Time-Aware Gradient Attack on Dynamic Network Link Prediction

Jinyin Chen, Jian Zhang, Zhi Chen, Min Du, and Qi Xuan, *Senior Member, IEEE*

Abstract—In network link prediction, it is possible to hide a target link from being predicted with a small perturbation on network structure. This observation may be exploited in many real world scenarios, for example, to preserve privacy, or to exploit financial security. There have been many recent studies to generate adversarial examples to mislead deep learning models on graph data. However, none of the previous work has considered the dynamic nature of real-world systems. In this work, we present the first study of adversarial attack on *dynamic network link prediction* (DNLP). The proposed attack method, namely *time-aware gradient attack* (TGA), utilizes the gradient information generated by deep dynamic network embedding (DDNE) across different snapshots to rewire a few links, so as to make DDNE fail to predict target links. We implement TGA in two ways: One is based on traversal search, namely TGA-Tra; and the other is simplified with greedy search for efficiency, namely TGA-Gre. We conduct comprehensive experiments which show the outstanding performance of TGA in attacking DNLP algorithms.

Index Terms—Dynamic network, link prediction, adversarial attack, transaction network, blockchain, deep learning

1 Introduction

In the era of big data, network analysis emerges as a powerful tool in various areas, such as recommendation on e-commerce websites [1], bug detection in software engineering [2], and behavioral analysis in sociology [3]. Existing algorithms like traditional similarity indices and newly developed network embedding methods learn structural features on static networks while most of them ignore the dynamic nature of real-world systems. The dynamics of networks, such as the establishment and dissolution of friendship in online social networks, could help the analysis. For example, in citation networks, the citations in different time which reflect the research focus of the authors should weigh differently in the prediction of future citations. And dynamic networks are naturally proposed for these occasions.

Dynamic network analysis, especially link prediction (DNLP) which we discuss in the paper, mainly focuses on the linkages status. That is predicting whether a link exists or not at a certain time based on historical information. For instance, given a social network, DNLP algorithms predict a given individual's relationships with others in the near future based on his/her historical friend lists. Comparing with traditional link prediction algorithms, DNLP algorithms could learn the underlying transitions of behavior patterns while the traditional ones could not. DNLP is generally performed in two ways: Time-series data analysis and machine learning-based methods. A representative category of the former one is Autoregressive Integrated Moving Average (ARIMA) [4], [5] related methods which treat time varying similarity scores as time series to predict future links. The latter one includes factorization-based approaches and deep learning models of which the majority consists of Recurrent Neural Networks (RNNs).

Regardless of their performance, DNLP algorithms may suffer from adversarial attacks as most machine learning methods do. The exploitation of adversarial examples may expose DNLP algorithms to security threats. However, from another perspective, this property could be useful in other fields like privacy-preserving. Privacy issues have aroused wide concern since the data theft of Facebook in 2018. In fact, one can infer private information, such as romantic relationship or visiting the same restaurant [6], [7], of target individuals with the help of advanced algorithms like DNLP. A well designed adversarial example may protect intimate relationships from being predicted by even the most advanced DNLP approach, which could provide a possible solution to privacy protection. The target link could be hidden by linking the user to someone unfamiliar, or removing intimate links in historical interactions. A bunch of works have explored the ways of generating adversarial examples for graphs in recent years. Adversarial attack could be classified into three categories:

- *White-box Attack*: The attacker knows everything of the model, including training data and the model parameters.
- *Gray-box Attack*: The Attacker has limited knowledge of the model. Usually, the attacker trains a surrogate model to approximate the information obtained

- *Jinyin Chen, Jian Zhang, and Qi Xuan are with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou, Zhejiang 310023, China. E-mail: {chenjinyin, jianzh, xuanqi}@zjut.edu.cn.*
- *Zhi Chen is with the Department of Computer Science, University of Illinois Urbana-Chamoaign, Urbana, IL 61801 USA. E-mail: zhic4@illinois.edu.*
- *Min Du is with the Palo Alto Networks, Santa Clara, CA 95054 USA. E-mail: min.du.email@gmail.com.*

Manuscript received 19 Mar. 2020; revised 16 Aug. 2021; accepted 21 Aug. 2021. Date of publication 8 Sept. 2021; date of current version 10 Jan. 2023.
(Corresponding author: Qi Xuan.)
Recommended for acceptance by Y. Zhang.
Digital Object Identifier no. 10.1109/TKDE.2021.3110580

from the victim model. And then adversarial examples are generated according to the surrogate model.

