box setting. For example, the attack designed for the low-pass GNN filter will inevitable perform badly on a GNN model that covers almost all frequency profiles because the attacker might target at some irrelevant bands of frequency. Thereby, the first challenge is to *identify a common cornerstone for various types of graph-based models.*
- Existing works always use model predictions or the feedback from surrogate models to quantify the effect of attacks. However, with no access to the queries and the correct labels, we can neither measure the quality of predictions, nor refer to a surrogate model. So the second challenge is *the effective quantification and efficient computation of the strength of graph attacks.*

To handle the above challenges, we first propose a generic graph filter which formulates the key components of various graph models into a compact form. Then we are able to measure the strength of attacks by the change of the proposed graph filter before and after attack. An intuitive but provably effective attack strategy is to maximize this change within a fixed amount of perturbation on the original graph. To obtain a suboptimal solution efficiently, we further relax the objective of this problem to a function of the eigenvalues, and then show that the relaxed objective can be approximated efficiently via eigensolution approximation theories. Besides the reduction in computational complexity, this approximation technique also captures inter-dependency between edge flips, which is ignored in previous works (Bojchevski et al. 2019; Chang et al. 2020). In addition, a restart mechanism is applied to prevent accumulation of approximation errors without sacrificing the accuracy.

We summarize our major contributions as follows:

- We bring attention to a critical yet overlooked *strict black-box* setting in adversarial attacks on graphs.
- We propose a generic graph filter applicable to various graph models, and also develop an efficient attack strategy to select adversarial edges.
- Extensive experiments demonstrate that even when the attacker is unaware of the victim model, the Macro-F1 in node classification drops 5.5% and that of graph classification drops 29.5%, both of which are significant results compared with SOTA approaches.

The success of our STACK method breaks the illusion that perfect protection for the victim model could block all kinds of attacks: even when the attacker is totally unaware of the underlying model and the downstream task, it is still able to launch an effective attack.

2 Strict Black-box Attacks (STACK)

Let $G = (V, E)$ be an undirected, unweighted graph with the node set $V = \{v_1, v_2, ..., v_N\}$ and the edge set $E \subseteq V \times V$. The adjacency matrix A of graph G is an $N \times N$ symmetric matrix with elements $A_{ij} = 1$ if $\{i, j\} \in E$ or $i = j$, and $A_{ij} = 0$ otherwise. We also intentionally set the diagonal elements of A to 1. Denote by D the diagonal degree matrix with $D_{ii} = \sum_{i=1}^{N} A_{ij}$.

Suppose we have a graph model designed for certain downstream tasks, *e.g.*, node classification, or graph classification. The attacker is asked to modify the graph structure and/or node attributes within a fixed budget such that the performance of downstream task degrades as much as possible. More specifically, we assume that the attacker will add or delete a limited number of edges from G, resulting in a *perturbed graph* $G' = (V, E')$. This setting is conventional in previous works (Chen et al. 2018; Bojchevski et al. 2019; Chang et al. 2020), and also practical in many scenarios like link spam farms (Gyongyi and Garcia-Molina 2005), Sybil attacks (Yu et al. 2006), and *etc*. Towards more practical scenarios, we assume the attacker can neither access any knowledge about the victim model nor query any examples. We call such attack as the *strict black-box attack (STACK)* on graphs. The attacker's goal is to figure out which set of edges flips can fool various graph-based victim models when they are applied to different downstream tasks. The generality of the proposed attack model is guaranteed by a generic graph filter, which is introduced in §2.1, and in §2.2 we formulate an optimization problem for construction of such a model.

2.1 Generic Graph Filter

Without access to any information from victim models and queries, we need to take into account the common characteristics of various graph-based victim models when designing the adversarial attack. Thus we propose a generic graph filter S:

$$S = D^{-\alpha} A D^{-1+\alpha}, \tag{1}$$

where $\alpha \in [0, 1]$ is a given parameter to enlarge the filter family. Many common choices of graph filters can be considered as special cases of the generic graph filter S. For instance, when $\alpha = 1/2$, the corresponding graph filter $S_{\text{sym}} = D^{-1/2} A D^{-1/2}$ is symmetric, and used in numerous applications, including spectral graph theory (Chung 1997) and the graph convolutional networks (GCNs) (Kipf and Welling 2017; Veličković et al. 2018). Another common choice of α is $\alpha = 1$, and the resulting graph filter $S_{\text{rw}} = D^{-1} A$ is widely used in many applications related to random walks on graphs; see, for example, (Chung 1997). Many graph properties, *e.g.*, network connectivity, centrality, can also be expressed in terms of the proposed generic graph filter (Lovász 1993). Furthermore, it is easy to see the relation between the graph Laplacian ($L = D - A$) and S, *i.e.*, $S = I - D^{-\alpha} L D^{-1+\alpha}$, where the change in S and the change in $D^{-\alpha} L D^{-1+\alpha}$ is exactly the same.

The proposed generic graph filter has many interesting properties. For example, its eigenvalues are invariant under isomorphic transformations. Consequently, many intrinsic spectral properties are preserved regardless of the choice of α, including distances between vertices, graph cuts, the stationary distributions of a Markov chain (Cohen et al. 2017)). In addition, the eigenvalues of S are bounded in $[-1, 1]$. Hence, the generic graph filter of the original graph G and that of the perturbed graph G' are more comparable from the perspective of graph isomorphism and spectral properties; see §3 for more details.

2.2 The Optimization Problem

With the mere observation of the input graph, our attacker aims to find the set of edge flips that would change the graph filter S most. Considering the dependency structure between the nodes, one adversarial perturbation is easy to propagate to other neighbors via relational information. Therefore, the generic graph filter S is used instead of the adjacency matrix A, because the impact of one edge flip on S can properly depict such cascading effects on a graph. Similar ideas are conventional in most graph learning models (Chung 1997). To flip the most influential edges, we formulate our attack model as the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}_1\left(A';\alpha\right) = \left\|(S')^k - S^k\right\|_F \\ \text{subject to} \quad & A'_{ii} = 1, \quad i = 1,\dots,N \\ & A'_{ij} \in \{0,1\} \text{ for } i \neq j \\ & \|A' - A\|_0 \leq 2\delta, \end{aligned} \tag{2}$$

where the optimization variable is the $N \times N$ symmetric matrix A'. The matrix power S^k takes account of all the neighbors *within* k hops (instead of those k-hop neighbors) because A is defined to include self-loops. When targeting on localized kernels (*e.g.*, GCN), we prefer $k = 1$ or 2, while a larger k is recommended to capture higher-order structural patterns. The last inequality constraint shows the budget δ with $0 < \delta \ll |E|$, which is consistent with most previous works (Jin et al. 2020; Dai et al. 2018). The left part of Figure 1 gives an example of the proposed problem.

3 Methodology

The formulation of Problem (2) is not ideal for our purpose. First, the objective $\mathcal{L}_1(A', \alpha)$ is affected by the choice of α. This would hurt the generality of our model as we seek for a unified black-box attack strategy. Moreover, the evaluation of the objective function involves a costly eigenvalue decomposition, which in turn makes the whole problem expensive to solve. Therefore, in this section, we aim to find a sub-optimal solution to Problem (2), with a reasonable computation expense. This process is illustrated in Figure 1. In §3.1 we relax the original objective to a lower bound, which only involves the spectrum of graph filter; this corresponds to the upper-right part of Figure 1. Then, we give an approximate solution to compute the perturbed spectrum in §3.2, as shown in the lower right corner of Figure 1. A detailed description of how to select adversarial edges is provided in §3.3 for solving the relaxed problem. Finally, in §3.4 we show that our model is flexibly extendable when more information is available.

3.1 A Viable Lower Bound for $\mathcal{L}_1$

Although the objective function $\mathcal{L}_1(A';\alpha)$ directly shows the difference between the original graph filter and the perturbed one, it has one significant drawback: the objective $\mathcal{L}_1(A';\alpha)$ is affected by the parameter α in S. This means that different choices of α in S will result in different solutions to Problem (2). This would largely hurt the generalization of our model as we are in favor of one stable strategy for different choices of graph filters. Hence, a relaxation for Problem (2) that is immune to the choice of α would be desirable.

To find such a relaxation, we first observe that the eigenvalues of S are independent of α, due to the following lemma.

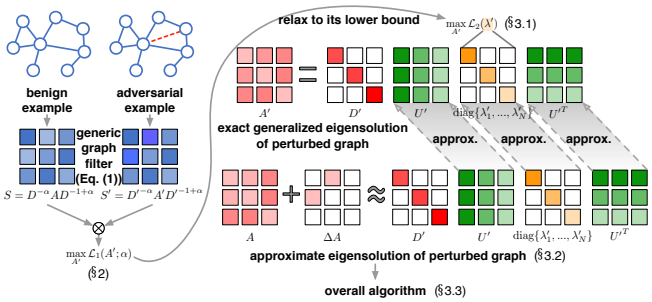


Figure 1: Overview of strict black-box adversarial attacks on graphs.

