DATA POISONING ATTACK AGAINST UNSUPERVISED NODE EMBEDDING METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Unsupervised node embedding methods (e.g., DeepWalk, LINE, and node2vec) have attracted growing interests given their simplicity and effectiveness. However, although these methods have been proved effective in a variety of applications, none of the existing work has analyzed the robustness of them. This could be very risky if these methods are attacked by an adversarial party. In this paper, we take the task of link prediction as an example, which is one of the most fundamental problems for graph analysis, and introduce a data poisoning attack to node embedding methods. We give a complete characterization of attacker's utilities and present efficient solutions to adversarial attacks for two popular node embedding methods: DeepWalk and LINE. We evaluate our proposed attack model on multiple real-world graphs. Experimental results show that our proposed model can significantly affect the results of link prediction by slightly changing the graph structures (e.g., adding or removing a few edges). We also show that our proposed model is very general and can be transferable across different embedding methods. Finally, we conduct a case study on a coauthor network to better understand our attack method.

1 INTRODUCTION

Node representations, which represent each node with a low-dimensional vector, have been proved effective in a variety of applications such as node classification (Perozzi et al., 2014), link prediction (Grover & Leskovec, 2016), and visualization (Tang et al., 2016). Some popular node embedding methods include DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015), and node2vec (Grover & Leskovec, 2016). These methods learn the node embeddings by preserving graph structures, which do not depend on specific tasks. As a result, the learned node embeddings are very general and can be potentially useful to multiple downstream tasks.

However, although these methods are very effective and have been used for a variety of tasks, none of the existing work has studied the robustness of these methods. As a result, these methods are susceptible to a risk of being maliciously attacked. Take the task of link prediction in a social network (e.g., Twitter) as an example, which is one of the most important applications of node embedding methods. A malicious party may create malicious users in a social network and attack the graph structures (e.g., adding and removing edges) so that the effectiveness of node embedding methods is maximally degraded. For example, the attacker may slightly change the graph structures (e.g., following more users) so that the probability of a specific user to be recommended/linked can be significantly increased or decreased. Such a kind of attack is known as *data poisoning*. In this paper we are interested in the robustness of the node embedding methods w.r.t. data poisoning, and their vulnerability to the adversarial attacks in the worst case.

We are inspired by existing literature on adversarial attack, which has been extensively studied for different machine learning systems (Szegedy et al., 2013; Goodfellow et al., 2014; Moosavi-Dezfooli et al., 2016; Carlini & Wagner, 2017; Xiao et al., 2018c;b;a; Xie et al., 2017; Cisse et al., 2017; Yang et al., 2018). Specifically, it has been shown that deep neural networks are very sensitive to adversarial attacks, which can significantly change the prediction results by slightly perturbing the input data. However, most of existing work on adversarial attack focus on image (Szegedy et al., 2013; Goodfellow et al., 2014; Moosavi-Dezfooli et al., 2016; Carlini & Wagner, 2017; Xiao et al., 2018c) and text data (Cheng et al., 2018; Jia & Liang, 2017), which are independently distributed while this work focuses on graph data. There are some very recent work which studied adversarial attack for graph data (Dai et al., 2018; Zugner et al., 2018). However, these work mainly studied graph neural networks, which are supervised methods, and the gradients for changing the output label can be leveraged. Therefore, in this paper we are looking for an approach that is able to attack the unsupervised node embedding methods for graphs.

In this paper, we introduce a systematic approach to adversarial attacks against unsupervised node embedding methods. We assume that the attacker can poison the graph structures by either removing or adding edges. Two types of adversarial goals are studied including *integrity attack*, which aims to attack the probabilities of specific links, and *availability attack*, which aims to increase overall prediction errors. We propose a unified optimization framework based on projected gradient descent to optimally attack both goals. In addition, we conduct a case study on a coauthor network to better understand our attack method. To summarize, we make the following contributions:

- We formulate the problem of attacking unsupervised node embeddings for the task of link prediction and introduce a complete characterization of attacker utilities.
- We propose an efficient algorithm based on projected gradient descent to attack unsupervised node embedding algorithms, specifically DeepWalk and LINE, based on the first order Karush Kuhn Tucker (KKT) conditions.
- We conduct extensive experiments on real-world graphs to show the efficacy of our proposed attack model on the task of link prediction. Moreover, results show that our proposed attack model is transferable across different node embedding methods.
- Finally, we conduct a case study on a coauthor network and give an intuitive understanding of our attack method.

2 RELATED WORK

Adversarial attack against image classification has been extensively studied in recent years (Szegedy et al., 2013; Goodfellow et al., 2014; Moosavi-Dezfooli et al., 2016; Carlini & Wagner, 2017; Xiao et al., 2018c;b). However, adversarial attacks against graph have rarely been investigated before. Existing work (Dai et al., 2018; Zugner et al., 2018) on adversarial attacks on graph are limited to graph neural networks (Kipf & Welling, 2017), a supervised learning method. Our work, instead, shows the vulnerabilities of unsupervised methods on graph. Here we briefly summarize previous work on graph embedding methods and then we will give an overview of adversarial attacks on graph data. Last, we will show the related work on the connection to matrix factorization.

Unsupervised Learning on Graph Previous work on knowledge mining in graph has mainly focused on embedding methods, where the goal is to learn a latent embedding for each node in the graph. DeepWalk (Perozzi et al., 2014), LINE (Tang et al., 2015) and Node2vec (Grover & Leskovec, 2016) are the three most representative unsupervised methods on graph.

Adversarial Attack on Graph There are a few work on adversarial attack on graph before. Test time attack on graph convolutional network has been investigated (Dai et al., 2018). Also, poisoning attack against graph is also studied (Zugner et al., 2018). However, they only consider the attack against graph convolutional network. More recently, Bojchevski & Gnnemann (2018) investigates the attack on node embedding methods. Different from this work, they use eigenvalue perturbation theory to generate poisoning graph.

Matrix Factorization Skip-gram model from the NLP community has been shown to be doing implicit matrix factorization (Levy & Goldberg, 2014). Recently, based on the previous work, it has been shown that most of the popular unsupervised methods for graph is doing implicit matrix factorization (Qiu et al., 2018). Moreover, poisoning attack has been demonstrated for matrix factorization problem (Li et al., 2016).