Lemma 1. *λ is an eigenvalue of $S = D^{-\alpha}AD^{-1+\alpha}$ if and only if (λ, u) solves the generalized eigenproblem $Au = \lambda Du$.*

With Lemma 1, we are able to construct a convenient lower bound for $\mathcal{L}_1$, involving only the eigenvalues of S'.

Theorem 1. *The function $\mathcal{L}_1(A';\alpha)$ is lower bounded by*

$$\mathcal{L}_1(A';\alpha) \geq \left|\sqrt{\textstyle\sum_{i=1}^{N}(\lambda'_i)^{2k}} - \sqrt{\textstyle\sum_{i=1}^{N}\lambda_i^{2k}}\right| = \mathcal{L}_2(\lambda'), \tag{3}$$

where λ_i and λ'_i are the i-th generalized eigenvalue of A and A', respectively, i.e., $Au_i = \lambda_i Du_i$ and $A'u'_i = \lambda'_i D'u'_i$. We assume that both eigenvalue sequences are numbered in a non-increasing order.

Now the problem of maximizing $\mathcal{L}_1\left(A';\alpha\right)$ can be properly relaxed to the problem of maximizing its lower bound $\mathcal{L}_2(\lambda')$. Intuitively, $\mathcal{L}_2(\lambda')$ is exactly describing the spectral change, which is a natural measure used in spectral graph theory (Chung 1997). Additionally, (3) is also valid for any symmetric matrices S and S', and hence the lower bound $\mathcal{L}_2(\lambda')$ holds for different types of networks (*e.g.*, weighted or unweighted) and different perturbation scenarios (*e.g.*, adjusting edge weights).

3.2 Approximating Perturbed Eigensolution

Although $\mathcal{L}_2(\lambda')$ is a desirable lower bound for its flexibility, its computation remains a great challenge for the following reasons. First, the computation of $\mathcal{L}_2(\lambda')$ relies on the eigenvalue decomposition of A', which costs $\mathcal{O}(N^3)$ flops. What's worse, this decomposition has to be re-evaluated again and again for every possible set of flipped edges, which turns out to be a computational bottleneck. Moreover, the inter-dependency between adversarial edges should be taken into consideration. For example, the spectral change caused by flipping two edges e_1 and e_2 is not identical to the sum of the change caused by flipping e_1 and that caused by e_2. This difference has been ignored in most studies (Bojchevski et al. 2019; Chang et al. 2020) but might turn out to be critical in designing adversarial attacks. In this work, however, we fill this gap by describing such dependency as a combinatorial optimization problem, which is usually difficult to solve.

To resolve the above issues, we can apply the first-order eigenvalue perturbation theory (Stewart and Sun 1990) to approximate $\mathcal{L}_2$. This approximation is accurate enough for

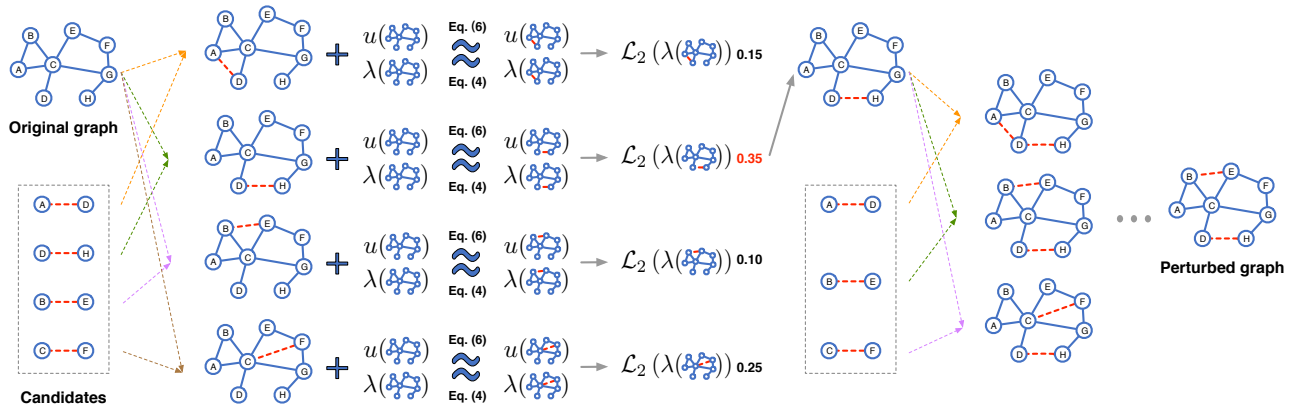


Figure 2: An illustrative example of the proposed attack procedure on graphs, in which the budget constraint δ is set to be 2.

our purpose because the perturbation on graphs is assumed to be extremely small, *i.e.*, $\delta \ll |E|$. We start with the simplest case where only one edge is flipped (Stewart and Sun 1990).

Theorem 2. *Let $\Delta A = A' - A$ and $\Delta D = D' - D$ be the perturbations of A and D, respectively. Let (λ_k, u_k) be the k-th generalized eigen-pair of A, i.e., $Au_k = \lambda_k Du_k$. Assume these eigenvectors are properly normalized so that $u_j^T Du_i = 1$ if $i = j$ and 0 otherwise. When only one edge $\{p, q\}$ is flipped, the perturbed k-th generalized eigen-pair can be approximated by*

$$\lambda'_k \approx \lambda_k + \Delta A_{pq}\big(2u_{kp} \cdot u_{kq} - \lambda_k(u_{kp}^2 + u_{kq}^2)\big) \tag{4}$$

$$u'_k \approx \left(1 - \frac{1}{2}\Delta A_{pq}\left(u_{kp}^2 + u_{kq}^2\right)\right) u_k + \sum_{i \neq k} \frac{\Delta A_{pq}\left(u_{ip}u_{kq} + u_{iq}u_{kp} - \lambda_k\left(u_{ip}u_{kp} + u_{iq}u_{kq}\right)\right)}{\lambda_k - \lambda_i} u_i, \tag{5}$$

where u_{kp} is the p-th entry of the vector u_k.

For one edge flip, the approximation of the eigenvalue λ'_k requires only $\mathcal{O}(1)$ flops to compute (*cf.*, Eq. (4)). By ignoring dependency among adversarial edges, some previous works directly adopt Theorem 2 to approximate the spectrum change caused by each edge flip and choose the edges with the highest spectral impact to perturb (Bojchevski et al. 2019; Chang et al. 2020). To explore and utilize such dependency, we update the eigenvalues and eigenvectors after each edge flip, and then choose the subsequent edge flips based on the updated eigensolutions. However, the computational cost associated with re-evaluating the eigenvectors via Eq. (5) is still too high ($\mathcal{O}(N^2)$ flops for each). What's worse, Eq. (5) is valid only when all eigenvalues are distinct, which is not guaranteed in practice. Therefore, inspired by the power iteration, we propose the following theorem to approximate the perturbed eigenvectors.

Theorem 3. *Let $\Delta A = A' - A$ and $\Delta D = D' - D$ denote the perturbations of A and D, respectively. Moreover, the number of non-zero entries of ΔA and ΔD are assumed to be much smaller than that of A and D. Let u_k be the k-th generalized eigenvector of A with generalized eigenvalue λ_k. We first assume that the eigenvectors are properly normalized so that $\|u_k\|_2 = 1$. The k-th generalized eigenvector u'_k can then be approximated by*

$$u'_k \approx \begin{cases} \mathbf{sign}(\lambda_k)u_k + \dfrac{\Delta C u_k}{|\lambda_k|}, & \text{if } \lambda_k \neq 0 \\ \dfrac{\Delta C u_k}{\|\Delta C u_k\|_2}, & \text{if } \lambda_k = 0, \end{cases} \tag{6}$$

where $\Delta C = (D + \Delta D)^{-1}(A + \Delta A) - D^{-1}A$. Specifically, when one edge $\{p, q\}$ is flipped, only the p-th and q-th elements of u'_k will be changed because only specific elements of ΔC are non-zero, i.e.,

$$\Delta C_{ij} = \begin{cases} A'_{pj}/D'_{pp} - A_{pj}/D_{pp}, & \text{if } j \in \mathcal{N}(p) \bigcup \{p, q\} \\ A'_{qj}/D'_{qq} - A_{qj}/D_{qq}, & \text{if } j \in \mathcal{N}(q) \bigcup \{p, q\} \\ 0, & \text{otherwise,} \end{cases}$$

where $\mathcal{N}(p)$ indicates the set of neighbors of node p.

When only one edge is flipped, Eq. (6) suggests that the approximation of the perturbed eigenvector requires only $\mathcal{O}(N)$ flops for each, which is far more efficient than $\mathcal{O}(N^2)$ utilizing Eq. (5). In addition, the evaluation of u'_k in Eq. (6) involves only the k-th eigenvalue λ_k, which removes the strict assumption regarding distinct eigenvalues.