3 PRELIMINARIES

We first introduce the graph embedding problem and link prediction problem. Then we will give an overview of the two existing algorithms for computing the embedding. Given a graph $G = (V, E)$, where V is the node set and $|E|$ is the edge set, the goal of graph embedding methods is to learn a mapping from V to R^d which maps each node in the graph to a d-dimensional vector. We use A to denote the adjacency matrix of the graph G. For each node i, we use Ω_i to denote node i's neighbor. X represents the learnt node embedding matrix where X_i is the embedding of node i.

In link prediction, the goal is to predict the missing edges or the edges that are most likely to emerge in the future. Formally, given a set of node pair $\mathcal{T} \in V \times V$, the task is to predict a score for each node pair. In this paper, we compute the score of each edge from the cosine similarity matrix XX^T.

Now we briefly review two popular graph embedding methods: DeepWalk (Perozzi et al., 2014) and LINE (Tang et al., 2015). DeepWalk extends the idea of Word2vec (Mikolov et al., 2013) to graph, where it views each node as a word and use the generated random walks on graph as sentences. Then it uses Word2vec to get the node embeddings. LINE learns the node embeddings by keeping both the first-order proximity (LINE$_{1st}$), which describes local pairwise proximity, and the second-order proximity (LINE$_{2nd}$) for sampled node pairs. For DeepWalk and LINE$_{2nd}$, there is a context embedding matrix computed together with the node embedding matrix. We use Y to denote the context embedding matrix.

Previous work (Qiu et al., 2018) has shown that DeepWalk and LINE$_{2nd}$ is implicitly doing matrix factorization.

- DeepWalk is solving the following matrix factorization problem:

$$\log\big(vol(G)\big(\frac{1}{T}\sum_{i=1}^{T}(D^{-1}A)^r\big)D^{-1}\big) - \log b = XY^T \tag{1}$$

- LINE$_{2nd}$ is solving the following matrix factorization problem:

$$\log\big(vol(G)D^{-1}AD^{-1}\big) - \log b = XY^T \tag{2}$$

where $vol(G) = \sum A_{ij}$ is the volume of graph G, D is the diagonal matrix where each element represents the degree of the corresponding node, T is the context window size and b is the number of negative samples. We use Z to denote the matrix that DeepWalk and LINE$_{2nd}$ is factorizing. We denote $\Omega = \{(i,j) : Z_{ij} \neq 0\}$ as the observable elements in Z when solving matrix factorization and $\Omega_i = \{j : Z_{ij} \neq 0\}$ as the nonzero elements in row i of Z. With these notations defined, we now give a unified formulation for DeepWalk and LINE$_{2nd}$:

$$\min_{X,Y} \|R_\Omega(Z - XY^T)\|_F^2 \tag{3}$$

where $[\mathcal{R}_\Omega(\mathbf{A})]_{ij}$ is $\mathbf{A}_{ij}$ if $(i,j) \in \Omega$ and 0 otherwise, $\|A\|_F^2$ denotes the squared Frobenious norm of matrix A.

4 Problem Statement

In this section we introduce the attack model, including attacker's action, attacker's utilities and the constraints on the attacker. We assume that the attacker can manipulate the poisoned graph G by adding or deleting edges. In this paper, we consider these two type of manipulation: adding edges and deleting edges respectively. We use G^{adv} to denote the poisoned graph.

We characterize two kinds of adversarial goals:

Integrity attack: Here the attacker's goal is either to increase or decrease the probability (similarity score) of a target node pair. For example, in social network, the attacker may be interested in increasing (or decreasing) the probability that a friendship occurs between two people. Also, in recommendation system, an attacker associated with the producer of a product may be interested to increase the probability of recommending the users with that specific product. Specifically, the attacker aims to change the probability of the edge connected with a pair of nodes whose embedding is learnt from the poisoned graph G^{adv}.

For integrity attack, we consider two kinds of constraints on the attacker: 1. *Direct Attack*: the attacker can only manipulate edges adjacent to the target node pair; 2. *Indirect Attack*: the attacker can only manipulate edges without connecting to the target node pair.

Availability attack Here the adversarial goal of availability attack is to reduce the prediction performance over a test set consisting of a set of node pairs $\mathcal{T} \in V \times V$. (Here $\mathcal{T}$ consists of both positive examples $\mathcal{T}_{pos}$ indicating the existence of edges, and negative examples $\mathcal{T}_{neg}$ for the absence of edges). In this paper, we choose average precision score (AP score) to evaluate the attack performance. Specifically, we consider the attacker whose goal is to decrease the AP score over $\mathcal{T}$ by adding small perturbation to a given graph.

5 ATTACKING UNSUPERVISED GRAPH EMBEDDING

In this section we show our algorithm for computing the adversarial strategy. Given that DeepWalk and LINE is implicitly doing matrix factorization, we can directly derive the back-propogated gradient based on the first order KKT condition. Our algorithm has two steps: 1. Projected Gradient Descent (PGD) step: gradient descent on the weighted adjacency matrix. 2. Projection step: projection of weighted adjacency matrix onto $\{0,1\}^{|V|\times|V|}$. We first describe the projected gradient descent step and then we describe the projection method we use to choose which edge to add or delete.

5.1 PROJECTED GRADIENT DESCENT (PGD)

Based on the matrix factorization formulation above, we describe the algorithm we use to generate the adversarial graph. The core part of our method is *projected gradient descent* (PGD) step. In this step, the adjacency matrix is continuous since we view the graph as a weighted graph, which allows us to use gradient descent.

First we describe the loss function we use. We use $L(X)$ to denote the loss function. For integrity attack, L is $\pm[XX^T]_{ij}$ where (i,j) is the target node pair. (Here the $+$ or $-$ sign depends on whether the attacker wants to increase or decrease the score of the target edge.) For availability attack, the loss function is $\sum_{(i,j)\in T_{pos}}[XX^T]_{ij} - \sum_{(i,j)\in T_{neg}}[XX^T]_{ij}$. The update of the weighted adjacency matrix A in iteration t is as follows:

$$A^{t+1} = \mathrm{Proj}_{\mathbb{A}}(A^t - s_t \cdot \nabla_A L) \tag{4}$$

Here Proj is the projection function which projects the matrix to $[0,1]$ space and s_t is the step size in iteration t. The non-trivial part is to compute $\nabla_A L$. We note that

$$\nabla_A L = \nabla_X L \cdot \nabla_A X \tag{5}$$

The computation of $\nabla_X L$ is trivial. Now to compute $\nabla_A X$, using the chain rule, we have: $\nabla_A X = \nabla_Z X \cdot \nabla_A Z$. First we show how to compute $\nabla_Z X$.

For the matrix factorization problem defined in Eq. 3, using the KKT condition, we have:

$$\sum_{j\in\Omega_i}(Z_{ij} - X_i Y_j^T)Y_j = 0 \tag{6}$$

$$\frac{\partial X_i}{\partial Z_{ij}} = (\sum_{j'\in\Omega_i} Y_{j'} Y_{j'}^T)^{-1} Y_j \tag{7}$$

Next we show how to compute $\nabla_A Z$ for DeepWalk and LINE separately. In the derivation of $\nabla_A Z$, we view $vol(G)$ and D as constant.

DeepWalk We show how to compute $\nabla_A Z$ for DeepWalk. For DeepWalk, From Eq. 1:

$$Z = \log\big(vol(G)\big(\frac{1}{T}\sum_{i=1}^{T}(D^{-1}A)^r\big)D^{-1}\big) - \log b \tag{8}$$

Now to compute $\nabla_A Z$, let $P = D^{-1}A$, then we only need to derive $\nabla_P P^r$ for each $r \in [1,T]$. Note that $(P^r)' = \sum_{k=1}^{r} P^{k-1}P'P^{r-k}$. Then since computing $\nabla_A P$ and $\nabla_P Z$ is easy, once we have $\nabla_P P^r$, we can compute $\nabla_A Z$.

LINE We show the derivation of $\nabla_A Z$ for LINE$_{2nd}$:

$$Z = \log(vol(G)D^{-1}AD^{-1}) - \log b \tag{9}$$

since $Z_{ij} = \log(vol(G)d_i^{-1}A_{ij}d_j^{-1}) - \log b$ where $d_i = D_{ii}$. We have:

$$\frac{\nabla Z_{ij}}{\nabla A_{ij}} = \frac{1}{A_{ij}} \tag{10}$$

Once we have $\nabla_Z L$ and $\nabla_L Z$, we can compute $\nabla_A X$.

5.2 PROJECTION

In the Projected Gradient Descent step, we compute a weighted adjacency matrix A^{opt}. We use A^{org} to denote the adjacency matrix of the clean graph. Therefore we need to project it back to $\{0,1\}^{|V|\times|V|}$. Now we describe the projection method we use, which is straightforward. First, we show our projection method for an attacker that can add edges. To add edges, the attacker needs to choose some cells in $S = \{(i,j) \mid A_{ij}^{org} = 0\}$ and turn it into 1. Our projection strategy is that the attacker chooses the cells (i,j) in S where A_{ij}^{opt} is closest to 1 as the candidate set of edges to add. For deleting edges, it works in a similar way. The only difference is that we start from the cells that are originally 1 in A^{org} and choose the cells that are closest to 0 in A^{opt} as the candidate set of edges to delete.

Now we briefly discuss when to use the projection step. A natural choice is to project once after the projected gradient descent step. Another choice is to incorporate the projection step into the gradient descent computation where we project every k iterations. The second projection strategy induces less loss in the projection step and is more accurate for computation but can take more iterations to converge than the first projection strategy. In our experiments, we choose the first projection strategy for its ease of computation.

6 EXPERIMENTS

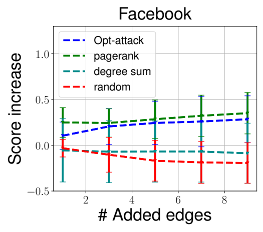
In this section, we show the results of poisoning attack against DeepWalk and LINE on real-world graph datasets. In the experiments, we denote our poisoning attack method as ‘Opt-attack’. We evaluate our attack method on three real-world graph datasets: 1). Facebook (Leskovec & Mcauley, 2012): a social networks with 4039 nodes and 88234 edges. 2). Cora (Sen et al., 2008): a citation network with 2708 nodes and 2708 edges. 3). Citeseer (Giles et al., 1998): a citation network with 2110 nodes and 7336 edges.

Baselines We compare with several baselines: (1) *random attack*: this baseline is general and used for both integrity attack and availability attack. We randomly add or remove edges; (2) *personalized PageRank* (Bahmani et al., 2010): this baseline is only used for integrity attack. Given a target edge (A,B), we use personalized PageRank to calculate the importance of the nodes. Given a list of nodes ranked by their importance, e.g., $(x_1, x_2, x_3, x_4, ...)$, we select the edges which connect the top ranked nodes to A or B, i.e., $(A, x_1), (B, x_1), (A, x_2), (B, x_2), ...$; (3) *degree sum*: We rank the node pair by the sum of degree of its two nodes. Then we add or delete the corresponding edges with the largest degree sum. This baseline is used for both availability and integrity attack. (4) *shortest path*: this baseline is used for availability attack. We rank the edge by the number of times that it is on the shortest paths between two nodes in graph. Then we delete the important edges measured by the number of shortest paths in graph that go through this edge.

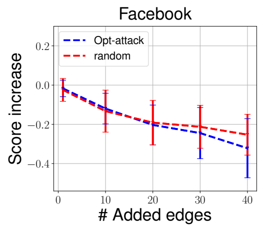
In our experiments, we choose 128 as the latent embedding dimension. We use the default parameter settings for DeepWalk and LINE. We generate the test set and validation set with a proportion of 2:1 where positive examples are sampled by removing 15% of the edges from the graph and negative examples are got by sampling an equal number of node pairs from the graph which has no edge connecting them. For both our attack and random attack, we guarantee that the attacker can’t directly modify any node pairs in the target set.

6.1 INTEGRITY ATTACK

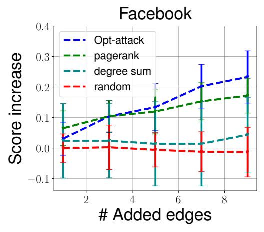
For each attack scenario, we choose 32 different target node pairs and use our algorithm to generate the adversarial graph. For increasing the score of the target edge, the target node pair is randomly sampled from the negative examples in the test set. For decreasing the score of the target edge, the target node pair is randomly sampled from the positive examples in the test set. We report the average score increase and compare it with the random attack baseline. We consider two kinds of attacker's actions: adding or deleting edges and two constraints: direct attack and indirect attack. Results for Citeseer dataset are deferred to the appendix.