3.3 Generating Adversarial Edges

Up to this point, we have aimed to flip δ edges so that $\mathcal{L}_2(\lambda')$ is maximized. Specifically, we first form a candidate set by randomly sampling several edge pairs, as in (Bojchevski et al. 2019). Our attack strategy involves three steps: (i) for each candidate, compute its impact $\mathcal{L}_2(\lambda')$ on the original graph, such that the eigenvalues can be approximated via Eq. (4); (ii) flip the candidate edge that scores highest on this metric; and (iii) recompute the eigenvalues (Eq. (4)) and eigenvectors via Eq. (6) after each time an edge is flipped. These steps are repeated until δ edges have been flipped. Figure 2 briefly illustrates such adversarial attack procedure.

Restart mechanism. In the above strategy, the approximation of the statistic in step (iii) is efficient. However, it inevitably leads to serious error accumulation as more edges are flipped. A straightforward solution is to reset the aggregated error (*i.e.*, recompute the exact eigensolutions) over

periodic time. Thus, the question of when to restart should be answered carefully. Recall that in Theorem 2, the perturbed eigenvectors must be properly normalized so that $(u_i')^T D' u_i' = 1$. Thus, we can perform an orthogonality check to verify whether the eigenvectors have been adequately approximated. Theoretically, if the perturbed eigenvectors are accurate enough and normalized properly, then $U'^T D' U'$ (u' is one column of U') will be close to the identity matrix. Thus, we propose to use the average of the magnitudes of the non-zero off-diagonal terms of $M = (U')^T D' U'$ to infer the approximation error of the perturbed eigenvectors:

$$\epsilon = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j \neq i} |M_{ij}|. \tag{7}$$

The smaller value of ϵ we get, the more accurate our approximation is. Thereby, the approximation error ϵ is monitored in every iteration, and restart is performed when the error exceeds a threshold τ. This restart technique ensures approximation accuracy and also reduces time complexity.

3.4 Extension to Different Knowledge Levels

Our proposed adversarial attack can also be easily extended to boost performance when additional knowledge (*e.g.*, model structure, learned parameters) is available. One straightforward way of doing this would be adopt $\mathcal{L}_2(\lambda')$ as a regularization term. Thereby, complementary to other types of adversarial attacks (*e.g.*, white- or gray-box), the proposed attack model facilitates rich discrimination on global changes in the spectrum.

As an example of the extension to gray-box attack, assume our goal is to attack a GNN designed for the semi-supervised node classification task. The attacker is able to alter both the graph structure and the node attributes, and its goal is to misclassify a specific node v_i. In this case, a surrogate model is built with all non-linear activation function removed (Zügner, Akbarnejad, and Günnemann 2018); this is denoted as $Z = \text{softmax}(S^k X W)$, where X denotes the feature matrix and W represents the learned parameters. Therefore, the combined attack model tries to solve the following optimization problem:

$$\text{maximize} \max_{c \neq c_0} ((S'^k X' W)_{v_i c} - (S'^k X W)_{v_i c_0}) + \gamma \mathcal{L}_2(\lambda')$$

where the variables are the graph structure A' and the feature matrix X', the scalar c_0 denotes the true label or the predicted label for v_i based on the unperturbed graph G, while c is the class of v_i to which the surrogate model assigns. The constant $\gamma > 0$ is a regularization parameter. Note that this case can also be considered as an extension to targeted attack.

4 Experiments

In this section, we evaluate the performance of the proposed method on both node classification and graph classification task. We also extend our model to white-box or targeted attacks, as explained in §3.4. In addition, we study the gap between the initial formulation (2) and the relaxed problem.

4.1 Experimental Setup

Datasets. For node-level attacks, we adopt Cora-ML, Citeseer and Polblogs for node classification task, and follow the preprocessing in (Dai et al. 2018). We additionally adopt a real-world mobile network built upon call logs provided by China Telecom, which consists 34,210 nodes and 187,845 edges. We obtain users' demographic information as node attributes, and label a user as the fraudster if he/she was reported in Baidu or Qihoo 360.

Baselines. We consider the following baselines.

- *Rand.*: this method randomly flips edges.
- *Deg./Betw./Eigen.* (Bojchevski et al. 2019): the flipped edges are selected in the decreasing order of the sum of the degrees, betweenness, or eigenvector centrality.
- *DW*: a black-box attack method designed for DeepWalk (Bojchevski et al. 2019).
- *GF-Attack*: targeted attack under the strict black-box setting (Chang et al. 2020). We directly adopt it under our untargeted attack setting by selecting the edge flips on the decreasing order of the loss for SGC/GCN.
- *GPGD* (Graph Projected Gradient Descent): we apply the GPGD algorithm (Xu et al. 2019a) to solve Problem (2).
- *STACK-r-d*: a variant of our method, where both the restart mechanism and the dependency among edge flips are not considered here. Specifically, the edge flips are selected on the decreasing order according to Eq. (4) in Theorem 2.

		(Unattacked)	*Rand.*	*Deg.*	*Betw.*	*Eigen.*	*DW*	*GF-Attack*	*GPGD*	*STACK-r-d*	*STACK-r*	*STACK*	White-box
Cora-ML	GCN	0.82±0.8	1.97±0.8	1.12±0.4	1.22±0.4	0.28±0.3	0.85±0.3	1.34±0.5	4.22±0.6	4.03±0.6	5.02±0.4	**5.27**±0.3	11.36±0.5
	Node2vec	0.79±0.8	6.37±1.8	5.40±1.6	3.33±1.0	2.84±1.0	3.25±1.3	5.76±1.5	5.33±1.8	5.82±1.7	6.92±1.0	**8.29**±1.0	(1.43±0.9)
	Label Prop.	0.80±0.7	4.10±1.3	2.45±0.7	2.71±0.8	2.07±0.7	1.79±0.9	3.18±0.5	4.28±1.6	5.01±0.7	6.02±0.9	**7.13**±0.9	(1.05±1.0)
Citeseer	GCN	0.66±1.4	2.02±0.6	0.16±0.4	0.70±0.4	0.64±0.4	0.21±0.4	1.36±0.7	2.14±0.9	2.63±0.7	3.16±0.6	**3.98**±0.5	6.42±0.6
	Node2vec	0.60±1.5	7.47±2.3	7.47±1.6	3.47±2.6	4.87±1.5	2.54±2.5	6.45±3.5	5.26±1.9	7.94±1.6	8.32±2.5	**9.32**±2.6	(0.12±1.0)
	Label Prop.	0.64±0.8	6.70±2.0	3.47±0.8	6.00±1.7	5.36±0.6	3.00±0.8	6.99±1.0	5.14±1.9	6.66±1.3	7.79±0.9	**8.16**±0.9	(2.47±1.2)
Polblogs	GCN	0.96±0.7	1.91±1.5	0.03±0.2	1.72±0.6	0.67±0.5	0.01±0.4	1.15±0.4	2.35±1.8	3.06±1.2	4.30±1.2	**5.32**±1.1	3.88±1.1
	Node2vec	0.95±0.3	3.01±0.7	0.04±0.6	3.07±0.6	1.84±0.3	0.18±0.4	1.00±0.5	2.49±0.6	2.57±0.9	2.74±0.5	**3.79**±0.5	(2.13±0.4)
	Label Prop.	0.96±0.5	4.99±0.7	0.08±0.4	3.45±0.7	2.15±0.3	0.37±0.5	2.18±0.4	4.15±0.8	5.17±0.8	5.84±0.7	**6.14**±0.7	(2.28±0.5)
Telecom	GCN	0.80±0.9	1.90±1.4	1.22±0.5	1.23±0.4	0.94±0.5	0.97±0.4	1.90±0.5	2.89±1.0	3.03±1.3	3.96±1.2	**4.24**±1.2	5.57±1.0
	Node2vec	0.76±0.7	4.95±1.9	3.70±1.1	2.98±1.0	2.77±1.1	3.01±1.4	5.01±1.5	5.85±1.6	5.77±1.7	5.89±1.6	**6.88**±1.6	(1.01±0.9)
	Label Prop.	0.75±0.5	4.04±1.9	3.10±0.7	3.87±0.8	2.98±0.8	2.66±0.8	4.45±0.9	5.03±1.1	4.99±1.6	5.07±1.5	**5.97**±1.5	(1.57±0.9)

Table 2: We apply various node-level attacks to different graphs models and different datasets. We report the decrease in Macro-F1 score (in percent) on the test set after the attack is performed; the higher the better. We also report the Macro-F1 on the unattacked graph.

		Rand.	*Deg.*	*Betw.*	*Eigen.*	*DW*	*GF-Attack*	*GPGD*	*STACK-r-d*	*STACK-r*	*STACK*
Proteins	GIN	9.05	9.31	12.50	8.00	13.44	10.82	9.49	11.48	12.81	**13.53**
	Diffpool	24.13	11.27	9.87	11.03	12.71	21.99	14.53	23.49	24.87	**24.88**
Enzymes	GIN	32.76	34.38	34.75	**40.25**	38.76	35.48	35.63	36.00	37.46	39.90
	Diffpool	38.09	17.48	9.51	17.74	13.55	37.19	20.32	38.18	**40.18**	39.62

Table 3: Graph-level attacks against GIN and Diffpool. We report the decrease in the Macro-F1 score (in percent) on test set. Higher numbers indicate better performance.