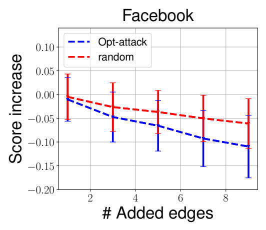
(a) DeepWalk-Add-Up



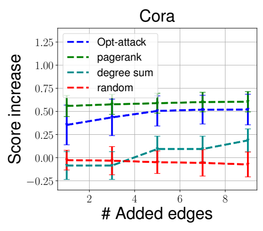
(b) DeepWalk-Add-Down



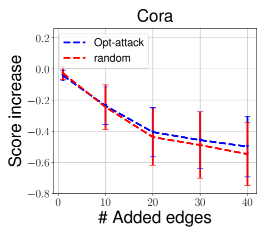
(c) LINE-Add-Up



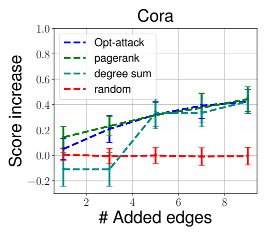
(d) LINE-Add-Down



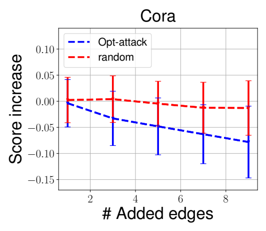
(e) DeepWalk-Add-Up



(f) DeepWalk-Add-Down



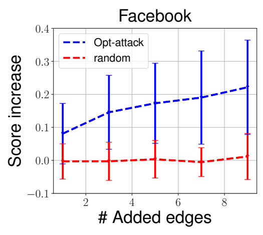
(g) LINE-Add-Up



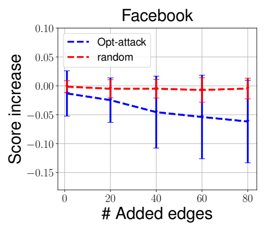
(h) LINE-Add-Down

Figure 1: Result for direct integrity attack on two datasets. The first line contains the results for Facebook dataset. The second line contains the results for Cora dataset. The format "Method —Type —Direction " is used to label the each sub-caption. "Method" refers to embedding method while "Type" refers to adding or deleting edges to poison the graph. "Direction" refers to increasing or decreasing the similarity score of the target node pair. This notation is also used in Figure 2, 3, 4.

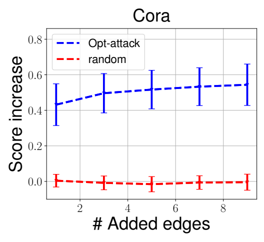
Adding edges We consider the adversary which can only add edges. Figure 1 shows the results under direct attack setting. In Figure 1a 1c 1e 1g, when the adversarial goal is to increase the score of the target node pair, we find that our attack method outperforms the random attack baseline by a significant margin. We also plot the personalized pagerank baseline. We can see this second baseline we propose is a very strong baseline, with attack performance the same level as our proposed method and outperforming the 'degree sum' baseline. To further understand it, we analyze the edges we add in this attack scenario. We find the following pattern: if the target node pair is (i, j), then our attack tends to add edges from node i to the neighbors of node j and also from node j to the neighbors of node i. This is intuitive because connecting to other node's neighbors can increase the similarity score of two nodes. In Figure 1d 1h, when the adversarial goal is to decrease the score of target node pair, our method is better than the random attack baseline for attacking LINE. For attacking DeepWalk (figure 1b 1f), the algorithm is able to outperform, on Facebook dataset (figure 1b), our attack is better than the random baseline when the number of added edges is large. Although for Cora (figure 1f) our attack is close to the random attack baseline, we note that in this attack case, random attack is already powerful and can lead to large drop (e.g. 0.8) in similarity score.



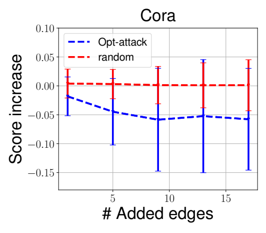
(a) DeepWalk-Add-Up



(b) DeepWalk-Add-Down



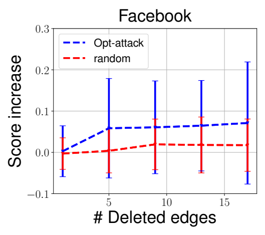
(c) DeepWalk-Add-Up



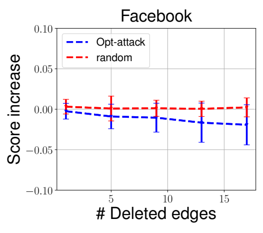
(d) DeepWalk-Add-Down

Figure 2: Result for indirect integrity attack against DeepWalk on two datasets where the action of attacker is adding edges.

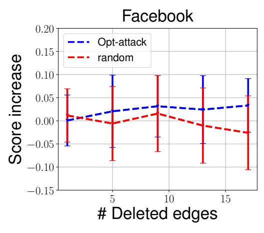
Figure 2 shows our result of indirect attack. We can see that for DeepWalk, even if the attacker can't modify the edges adjacent to the target node pair, it can still manipulate the score of the target edge with a few edges added. We also analyze the edges our algorithm chooses when the adversarial goal is to increase the score of the target node pair (i, j) (the case in figure 2a 2c), we find that our attack tends to add edges between the neighbors of node i and the neighbors of node j. It also follows the intuition that connecting the neighbor of two nodes can increase the similarity of two nodes. When the goal is to decrease the score (figure 2b 2d), our attack is still better than random baseline by a noticeable margin.



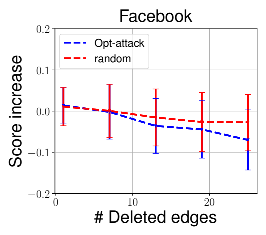
(a) DeepWalk-Del-Up



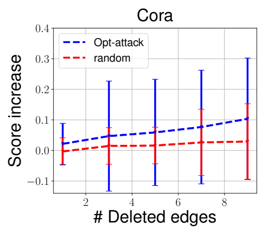
(b) DeepWalk-Del-Down



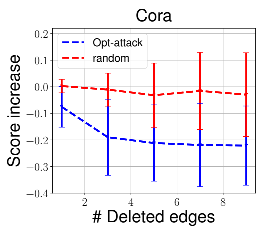
(c) LINE-Del-Up



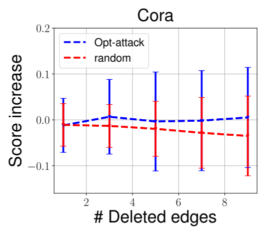
(d) LINE-Del-Down



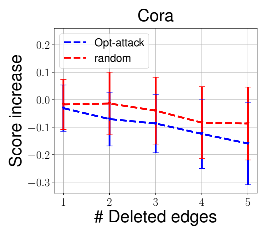
(e) DeepWalk-Del-Up



(f) DeepWalk-Del-Down

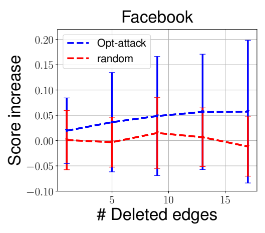


(g) LINE-Del-Up

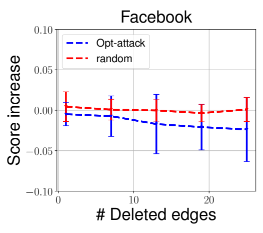


(h) LINE-Del-Down

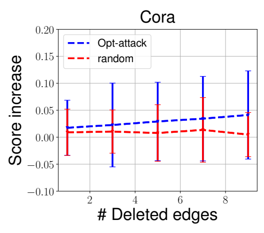
Figure 3: Result for direct integrity attack against two methods on two datasets where the action of the attack is deleting edges.



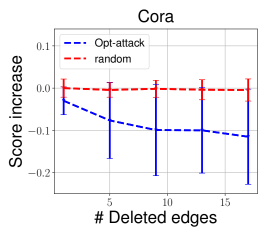
(a) DeepWalk-Del-Up



(b) DeepWalk-Del-Down



(c) DeepWalk-Del-Up



(d) DeepWalk-Del-Down

Figure 4: Result for indirect integrity attack against DeepWalk on two datasets where the action of the attacker is deleting edges.

Deleting edges Now we consider the adversary which can delete existing edges. Figure 3 summarizes our result for direct attack. We can see that our attack method works well for attacking DeepWalk (figure 3a 3b 3e 3f). The large variance may be because that different edges have different sensitivity to deleting edges. Also, we notice that LINE is more robust to deleting edges, with average magnitude of score increase(decrease) lower than DeepWalk. Figure 4 summarizes our results for indirect attack. Still, on average, our attack is able to outperform the random attack baseline.

6.2 AVAILABILITY ATTACK

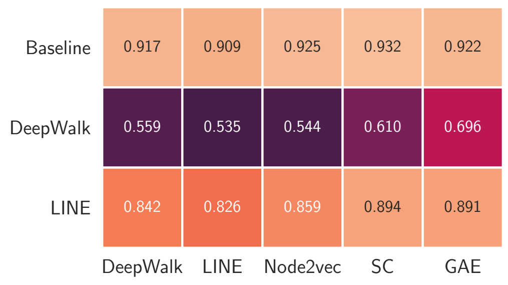
In this part, we show the results for availability attack. We report our results on two dataset: Cora and Citeseer. Results for Citeseer are deferred to the appendix. For both datasets, we choose the test set to be attack. Table 1 summarizes our result. We can see that our attack almost always outperforms the baselines. When adding edges, our optimization attack outperforms all other baselines by a significant margin. When deleting edges, we can see that LINE is more robust than DeepWalk. In general, we notice the that adding edges is more powerful than deleting edges in our attack.

Action	Model	Baseline	Attack Method	# added/deleted edges						
				25	50	100	150	200	250	300
Add	DeepWalk	0.917	random	0.922	0.923	0.920	0.919	0.923	0.922	0.922
			degree sum	0.915	0.913	0.909	0.904	0.906	0.906	0.904
			Opt-attack	**0.832**	**0.773**	**0.666**	**0.597**	**0.559**	**0.532**	**0.503**
	LINE	0.909	random	0.908	0.908	0.913	0.900	0.901	0.903	0.905
			degree sum	0.903	0.904	0.899	0.890	0.886	0.886	0.888
			Opt-attack	**0.898**	**0.892**	**0.886**	**0.846**	**0.826**	**0.803**	**0.766**
Delete	DeepWalk	0.917	random	0.916	0.917	0.921	0.918	0.920	0.914	0.913
			degree sum	0.917	0.918	0.919	0.920	0.922	0.918	0.916
			shortest path	0.916	0.919	0.917	0.918	0.922	0.921	0.924
			Opt-attack	**0.897**	**0.892**	**0.876**	**0.866**	**0.853**	**0.838**	**0.835**
	LINE	0.909	random	**0.901**	**0.899**	**0.892**	0.901	0.898	0.894	0.886
			degree sum	0.903	0.904	0.908	0.919	0.911	0.890	0.888
			shortest path	0.903	0.904	0.908	0.919	0.911	0.890	0.889
			Opt-attack	0.915	0.909	0.898	**0.890**	**0.876**	**0.859**	**0.861**

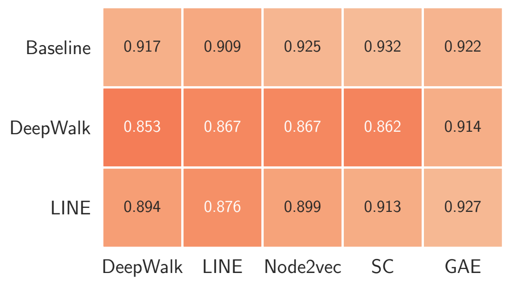
Table 1: Results for availability attack on Cora dataset. Here we report the AP score.

6.3 TRANSFERABILITY

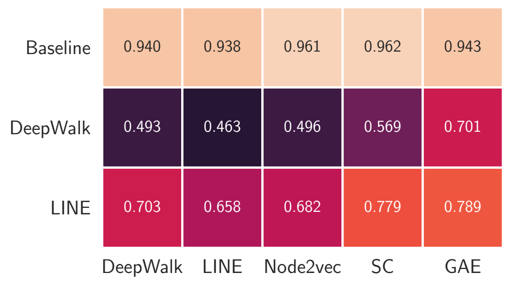
In this part, we show that our attack can be transferred across different embedding methods. Besides DeepWalk and LINE, we choose another three embedding methods to test the transferability of our approach: 1. Variational Graph Autoencoder(GAE) (Kipf & Welling, 2016); 2. Spectral Clustering (Tang & Liu, 2011); 3. Node2Vec (Grover & Leskovec, 2016). For GAE, we use the default setting as in the original paper. For Node2Vec, we first tune the parameters p, q on a validation set and use the best p, q for Node2Vec.