	Nettack	*STACK-ext*
Cora-ML	55.70 (0.53)	**59.75** (0.57)
Citeseer	63.41 (0.60)	**65.58** (0.63)
Polblogs	5.16 (0.23)	**5.40** (0.23)

Table 4: Extension to *Nettack*. We report the average decrease in prediction confidence (in percent) of true labels and the misclassification rate in parentheses.

- *STACK-r*: another variant of our method, where the restart mechanism is not considered here.

Implementation details. For the node classification task, we choose GCN (Kipf and Welling 2017), Node2vec (Grover et al. 2016) and Label Propagation (Xiaojin and Zoubin 2002) as the victim models. We set the training/validation/test split ratio as 0.1:0.1:0.8, and allow the attacker to modify 10% of the total edges. Comparatively, GIN (Xu et al. 2019b) and Diffpool (Ying et al. 2018) are used as victim models in the graph classification task due to their excellence. We follow the default setting in the above models, including the split ratio, and allow the attacker to modify 20% of the total edges.

Throughout the experiment, we follow the strict black-box setting. We set the spatial coefficient $k = 1$ and the restart threshold $\tau = 0.03$. The candidate set of adversarial edges is randomly sampled in every trial, and its size is set as 20K. As shown in Table 2, this randomness does not hurt the performance overall. The reported results are all averaged over 10 trials. Our codes are available at: https://github.com/galina0217/stack.

4.2 Experimental Results

Node classification task. Table 2 reports the decrease in Macro-F1 score (in percent) in node classification for all the three datasets and three victim models. We can see that our method performs the best across all datasets. First, heuristics (*Rand.*, *Deg.*, *Betw.*, and *Eigen.*) fail to find the most influential edges in strict black-box setting. Previous works (*DW* and *GF-Attack*) do not perform well either, mainly due to the blindness to additional information (*i.e.*, model type). The limit of *GPGD* is probably attributed to its relaxation from the binary graph topology to the convex hull. The ablation study on *Ours-r-d* and *Ours-r* further highlights the significance of the edge dependencies and the restart mechanism. In addition, the last column in Table 2 shows the results of white-box attacks which applies *GPGD* for GCN (Xu et al. 2019a), which shares the same untargeted attack setting as ours, and is one of the simplest and best-performed white-box approach. The original paper presents great success when using GCN as the victim model. But our experiments show the failure of its intuitive extension to node-embedding models (*e.g.*, Node2vec) and diffusion models (*e.g.*, Label Propagation).

Table 5 further shows that under increasing perturbation rates (5/10/15%), our node-level attacker can do more damage to GCN while still achieves the best performance. Note that when the perturbation rate is not very high, our solution without restart is already good enough to mount attacks.

Graph classification task. Table 3 reports the results in graph classification. Especially, our method is 2.65% on average better than other methods in terms of Macro-F1. Interestingly, *STACK-r* (without restart) sometimes performs well, indicating that we can apply a version with lower complexity in practice yet not sacrificing much accuracy.

Attacks on Defensive models. We further validate the effectiveness of our attacks against three defensive models: EdgeDrop (Dai et al. 2018), Jaccard (Wu et al. 2019) and SVD (Entezari et al. 2020). We utilize GCN as the backbone classifier and test on node classification task. We compare our method with the strongest competitor *GPGD*. The experimental settings are the same as those used in node-level attack. The results in Table 6 demonstrate that our method outperforms *GPGD* in terms of its ability to attack the defensive models. We can see that EdgeDrop cannot effectively defend against our attack, while the two pre-processing defense approaches (Jaccard and SVD) can defend against our attack to a certain extent.

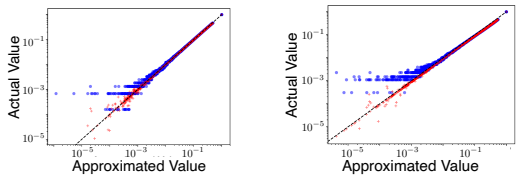
Extension when additional knowledge is available. Here we explore whether our attack can be successfully extended as the example in §3.4. Specifically, we focus on the *Nettack* model (Zügner, Akbarnejad, and Günnemann 2018) and follow their targeted attack settings. We randomly select 30 correctly-classified nodes from the test set and mount targeted attacks on these nodes. From Table 4, we see that the naive extension of our method can further drop 4.84% in terms of prediction confidence and increase 4.18% in terms of misclassification rate on average.

attacker \ pert. rate	5%	10%	15%	attacker \ pert. rate	5%	10%	15%
Rand.	0.75	1.97	2.41	*GPGD*	3.94	4.22	5.03
Deg.	0.59	1.12	1.34	*GF-Attack*	1.10	1.34	2.11
Betw.	0.63	1.22	1.45	*STACK-r-d*	3.90	4.03	5.11
Eigen.	0.30	0.28	1.12	*STACK-r*	**4.43**	5.02	5.77
DW	0.34	0.85	1.23	*STACK*	4.30	**5.27**	**6.40**

Table 5: Decrease in Macro-F1 score with different perturbation rates when attacking GCN on Cora-ML.

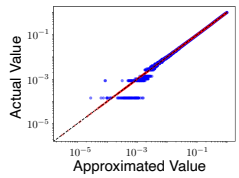
	w/o defense	EdgeDrop	Jaccard	SVD
GPGD	4.22	6.04	3.94	3.47
STACK	5.27	7.11	4.71	4.02

Table 6: Attack against defensive models on Cora-ML. The decrease in Macro-F1 score is reported here.

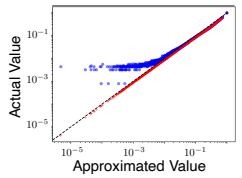


(a) Erdős–Rényi

(b) Barabási–Albert



(c) Watts–Strogatz



(d) Traid Formation

Figure 3: The true eigenvalues (y-axis) are plotted against the approximations (x-axis) in log scale. We present our approximations with (red plus) and without (blue circle) the restart step.

	Cora-ML	Citeseer	Polblogs
Pearson	0.89	0.91	0.85
Spearman	0.91	0.93	0.93

Table 7: The Pearson and Spearman correlation coefficients between $\mathcal{L}_1$ and approximated $\mathcal{L}_2$.

Approximation quality. From the original objective function $\mathcal{L}_1$ to our final model, we have made one relaxation and one approximation; both steps would accumulate errors in evaluating the original objective. One thing worth noting is that the estimation error caused by sampling can be ignored here, as demonstrated in (Bojchevski et al. 2019).

To evaluate the effectiveness of eigen-approximation, we first generate the following random graphs with 1K nodes: Erdős–Rényi (Bollobás 2013), Barabási–Albert (Albert and Barabási 1998), Watts–Strogatz (Watts and Strogatz 1998), and Triad Formation (Holme and Kim 2002). Then we flip 10 edges in every synthetic graph and compare the true eigenvalues with approximated ones. Figure 3 presents the average results of 100 repeated experiments, where x-axis denotes the approximated value, and y-axis indicates the actual value. The approximate linearity of the plotted red points suggests the effectiveness of our eigensolution approximation with restart algorithm (§3.3). As an ablation study, results generated by our algorithm without the restart step (blue points) show a tendency of overestimating the change in eigenvalues before and after attacks.

Furthermore, we study the quality of both the relaxation and the approximation by examining the gap between $\mathcal{L}_1$ and $\mathcal{L}_2$. We compute their Pearson and Spearman correlation coefficients on three real-world datasets in Table 7. Results are all close to 1, which indicates a linear correlation between the original objective $\mathcal{L}_1$ and the approximation of $\mathcal{L}_2$.

5 Related Work

Deep learning on graphs has been shown to be vulnerable to noisy or adversarial examples (Zheng et al. 2021; Xu et al. 2020b, 2021, 2020a). The setting of graph adversarial attacks can be classified into white-box, gray-box, restricted black-box, black-box and strict black-box settings. Getting access to full knowledge (*e.g.*, model structure and learned parameters) about a victim model, the adversaries generally prefer white-box attacks (Wu et al. 2019; Xu et al. 2019a; Chen et al. 2018; Wang et al. 2018; Wang and Gong 2019). However, adversaries may not be able to acquire such perfect knowledge, preventing them from adopting gradient-based algorithms directly. Gray-box attacks are proposed when the attackers are usually familiar with the architecture of the victim models (Chen et al. 2020; Jin et al. 2020). One common method is to train substitute models as surrogates to estimate the information of the victim models (Zügner, Akbarnejad, and Günnemann 2018; Zügner and Günnemann 2019; Bojchevski et al. 2019). Despite its tractability, such surrogate models rely on labels for training and the attack performance suffers the approximation error of surrogate models. So long as less information is exposed, the adversaries will have to adopt restrict black-box attacks or black-box attacks instead. Under restricted black-box attacks, (Chang et al. 2020) assumes the family of graph-based models (*i.e.*, GNN-based or sampling-based) is known and design a graph signal processing-based attack method. For black-box attacks, the adversaries can only access the model as an oracle and may query some or all of the examples to obtain continuous-valued predictions or discrete classification decisions (Dai et al. 2018). Unlike the above-mentioned counterparts, our proposed strict black-box attacks assume the attacker has totally no knowledge of the victim model and queries, which is a more practical but strict setting. Besides, another line of work (Ma, Ding, and Mei 2020), not part of the five categories, assumes the victim model is GNN-based but the training input is partly available, which is anther view of practical merit.