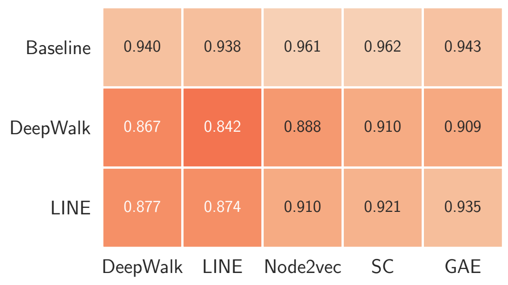
(a) Cora-Add



(b) Cora-Del



(c) Citeseer-Add



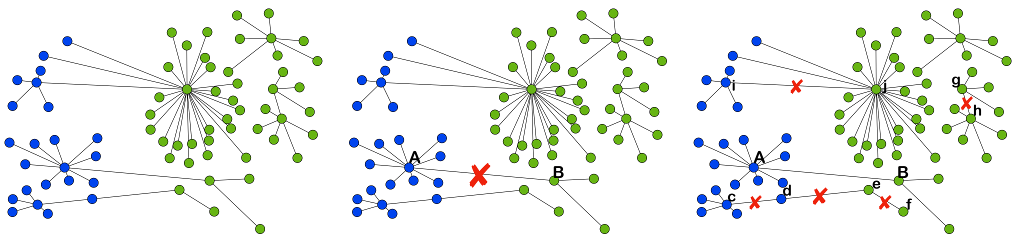
(d) Citeseer-Add

Figure 5: Result for transferability analysis of our attack on two datasets, where the number of added/deleted edges is 200. X-axis indicates the method the attack is evaluated on. Y-axis includes the methods to generate the attack and also the baseline. The format "Dataset — Type " here is used to label the each sub-caption. "Dataset" refers to dataset while "Type" refers to attacker's action.

Figure 5 shows our result for transferability test of our attack, where the number of added(deleted) edges is 200. Results when the number of added(deleted) edges is 100 and 300 are deferred to the appendix. Comparing Figure 5a 5c and Figure 5b 5d, we can see that adding edges is more effective than deleting edges in our attack. The attack on DeepWalk has higher transferability compared with other four methods (including LINE$_{2nd}$). Comparing the decrease of AP score for all five methods, we can see that GAE is more robust against transferability based attacks.

7 Case Study: Attack DeepWalk on Coauthor Network

In this section, we conduct a case study on a real-world coauthor network extracted from DBLP (Tang et al., 2008). We construct a coauthor network from two different research communities: machine learning & data mining (ML&DM), and security (Security). For each research community, we select some conferences in each field: ICML, ICLR, NIPS, KDD, WWW, ICWD, ICDM from ML&DM and IEEE S&P, CCS, Usenix and NDSS from Security. We sort the authors according to the number of published papers and keep the top-500 authors that publish most papers in each community, which eventually yields a coauthor network with 1,000 nodes in total. The constructed coauthor graph contains 27260 in-field edges and 1014 cross-field edges. We analyze both integrity attack and availability attack on this coauthor network.



(a) Original Network (b) Adversarial Goal (c) Attack Example: Victims

Figure 6: Visualization for integrity attack. Green nodes denote authors from ML&DM community and blue nodes denote authors from Security community. Nodes and their corresponding researchers: A: John C. Mitchell; B: Trevor Hastie; c: Susan Landau; d: Michael Lesk; e: Hector Garcia-Molina; f: Jure Leskovec; g: David D. Jensen; h: Thomas G. Dietterich; i: Matthew Fredrikson; j: Jiawei Han.

Integrity Attack We consider an indirect attack setting where the adversarial goal is to decrease the score of a target node pair and the attacker's action is deleting existing edges. We show the subgraph that contains the nodes in the target edge and 5 edges chosen by our algorithm, as well as the nodes that coauthored more than 3 papers with them, e.g. frequent collaborators. We visualize the original graph in Figure 6a. Green nodes represent ML&DM authors and blue nodes denote authors from Security community.

In Figure 6b, we show the target node pair A and B. Node A denotes John C. Mitchell, a professor in Stanford University from the security community and node B denotes Trevor Hastie, also a Stanford professor, from the ML & DM community. After the attack, the similarity score of the target node pair is reduced from 0.67 to 0.37. We make the following observations: 1) We find that the top 2 edges (d, e) and (e,f) chosen by our attack lie on the shortest path between A and B, which corresponds to the intuitive understanding that cutting the paths connecting A with B makes it less likely to predict that an edge exists between A and B; 2) We find that many of the edges chosen by our algorithm are cross-field edges (figure 6c): edge (i, j) and edge (d, e). Considering how small a proportion the cross-field edges consist of(3.6% of all edges), we hypothesize that cutting the cross-field edges could impede the information flow between two communities and therefore making the similarity score of cross-field link lower.

Availability Attack For availability attack, we analyze the adversary that add edges. It turns out that our algorithm tends to add cross-field edges: for the top 40 added edges chosen by our attack method, 39 are cross-field edges. We hypothesize that this is because adding more edges between two communities can disrupt the existing information flow and therefore lead the learnt embedding to carry incorrect information about the network structure.

8 Conclusion

In this paper, we investigate data poisoning attack against unsupervised node embedding methods and take the task of link prediction as an example. We study two types of data poisoning attacks including integrity attack and availability attack. We propose a unified optimization framework to optimally attack the node embedding methods for both types of attacks. Experimental results on several real-world graphs show that our proposed approach can effectively attack the results of link prediction by adding or removing a few edges. Results also show that the adversarial examples discovered by our proposed approach are transferable across different node embedding methods. Finally, we conduct a case study analysis to better understand our attack method. In the future, we plan to study how to design effective defense strategies for node embedding methods.

References

Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. In *VLDB*, 2010.

Aleksandar Bojchevski and Stephan Gnnemann. Adversarial attacks on node embeddings. *arXiv preprint arXiv:1809.01093*, 2018.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy, 2017*, 2017.

Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128*, 2018.

Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling deep structured prediction models. *arXiv preprint arXiv:1707.05373*, 2017.

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

C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: an automatic citation indexing system. In *The Third ACM Conference on Digital Libraries*, 1998.

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

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

Thomas Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Jure Leskovec and Julian J. Mcauley. Learning to discover social circles in ego networks. In *NIPS*. 2012.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *NIPS*, 2016.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *NIPS*, 2013.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, 2016.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization. In *WSDM*, 2018.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

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

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *WWW*, 2016.