6 Conclusion

In this paper, we describe a strict black-box setting for adversarial attacks on graphs: The attacker not only has zero knowledge about the victim model, but is unable to send any queries as well. To handle this challenging but more realistic setting, a generic graph filter is proposed to unify different families of graph models, and the change in the proposed graph filter is used to quantify the strength of attacks. By maximizing this change, we would be able to find an effective attack strategy. For efficient solution to the problem, we also propose a relaxation technique and an approximation algorithm. Extensive experiments show that the proposed attack strategy substantially outperforms other existing methods. Extension of STACK on node- and attribute-level perturbations would be interesting research directions.

Acknowledgements. This work was partially supported by NSFC (62176233), the National Key Research and Development Project of China (2018AAA0101900), NSF III-1705169, Okawa Foundation Grant, Amazon Research Awards and Picsart gift. Jiarong Xu's work is supported by NSFC (71531006), the Major Scientific Project of Zhejiang Laboratory (2020MC0AE01), the Fundamental Research Funds for the Central Universities (Zhejiang University New Generation Industrial Control System (NGICS) Platform) and the Zhejiang University Robotics Institute (Yuyao) Project (K12001).

References

Albert, R.; and Barabási, A.-L. 1998. Statistical Mechanics of Complex Networks. *Nature*, 393(6684): 440–442.

Bojchevski, A.; et al. 2019. Adversarial Attacks on Node Embeddings via Graph Poisoning. In *ICML*, 695–704.

Bollobás, B. 2013. *Modern graph theory*. Springer Science and Business Media.

Chang, H.; Rong, Y.; Xu, T.; Huang, W.; Zhang, H.; Cui, P.; Zhu, W.; and Huang, J. 2020. A Restricted Black-Box Adversarial Framework Towards Attacking Graph Embedding Models. In *AAAI*, 3389–3396.

Chen, J.; Wu, Y.; Xu, X.; Chen, Y.; Zheng, H.; and Xuan, Q. 2018. Fast Gradient Attack on Network Embedding. *ArXiv*.

Chen, L.; Li, J.; Peng, J.; Xie, T.; Cao, Z.; Xu, K.; He, X.; and Zheng, Z. 2020. A Survey of Adversarial Learning on Graphs. *ArXiv*.

Chung, F. R. K. 1997. *Spectral graph theory*. American Mathematical Soc.

Cohen, M. B.; Kelner, J.; Peebles, J.; Peng, R.; Rao, A. B.; Sidford, A.; and Vladu, A. 2017. Almost-Linear-Time Algorithms for Markov Chains and New Spectral Primitives for Directed Graphs. In *STOC*, 410–419.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial Attack on Graph Structured Data. In *ICML*, 1115–1124.

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 *WSDM*, 169–177.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*, 1263–1272.

Grover, A.; et al. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*, 855–864.

Gyongyi, Z.; and Garcia-Molina, H. 2005. Link spam alliances. In *VLDB*, 517–528.

Holme, P.; and Kim, B. J. 2002. Growing Scale-Free Networks with Tunable Clustering. *Physical review E*, 65(2): 026107.

Horn, R. A.; and Johnson, C. R. 1994. *Topics in matrix analysis*. Cambridge university press.

Jin, W.; Li, Y.; Xu, H.; Wang, Y.; and Tang, J. 2020. Adversarial Attacks and Defenses on Graphs: A Review and Empirical Study. *ArXiv*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

Li, Q.; Han, Z.; Wu, Y.; and Xiao-Ming. 2018. Deeper Insights into Graph Convolutional Networks for Semi-supervised Learning. In *AAAI*, 3538–3545.

Lovász, L. 1993. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1): 1–46.

Ma, J.; Ding, S.; and Mei, Q. 2020. Towards More Practical Adversarial Attacks on Graph Neural Networks. In *NeurIPS*, 4756–4766.

Paranjape, A.; Benson, A. R.; and Leskovec, J. 2017. Motifs in Temporal Networks. In *WSDM*, 601–610.

Pei, Y.; Chakraborty, N.; and Sycara, K. 2015. Nonnegative matrix tri-factorization with graph regularization for community detection in social networks. In *IJCAI*, 2083–2089.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online Learning of Social Representations. In *SIGKDD*, 701–710.

Stewart, G. W.; and Sun, J.-g. 1990. *Matrix Perturbation Theory*. Computer Science and Scientific Computing. Boston: Academic Press.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Wang, B.; and Gong, N. Z. 2019. Attacking Graph-Based Classification via Manipulating the Graph Structure. In *CCS*, 2023–2040.

Wang, X.; Eaton, J.; Hsieh, C.-J.; and Wu, S. F. 2018. Attack Graph Convolutional Networks by Adding Fake Nodes. *ArXiv*.

Watts, D. J.; and Strogatz, S. H. 1998. Collective Dynamics of 'Small-World' Networks. *Nature*, 393(6684): 440–442.

Wu, H.; Wang, C.; Tyshetskiy, Y.; Docherty, A.; Lu, K.; and Zhu, L. 2019. Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *IJCAI*, 4816–4823.

Xiaojin, Z.; and Zoubin, G. 2002. Learning from labeled and unlabeled data with label propagation. *CMU CALD tech report CMU-CALD-02-107*.

Xu, J.; Yang, Y.; Chen, J.; Jiang, X.; Wang, C.; Lu, J.; and Sun, Y. 2020a. Unsupervised Adversarially-Robust Representation Learning on Graphs. *ArXiv*.

Xu, J.; Yang, Y.; Pu, S.; Fu, Y.; Feng, J.; Jiang, W.; Lu, J.; and Wang, C. 2021. NetRL: Task-aware Network Denoising via Deep Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering*, 1–1.

Xu, J.; Yang, Y.; Wang, C.; Liu, Z.; Zhang, J.; Chen, L.; and Lu, J. 2020b. Robust Network Enhancement from Flawed Networks. *IEEE Transactions on Knowledge and Data Engineering*, 1–1.

Xu, K.; Chen, H.; Liu, S.; Chen, P.-Y.; Weng, T.-W.; Hong, M.; and Lin, X. 2019a. Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *IJCAI*, 3961–3967.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019b. How Powerful Are Graph Neural Networks? In *ICLR*.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 4805–4815.

Yu, H.; Kaminsky, M.; Gibbons, P. B.; and Flaxman, A. 2006. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM*, 267–278.

Yu, S.; Zheng, J.; Chen, L.; Chen, J.; Xuan, Q.; and Zhang, Q. 2020. Unsupervised Euclidean Distance Attack on Network Embedding. In *DSC*, 71–77.

Zheng, Q.; Zou, X.; Dong, Y.; Cen, Y.; Yin, D.; Xu, J.; Yang, Y.; and Tang, J. 2021. Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning. *NeurIPS D&B*.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *SIGKDD*, 2847–2856.

Zügner, D.; and Günnemann, S. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.

A Appendix

A.1 Notations

The main notations can be found in the Table 8.

Notation	Description
G, A, S, E	The original graph, the adjacency matrix, the generic graph filter and the edge set of G
G', A', S', E'	The perturbed graph, the adjacency matrix, the generic graph filter and the edge set of G'
u, u'	The generalized eigenvectors of A and A'
λ, λ'	The generalized eigenvalues of A and A'
$\lambda(A), \lambda(A')$	The eigenvalues of A and A'
$\lambda(S), \lambda(S')$	The eigenvalues of S and S'
α	The normalization parameter of the generic graph filter S
δ	The perturbation budget
k	The spatial coefficient
τ	The restart threshold
r	The number of restart
ϵ	The approximation error of the perturbed eigenvectors

Table 8: Definition of major symbols.

A.2 Proofs and Derivations

Proof of Lemma 1. $Au = \lambda Du$ implies

$$(D^{-\alpha}AD^{-1+\alpha})(D^{1-\alpha}u) = \lambda(D^{1-\alpha}u)$$

for any real symmetric A and any positive definite D, which completes the proof. □

Proof of Theorem 1. We first show that $\mathcal{L}_1(A'; 0.5)$ is lower bounded by

$$\begin{aligned}\mathcal{L}_1(A'; 0.5) \geq& \left| \|(S')^k\|_F - \|S^k\|_F \right| \\ =& \left| \sqrt{\sum_{i=1}^{n} \lambda_i(S')^{2k}} - \sqrt{\sum_{i=1}^{n} \lambda_i(S)^{2k}} \right| \\ =& \left| \sqrt{\sum_{i=1}^{N} (\lambda'_i)^{2k}} - \sqrt{\sum_{i=1}^{N} \lambda_i^{2k}} \right|,\end{aligned}$$

where the first line follows from the triangle inequality, the second line holds because S and S' are real symmetric matrices when $\alpha = 0.5$, and the last step follows from Lemma 1.

Now, we observe that $\mathcal{L}_1(A'; \alpha)$ is convex on $[0, 1]$ and that the minimum of $\mathcal{L}_1$ is attained at $\alpha = 0.5$. So, we conclude that $\mathcal{L}_1(A'; \alpha) \geq \left| \sqrt{\sum_{i=1}^{N} (\lambda'_i)^{2k}} - \sqrt{\sum_{i=1}^{N} \lambda_i^{2k}} \right|$ $(\alpha \in [0, 1])$.

By applying the Weyl's inequalities (cf. Theorem 6 in § A.3) (Horn and Johnson 1994) twice, and according to $(d'_1)^{-\alpha} \geq (d'_2)^{-\alpha} \geq \cdots \geq (d'_N)^{-\alpha}$ and $(d'_1)^{-1+\alpha} \geq (d'_2)^{-1+\alpha} \geq \cdots \geq (d'_N)^{-1+\alpha}$ because $\alpha \in [0, 1]$, we have

$$\begin{aligned}\lambda_i(S') &= \lambda_i\left(D'^{-\alpha}A'D'^{-1+\alpha}\right) \\ &\leq \lambda_1\left(D'^{-\alpha}\right)\lambda_i\left(A'D'^{-1+\alpha}\right) \\ &\leq \lambda_1\left(D'^{-\alpha}\right)\lambda_i\left(A'\right)\lambda_1\left(D'^{-1+\alpha}\right) \\ &= {d'_1}^{-\alpha}\lambda_i\left(A'\right){d'_1}^{-1+\alpha} \\ &= \frac{1}{d'_{\min}}\lambda_i\left(A'\right),\end{aligned}$$

where λ_1 is the largest eigenvalue and $d'_1 = d'_{\min}$ is the smallest degree in G'. □

Proof of Theorem 3. According to one iteration of power iteration, when $\lambda_k \neq 0$,

$$\begin{aligned}u'_k &\approx \frac{(D+\Delta D)^{-1}(A+\Delta A)u_k}{\|(D+\Delta D)^{-1}(A+\Delta A)u_k\|_2} \\ &= \frac{(D^{-1}A+\Delta C)u_k}{\|(D^{-1}A+\Delta C)u_k\|_2} \\ &= \frac{\lambda_k u_k + \Delta C u_k}{\|\lambda_k u_k + \Delta C u_k\|_2} \\ &= \frac{\lambda_k u_k + \Delta C u_k}{\sqrt{\lambda_k^2 {u_k}^T u_k + 2\lambda_k {u_k}^T \Delta C u_k + {u_k}^T \Delta C^T \Delta C u_k}} \\ &\approx \frac{\lambda_k u_k + \Delta C u_k}{|\lambda_k|} = \mathbf{sign}(\lambda_k)u_k + \frac{\Delta C u_k}{|\lambda_k|}.\end{aligned} \tag{8}$$

When $\lambda_k = 0$, we can derive from (8) that $u'_k \approx (\Delta C u_k)/(\|\Delta C u_k\|_2)$. □

A.3 Additional Theorems

Theorem 4. *(Eigenvalue perturbation theory (Stewart and Sun 1990)). Let $\Delta A = A' - A$ and $\Delta D = D' - D$ be the perturbations of A and D, respectively. Moreover, the number of non-zero entries of ΔA and ΔD are assumed to be much smaller than that of A and D. Let (λ_k, u_k) be the k-th pair of generalized eigenvalues and eigenvectors of A,* i.e., *$Au_k = \lambda_k Du_k$. We assume that these eigenvectors are properly normalized such that they satisfy ${u_j}^T Du_i = 1$ if $i = j$ and 0 otherwise. Thus, the k-th generalized eigenvalue $\lambda'_k = \lambda_k + \Delta\lambda_k$ can be approximated by*

$$\lambda'_k \approx \lambda_k + {u_k}^T(\Delta A - \lambda_k \Delta D)u_k.$$

Moreover, when the eigenvalues are distinct, the k-th generalized eigenvector u'_k of A' can be approximated by

$$u'_k \approx \left(1 - \frac{1}{2}{u_k}^T \Delta D u_k\right)u_k + \sum_{i \neq k} \frac{{u_i}^T(\Delta A - \lambda_k \Delta D)u_k}{\lambda_k - \lambda_i}u_i.$$

Theorem 5. *(Bound of eigenvalues of S'). The i-th generalized eigenvalue of A' (i.e., the i-th eigenvalue of S') is upper bounded by*

$$\lambda'_i = \lambda_i(S') \leq \frac{1}{d'_{\min}} \cdot \lambda_i(A'), \tag{9}$$

where $d'_{\min}$ is the smallest degree in G', $\lambda_i(S')$ and $\lambda_i(A')$ are the i-th eigenvalue of S' and A', respectively. This suggests that the eigenvalues of S' are always bounded by the eigenvalues of A'.

Theorem 6. *(Weyl's inequality for singular values (Horn and Johnson 1994)). Let two symmetric matrices $P, Q \in \mathbb{R}^{N\times N}$. Then, for the decreasingly ordered singular values σ of P, Q and PQ, we have $\sigma_{i+j-1}(PQ) \leq \sigma_i(Q) \times \sigma_j(Q)$ for any $1 \leq i, j \leq N$ and $i + j \leq N + 1$.*

A.4 Algorithm

Our detailed attack strategy in given in Algorithm 1. Compared with the exact solution that costs $\mathcal{O}(N^3)$, we here present two approximate solutions. The approximate solution with restart mechanism (*i.e.*, $0 < r < \delta$) is able to achieve the best performance, with a speed up of $\sim \delta/r$ compared with the exact one. Even without the restart mechanism (*i.e.*, r=0), our approximate solution is still able to give comparable results (as is shown by the performance of *Ours-r* in the experiments) and meanwhile reduce the runtime complexity to $\mathcal{O}(N^2)$.

Algorithm 1: Overall attack strategy via eigensolution approximation with restart.

Input: Graph $G = (A)$, perturbation budget δ, restart threshold τ.
Output: Modified Graph $G' \leftarrow (A')$.

$A' \leftarrow A$, $D' \leftarrow D$;
Solve the exact eigensolutions: $A'u' = \lambda' D'u'$;
Cand $\leftarrow \text{candidate}(A')$;
while $\|A' - A\|_0 \leq 2\delta$ **do**
  $e' \leftarrow \underset{e\in\text{Cand}}{\text{argmax}}\, \mathcal{L}_2(e)$, which is approximated via Eq. 4;
  $A', D' \leftarrow$ insert or remove e' to/from A';
  Approximately update eigenvalue λ' via Eq. 4 and update eigenvector u' via Eq. 6;
  if $u'! =$ **0 then**
    Normalize the perturbed eigenvector u' s.t. ${u'_j}^T D u'_i = 1$ if $i = j$;
    Compute ϵ via Eq. 7;
    if $\epsilon > \tau$ **then**
      Recompute the exact eigensolutions;
    end if
  else
    Recompute the exact eigensolutions;
  end if
  Cand $\leftarrow$ remove e';
end while
$G' \leftarrow (A')$.

A.5 Dataset Details

Synthetic datasets. We use four synthetic random graphs to evaluate the approximation quality. All synthetic graphs have 1000 nodes and parameters are chosen so that the average degree is approximately 10. Specifically, for the Erdős–Rényi graph, we set the probability for edge creation as 0.01. For the Barabási–Albert graph, we set the number of edges attached from a new node to existing nodes as 5. When generating the Watts–Strogatz graph, each node is connected to its 10 nearest neighbors in a ring topology, and the probability of rewiring each edge is 0.1. When generating growing graphs with the power-law degree distribution (Holme and Kim 2002), the number of random edges to add for each new node is 5, and the probability of adding a triangle after adding a random edge is 0.1.

Real-world datasets. We use three social network datasets: Cora-ML, Citeseer and Polblogs. The former two are citation networks mainly containing machine learning papers. Here, nodes are documents, while edges are the citation links between two documents. Each node has a human-annotated topic as the class label as well as a feature vector. The feature vector is a sparse bag-of-words representation of the document. All nodes are labeled to enable differentiation between their topic categories. Polblogs is a network of weblogs on the subject of US politics. Links between blogs are extracted from crawls of the blog's homepage. The blogs are labelled to identify their political persuasion (liberal or conservative). Detailed statistics of the social network datasets are listed in Table 9.

We also use two protein graph datasets: Proteins and Enzymes. Proteins is a dataset in which nodes represent secondary structure elements (SSEs) and two nodes are connected by an edge if they are neighbors in either the amino-acid sequence or 3D space. The label indicates whether or not a protein is a non-enzyme. Moreover, Enzymes is a dataset of protein tertiary structures. The task is to correctly assign each enzyme to one of the six EC top-level classes. More detailed statistics of the protein graph datasets are listed in Table 10.

dataset	Cora-ML	Citeseer	Polblogs
type	citation network	citation network	web network
# vertices	2,810	2,110	1,222
# edges	7,981	3,757	16,714
# classes	7	6	2
# features	1,433	3,703	0

Table 9: Data statistics of social datasets.