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, 2008.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, November 2011.

Chaowei Xiao, Ruizhi Deng, Bo Li, Fisher Yu, Dawn Song, et al. Characterizing adversarial examples based on spatial consistency information for semantic segmentation. In *ECCV*, 2018a.

Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *IJCAI*, 2018b.

Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *ICLR*, 2018c.

Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *ICCV*, 2017.

Dawei Yang, Chaowei Xiao, Bo Li, Jia Deng, and Mingyan Liu. Realistic adversarial examples in 3d meshes. *arXiv preprint arXiv:1810.05206*, 2018.

Daniel Zugner, Amir Akbarnejad, and Stephan Gunnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.

APPENDIX

A IMPLEMENTATION DETAIL

In this part, we discuss the initialization of weighted adjacency matrix in the projected gradient descent step. From the formulation in section 5.1, if we initialize all cells which are initially 0 to 0. Then there won't be back-propagated gradient on these cells. (This is because Ω won't contain these cells.) To handle this issue, we initialize these cells with a small value, for example 0.001, which allows the gradient on these cells to be efficiently computed.

B RANDOM GRAPH EXPERIMENTS

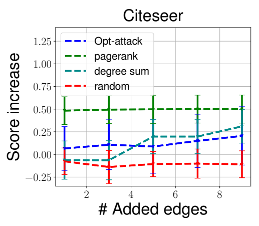
We experiment with three random graphs: Erdos-Renyi graph, Barabasi-Albert graph and Watts-Strogatz graph. We can see that from random graphs, our attack method still outperformas the random attack baseline.

Action	Model	Random Graph	Baseline	Attack Method	# added/deleted edges				
					25	50	100	250	500
Add	DeepWalk	Erdos-Renyi	0.492	random	0.485	0.483	0.492	0.491	0.494
				Opt-attack	**0.483**	0.483	**0.471**	**0.448**	**0.415**
		Barabasi-Albert	0.436	random	0.415	0.406	0.406	0.417	0.411
				Opt-attack	**0.410**	**0.404**	**0.404**	**0.378**	**0.343**
		Watts-Strogatz	0.964	random	0.945	0.945	0.945	0.944	0.946
				Opt-attack	**0.944**	**0.942**	**0.940**	**0.937**	**0.933**

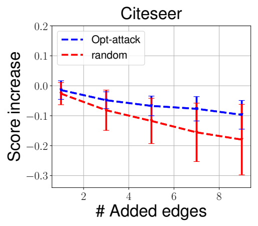
Table 2: Results for attack on three random graphs: Erdos-Renyi graph, Barabasi-Albert graph and Watts-Strogatz graph. Here we report the AP score.

C ADDITIONAL EXPERIMENTAL RESULTS

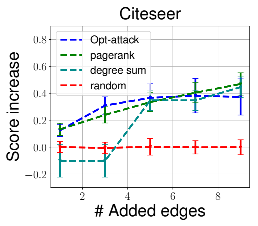
C.1 INTEGRITY ATTACK



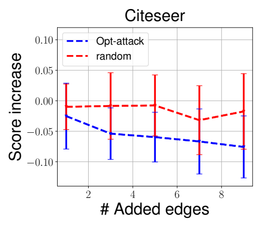
(a) DeepWalk-Add-Up



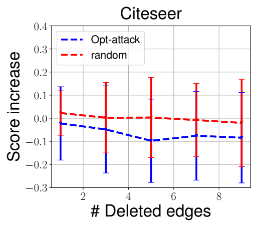
(b) DeepWalk-Add-Down



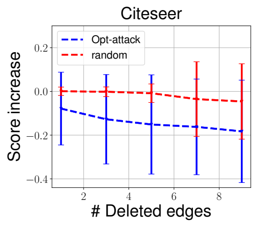
(c) LINE-Add-Up



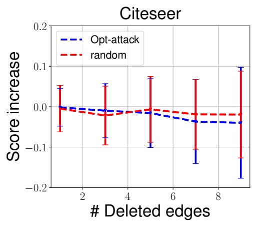
(d) LINE-Add-Down



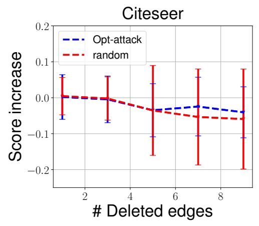
(e) DeepWalk-Del-Up



(f) DeepWalk-Del-Down



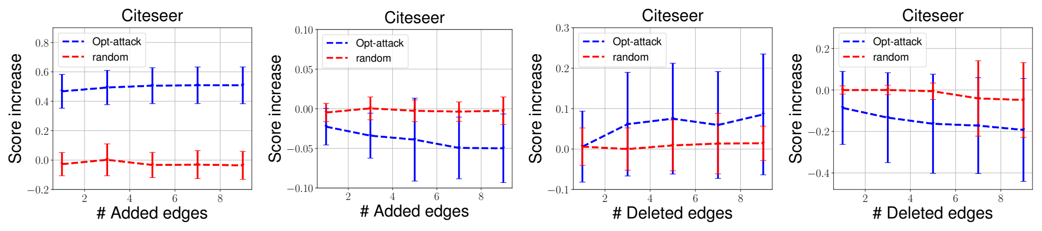
(g) LINE-Del-Up



(h) LINE-Del-Down

Figure 7: Result for direct integrity attack against DeepWalk and LINE on Citeseer dataset.

Here we show additional experimental results. Figure 7, 8 summarize the results of integrity attack on Citeseer dataset. Figure 7 shows our results for direct integrity attack and Figure 8 shows our results for indirect integrity attack.



(a) DeepWalk-Add-Up (b) DeepWalk-Add-Down (c) DeepWalk-Del-Up (d) DeepWalk-Del-Down

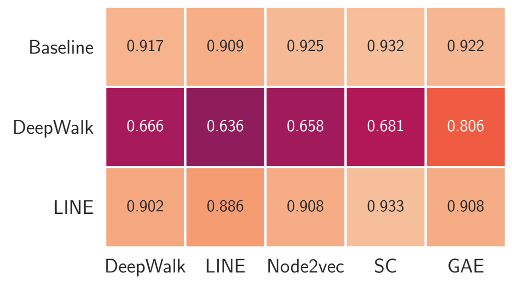
Figure 8: Result for indirect integrity attack against DeepWalk on Citeseer dataset.