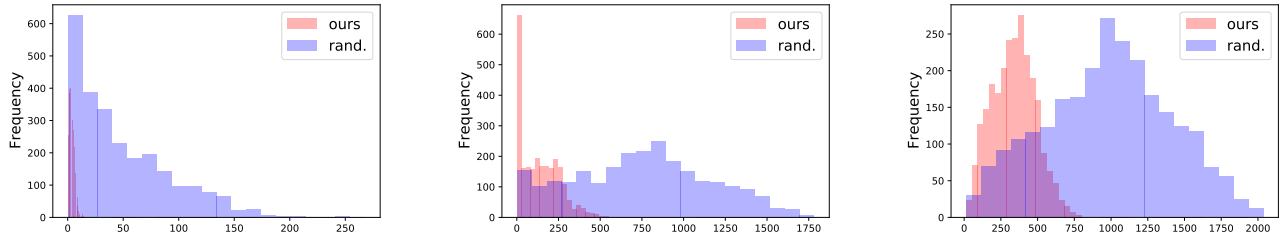
A.6 Implementation Details

Here, we provide additional implementation details of our experiments in § 4.

For GCN, GIN and Diffpool, we use PyTorch Geometric [1] for implementation. We set the learning rate as 0.01 and adopt Adam as our optimizer. For Node2vec, we use its default hyper-parameter setting (Grover et al. 2016), but set the embedding dimension to 64 and use the implementation by CogDL [2]. A logistic regression classifier is then used for classification given the embedding. For Label Propagation, we use an implementation that is adapted to graph-structured data [3].

For all victim models, we tune the number of epochs based on convergence performance. When performing GCN on Cora-ML, Citeseer and Polblogs, the number of epochs is set to 100. When performing GIN or Diffpool on Enzymes, we set the number of epochs to 20, while when performing GIN on Proteins, the number of epochs is set to 10. Finally, when

[1] https://github.com/rusty1s/pytorch_geometric
[2] https://github.com/THUDM/cogdl
[3] https://github.com/thibaudmartinez/label-propagation



(a) Degree centrality (b) Betweenness centrality (c) Eigenvector centrality

Figure 4: The comparison of the centrality distributions for our selected adversarial edges and those of the randomly selected edges on Polblogs dataset.

dataset	Proteins	Enzymes
type	protein network	protein network
# graphs	1,113	600
# classes	2	6
# features	3	3
avg # nodes	39.06	32.63
avg # edges	72.82	62.14

Table 10: Data statistics of protein datasets.

		$\mathcal{L}_1^*$	$\mathcal{L}_2^*$	*STACK*
Cora-ML	GCN	1.92	1.72	**5.27**
	Node2vec	8.05	8.14	**8.29**
	Label Prop.	5.32	5.37	**7.13**
Citeseer	GCN	1.65	2.27	**3.98**
	Node2vec	8.64	9.05	**9.32**
	Label Prop.	7.09	5.41	**8.16**
Polblogs	GCN	2.65	2.92	**5.32**
	Node2vec	3.68	3.43	**3.79**
	Label Prop.	4.80	4.20	**6.14**

Table 11: The experimental results of other objective functions of node-level attacks against three types of victim models. We report the decrease in Macro-F1 score (in percent) on the test set after the attack.

performing Diffpool on Proteins, the number of epochs is set to 2. We conduct all experiments on a single machine of Linux system with an Intel Xeon E5 (252GB memory) and a NVIDIA TITAN GPU (12GB memory).

When implementing baselines, we directly apply the default hyperparameters of *DW* (Bojchevski et al. 2019), another black-box attack (Chang et al. 2020) and *Nettack* (Zügner, Akbarnejad, and Günnemann 2018). For *GPGD* (Xu et al. 2019a) adopting $\mathcal{L}_1(A')$ as the objective, we train 500 epochs with a decaying learning rate as $\text{lr} = \text{lr\_init}(1/(1 + i * \text{decay\_rate}))$ where lr_init is the initial learning rate, decay_rate is the decaying rate and i is the current number of iterations. We set the initial learning rate as 2e-4 with a decaying rate 0.2 on Cora-ML, set the initial learning rate 1e-3 as with a decaying rate 0.2 on Citeseer, Polblogs, and set the initial learning rate 1e-2 as with a decaying rate 0.2 on Proteins and Enzymes. When applying *GPGD* for the white-box setting, we set the learning rate as 10 and run 700 epochs.

A.7 Additional Experimental Results

Analysis of objective function. In addition, we compare with some candidates of objective under the same setting of query-free black-box attacks to validate the decent choice of $\mathcal{L}_2(\lambda')$, which is proposed to capture the adversarial structural changes by the graph filter S. Other candidates, like $\|A' - A\|_F^2$, obviously cannot distinguish the cases when the same number of edges are flipped, while $\mathcal{L}_1^* = \|(A')^k - A^k\|_F^2$ only considers the number of walks between two nodes of length k. Moreover, $\mathcal{L}_2^* = \left\|\sum_{i=1}^k ((A')^i - A^i)\right\|_F^2$ is computationally expensive, also $\mathcal{L}_2^*$ and the above candidates' spectrum are not bounded in the same range across different graphs. We additional present the results (see Table 11) of utilizing these candidates to attack node classification task as the setting of node-level attack. From the results in Table 11, we can see that the superiority of our model, which constantly demonstrates the effectiveness of our objective function.

Can heuristics explain adversarial edges? A most straightforward strategy of identifying adversarial edges is to utilize simple heuristics to capture "important" edges (Bojchevski et al. 2019). However, recall that in Table 2, some results unexpectedly revealed that some heuristic methods (*i.e.*, *Deg.*, *Betw.*, *Eigen.*) sometimes performs worse than randomly selection of adversarial edges. We here analyze this observation by comparing the degree, betweenness or eigenvector centrality distribution of our selected adversarial edges with that of the randomly selected edges. In Figure 4, we find that our selected adversarial edges tend to have smaller degree, betweenness or eigenvector centrality. Whereas, common heuristic methods, following a previous work (Bojchevski et al. 2019), are performed by selecting the adversarial edges with bigger centrality (indicating their importance). Our findings actually run counter to these heuristic methods, thus this is one possible reason why they perform badly. Moreover, Eq. 9 gives us an intuitive spectral view to analyze the degree centrality, *i.e.*, the spectrum of S' is upper bounded by a term w.r.t the smallest degree in G'.

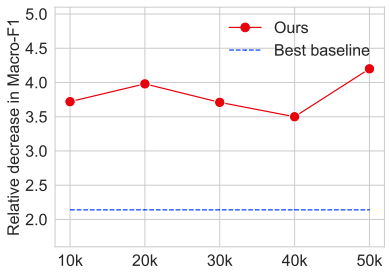


Figure 5: The relative decrease in Macro-F1 score of node-level attacks against GCN with different size of the candidate set on Citeseer dataset. The blue dashed line denotes the performance of the best baseline when candidate size is chosen as 20k.

We therefore propose to select adversarial edges based on the increasing order of the sum of the degree/betweenness/eigenvector centrality (*i.e.*, *SmallDeg.*, *SmallBetw.* and *SmallEigen.*) and report their results as the setting of node-level attack in Table 12. We observe that the smaller centrality does perform better than the larger one in most cases although still can not beat our method.

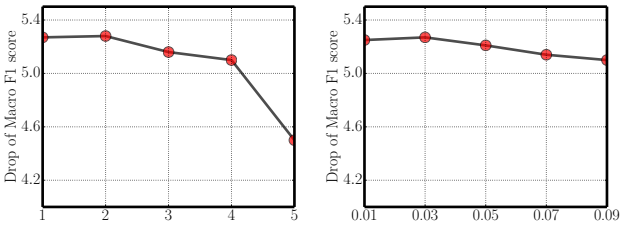
		SmallDeg.	*SmallBetw.*	*SmallEigen.*
Cora-ML	GCN	2.27 ± 0.4	2.29 ± 0.6	1.44 ± 0.4
	node2vec	4.47 ± 0.9	5.64 ± 1.4	4.85 ± 0.8
	Label Prop.	5.14 ± 0.6	5.37 ± 0.5	4.63 ± 0.3
Citeseer	GCN	2.77 ± 0.7	2.81 ± 0.6	2.34 ± 0.8
	node2vec	5.78 ± 2.4	7.99 ± 2.0	2.26 ± 1.7
	Label Prop.	7.66 ± 0.8	7.53 ± 1.3	5.33 ± 0.5
Polblogs	GCN	3.14 ± 0.8	2.64 ± 1.0	3.58 ± 0.7
	node2vec	2.00 ± 0.5	1.33 ± 0.6	1.75 ± 0.7
	Label Prop.	4.81 ± 0.6	4.71 ± 0.7	5.92 ± 0.8

Table 12: The relative decrease in Macro-F1 score of additional heuristic methods with standard deviation of node-level attacks against three types of victim models.