D AVAILABILITY ATTACK

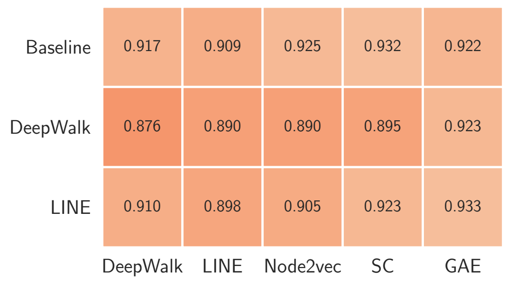
Here we show additional results for availability attack. Table 3 shows the results of availability attack on Citeseer dataset. Figure 9, 10 show additional results of transferability analysis.

Action	Model	Baseline	Attack Method	# added/deleted edges						
				25	50	100	150	200	250	300
Add	DeepWalk	0.940	random	0.940	0.936	0.935	0.940	0.937	0.941	0.942
			degree sum	0.919	0.906	0.886	0.865	0.864	0.843	0.828
			Opt-attack	**0.810**	**0.706**	**0.601**	**0.536**	**0.493**	**0.461**	**0.433**
	LINE	0.938	random	0.931	0.934	0.925	0.925	0.923	0.926	0.936
			degree sum	0.929	0.922	0.925	0.921	0.920	0.924	0.924
			Opt-attack	**0.912**	**0.863**	**0.788**	**0.725**	**0.658**	**0.621**	**0.570**
Delete	DeepWalk	0.940	random	0.936	0.927	0.925	0.926	0.925	0.919	0.918
			degree sum	0.932	0.936	0.927	0.924	0.921	0.919	0.922
			shortest path	0.943	0.925	0.916	0.907	0.902	0.900	0.895
			Opt-attack	**0.923**	**0.914**	**0.897**	**0.884**	**0.867**	**0.860**	**0.832**
	LINE	0.938	random	0.923	**0.919**	0.915	0.898	0.900	0.887	0.893
			degree sum	0.924	0.926	0.906	**0.889**	0.902	0.895	0.888
			shortest path	**0.907**	**0.919**	0.899	0.915	0.884	0.884	0.891
			Opt-attack	0.925	0.930	**0.896**	0.897	**0.874**	**0.875**	**0.880**

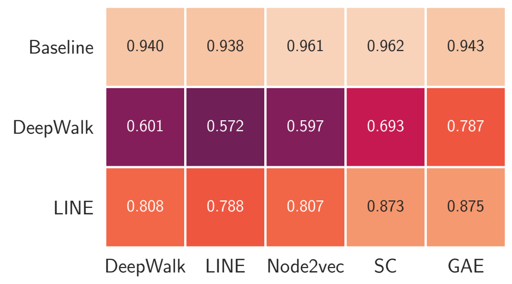
Table 3: Results for availability attack on Citeseer dataset. Here we report the AP score.



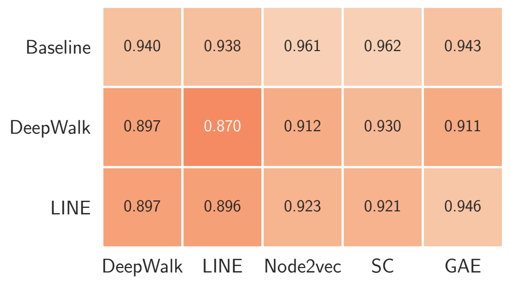
(a) Cora-Add



(b) Cora-Del

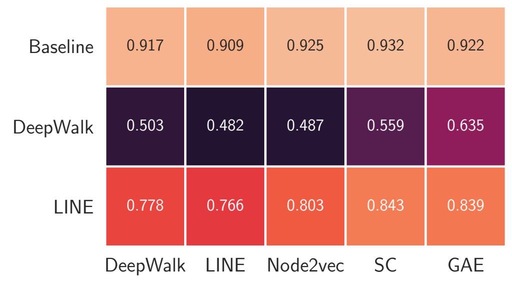


(c) Citeseer-Add

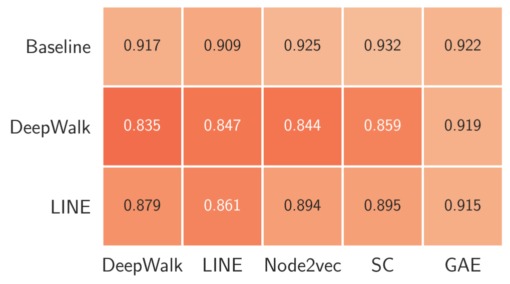


(d) Citeseer-Add

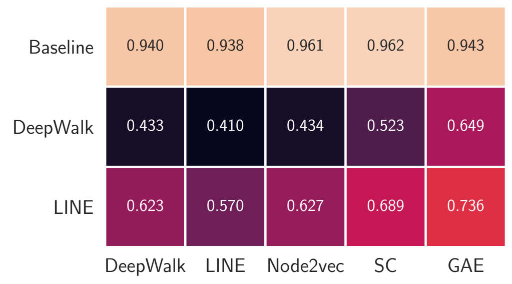
Figure 9: Result for transferability analysis of our attack on two datasets, where the number of added/deleted edges is 100. X-axis indicates the method the attack is evaluated on. Y-axis includes the methods to generate the attack and also the baseline. The format "Dataset — Type " here is used to label the each sub-caption. "Dataset" refers to dataset while "Type" refers to attacker's action.



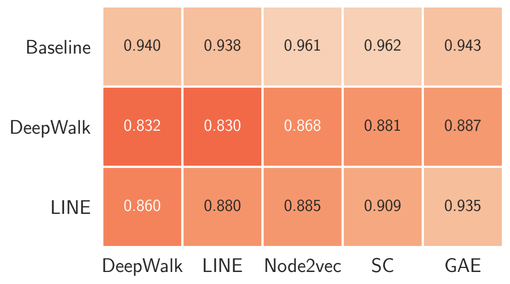
(a) Cora-Add



(b) Cora-Del



(c) Citeseer-Add



(d) Citeseer-Add

Figure 10: Result for transferability analysis of our attack on two datasets, where the number of added/deleted edges is 300. X-axis indicates the method the attack is evaluated on. Y-axis includes the methods to generate the attack and also the baseline. The format "Dataset — Type " here is used to label the each sub-caption. "Dataset" refers to dataset while "Type" refers to attacker's action.