The selection of candidate set. We have mentioned that one limitation of our attack strategy is the randomly selected candidate set, which introduces a further approximation. We here further analyze the impact of the candidate set size. Figure 5 shows the relative decrease when we choose the size of candidate set among $\{10\text{k}, 20\text{k}, 30\text{k}, 40\text{k}, 50\text{k}\}$. We observe that although our model performs slightly better if we randomly select 50k edge pairs as the candidate, the candidate size appears to have little influence on the overall performance, which partly verifies the practical utility of our strategy of randomly selecting candidates.

Parameter Analysis. Our algorithm involves two hyperparameters: namely, the spatial coefficient k and restart threshold τ. We use grid search to find their suitable values on Cora-ML and present here for reference (see Figure 6). Better performance can be obtained when $k \in \{1, 2\}$ and $\tau \leq 0.03$. The observation that our model's superiority when $k \leq 2$ is consistent with many conclusions of graph over-smoothing problem (Li et al. 2018; Klicpera, Bojchevski, and Günnemann 2019). Remarkably, our model can still achieve relatively good performance (*i.e.*, better than the baseline methods) regardless of how hyperparameters are changed.

Detailed results of Table 2. We further report the detailed results (decrease in Macro-F1 score, Macro-Precision score and Macro-Recall score) with standard deviation of node-level attacks in Table 13.



(a) k: spatial coefficient (b) τ: restart threshold

Figure 6: Parameter analysis for node classification task.

			(Unattacked)	*Rand.*	*Deg.*	*Betw.*	*Eigen.*	*DW*	*GF-Attack*	*GPGD*	*STACK-r-d*	*STACK-r*	*STACK*	White-box
Cora-ML	GCN	F1	0.82±0.8	1.97±0.8	1.12±0.4	1.22±0.4	0.28±0.3	0.85±0.3	1.34±0.5	4.22±0.6	4.03±0.6	5.02±0.4	**5.27**±0.3	11.36±0.5
		Prec.	0.81±0.9	0.55±0.5	0.20±0.4	0.11±0.3	0.18±0.3	0.32±0.2	0.41±0.5	1.69±0.6	1.44±0.6	3.02±0.4	**3.43**±0.3	6.58±0.6
		Recall	0.82±1.1	1.15±0.9	0.94±0.4	1.23±0.6	0.49±0.4	0.87±0.4	1.50±0.6	2.51±0.6	2.10±0.6	3.84±0.4	**4.05**±0.4	6.87±0.4
	Node2vec	F1	0.79±0.8	6.37±1.8	5.40±1.6	3.33±1.0	2.84±1.0	3.25±1.3	5.76±1.5	5.33±1.8	5.82±1.7	6.92±1.0	**8.29**±1.0	(1.43±0.9)
		Prec.	0.78±0.7	6.06±1.8	3.04±1.7	3.64±0.9	2.15±1.1	2.96±1.3	5.03±1.4	4.96±1.7	5.21±1.6	5.95±1.0	**8.04**±1.0	(1.39±1.0)
		Recall	0.78±0.9	6.69±2.0	3.76±1.5	4.92±1.3	3.01±1.0	3.57±1.4	5.22±1.6	5.10±1.8	5.30±1.7	6.32±1.0	**8.54**±0.9	(1.52±1.1)
	Label Prop.	F1	0.80±0.7	4.10±1.3	2.45±0.7	2.71±0.8	2.07±0.7	1.79±0.9	3.18±0.5	4.28±1.6	5.01±0.7	6.02±0.9	**7.13**±0.9	(1.05±1.0)
		Prec.	0.81±0.8	2.98±1.3	1.10±0.6	1.20±0.4	1.46±0.7	1.05±0.8	2.74±0.4	3.05±1.0	4.10±0.6	4.62±0.8	**4.98**±1.0	(0.09±0.8)
		Recall	0.80±1.0	4.95±1.2	3.61±0.9	4.32±1.0	2.64±0.9	2.54±1.2	4.71±0.6	5.03±1.9	5.15±0.7	7.92±1.0	**7.99**±1.0	(1.51±1.0)
Citeseer	GCN	F1	0.66±1.4	2.02±0.6	0.16±0.4	0.70±0.4	0.64±0.4	0.21±0.4	1.36±0.7	2.14±0.9	2.63±0.7	3.16±0.6	**3.98**±0.5	6.42±0.6
		Prec.	0.64±1.6	1.15±0.7	0.10±0.5	0.33±0.2	0.30±0.4	0.09±0.5	1.01±0.6	2.01±0.8	2.50±0.7	2.94±0.6	**3.28**±0.7	5.92±0.6
		Recall	0.64±1.6	1.90±0.6	-81.41±245.3	0.72±0.4	0.64±0.5	0.13±0.4	1.16±0.7	2.51±0.9	2.66±0.8	3.62±0.7	**4.00**±0.5	7.06±0.5
	Node2vec	F1	0.60±1.5	7.47±2.3	7.47±1.6	3.47±2.6	4.87±1.5	2.54±2.5	6.45±3.5	5.26±1.9	7.94±1.6	8.32±2.5	**9.32**±2.6	(0.12±1.0)
		Prec.	0.59±1.6	7.28±2.4	7.33±2.1	3.01±2.7	4.26±1.5	2.88±3.2	6.21±3.0	5.00±1.9	7.56±1.5	7.91±2.4	**8.69**±2.7	(0.10±1.0)
		Recall	0.62±1.5	7.24±2.2	6.94±1.6	4.51±2.4	5.00±1.4	2.62±2.5	6.51±3.7	5.86±1.9	8.06±1.6	8.69±2.5	**9.86**±2.7	(0.19±1.0)
	Label Prop.	F1	0.64±0.8	6.70±2.0	3.47±0.8	6.00±1.7	5.36±0.6	3.00±0.	6.99±1.0	5.14±1.9	6.66±1.3	7.79±0.9	**8.16**±0.9	(2.47±1.2)
		Prec.	0.62±0.8	6.01±1.9	2.95±0.6	5.59±1.8	5.06±0.6	2.54±0.6	4.03±0.8	4.52±1.8	6.23±1.3	6.69±1.0	**7.77**±0.9	(2.06±1.1)
		Recall	0.67±0.7	6.88±2.0	3.01±0.9	4.35±1.8	6.02±0.6	3.10±0.9	7.03±1.0	5.45±1.9	7.25±1.4	8.16±0.9	**8.55**±1.0	(2.66±1.2)
Polblogs	GCN	F1	0.96±0.7	1.91±1.5	0.03±0.2	1.72±0.6	0.67±0.5	0.01±0.4	1.15±0.4	2.35±1.8	3.06±1.2	4.30±1.2	**5.32**±1.1	3.88±1.1
		Prec.	0.95±0.7	1.65±1.0	0.02±0.2	1.51±0.5	0.35±0.5	0.02±0.3	0.98±0.3	2.10±1.8	2.86±1.2	4.02±1.1	**4.99**±1.0	3.05±1.0
		Recall	0.96±0.7	1.93±1.5	0.01±0.1	1.74±0.6	0.75±0.5	0.01±0.4	1.17±0.3	2.53±1.9	3.36±1.0	4.58±1.2	**6.01**±1.0	(4.06±1.1)
	Node2vec	F1	0.95±0.3	3.01±0.7	0.04±0.6	3.07±0.6	1.84±0.3	0.18±0.4	1.00±0.5	2.49±0.6	2.57±0.9	2.74±0.5	**3.79**±0.5	(2.13±0.4)
		Prec.	0.95±0.3	2.01±0.6	0.03±0.5	3.06±0.6	1.22±0.4	0.17±0.5	0.86±0.5	2.06±0.7	2.11±0.8	2.65±0.5	**3.51**±0.4	(2.09±0.4)
		Recall	0.95±0.3	3.04±0.7	0.04±0.4	3.04±0.6	2.66±0.3	0.18±0.8	1.12±0.5	2.66±0.5	2.63±0.9	2.94±0.5	**3.99**±0.5	(2.13±0.3)
	Label Prop.	F1	0.96±0.5	4.99±0.7	0.08±0.4	3.45±0.7	2.15±0.3	0.37±0.5	2.18±0.4	4.15±0.8	5.17±0.8	5.84±0.7	**6.14**±0.7	(2.28±0.5)
		Prec.	0.96±0.5	4.41±0.8	0.10±0.4	3.03±0.6	2.45±0.8	0.34±0.4	1.99±0.4	3.51±0.7	4.91±0.8	5.05±0.8	**5.08**±0.7	(2.05±0.5)
		Recall	0.96±0.5	4.67±0.8	0.04±0.4	3.15±0.7	2.05±0.8	-87.62±264.2	2.20±0.4	4.22±0.8	5.30±0.9	5.86±0.7	**6.56**±0.6	(2.51±0.5)

Table 13: The detailed experimental results with standard deviation of node-level attacks against three types of victim models. We report the decrease of performance (in percent) on the test set after the attack is performed; the higher the better (note that negative results denote that the performance adversely increases after the attack). We also report the performance on the unattacked graph.