- *Black-box Attack*: The attacker could acquire nothing of the model. The attacker could only feed the model with input data and obtain the output scores or labels.

In this paper, we mainly focus on white-box attack to investigate the vulnerability of DNLP models. Though white-box attacks on image and graph both use gradient information to guide the generation of adversarial examples, we could only flip the linkage status if we want to change the graph structure while there is no such limitation of the attack on images. Most existing researches [8] focus on generating adversarial graphs to fool Graph Convolutional Network (GCN) [9] and achieve considerable attack performance. However, few of them involve dynamic networks, not even DNLP.

In this paper, we propose a novel adversarial attack targeting DNLP, which we refer as *Time-aware Gradient Attack* (TGA), to hide target links from being predicted. Benefitting from the gradients generated by deep learning model, i.e., DDNE, TGA is able to find candidate links to be modified without extensive search, and perform attack at minimum cost. Considering the dynamics of networks, TGA compares the gradients on different snapshots separately rather than does simple sorting on all snapshots; furthermore, it searches candidate links across iterations to make full use of the gradients. Overall, our main contributions are summarized as follows.

- We design TGA to generate adversarial examples based on the gradients obtained by DDNE. As far as we know, it is the first work about adversarial attacks on DNLP.
- We conduct extensive experiments on six real-world dynamic networks and compare TGA with several baselines. The results show that TGA achieves the state-of-the-art attack performance. And a case study on Ethereum transaction network is carried out to validate the practicability.
- We vary DNLP model parameters and observe several interesting phenomena which could be inspiring to future research. For example, long-term prediction is more vulnerable to adversarial attacks; while integrating more historical information can increase the robustness of DDNE.

For the rest of this paper, we first review related works in Section 2 and preliminaries in Section 3. Then, we present the proposed attack details of TGA-Tra and TGA-Gre in Section 4, and gives the results of the proposed attack methods as well as the performance under some certain circumstances in Section 5. Finally, we conclude the paper with the prospect of future work in Section 6.

2 Related Work

This section briefly reviews the literature of DNLP algorithms and the related work on adversarial attacks.

DNLP Algorithms. Recently, a temporal restricted Boltzmann machine (TRBM) is adopted with additional neighborhood information, namely ctRBM [10], to learn the dynamics as well as the structural characteristics of networks. As an extension of ctRBM, GTRBM [11] combines TRBM and boosting decision tree to model the evolving pattern of each node. RBM deals with temporal networks in a 2-layer neural network and maybe not able to handle the high dimensional structural features. Besides the RBM-based methods, recurrent neural networks (RNN), like long short-term memory (LSTM), plays an important role in other DNLP algorithms. A stacked LSTM module is applied inside the autoencoder framework to capture time dependencies of the whole network [12], and a gated recurrent network (GRU) is used as the encoder which could relatively lower the computational complexity [13]. Goyal *et al.* [14] propose dyngraph2vec and its variations which combine multiple LSTM cells in an auto-encoder framework. To better learn graph structures, GCN-GAN [15] incorporates GCN with LSTM and the generative adversarial network (GAN) to generate temporal links. Above approaches are mostly based on deep learning models, limiting them to the dynamic networks with fixed scale. To address the problem, Wu *et al.* [16] propose a similarity index for node pairs based on node ranking which is able to deal with the dynamic networks with growing size of nodes. Instead of dividing a network into snapshots, CTDNE [17] performs future link prediction on continuous dynamic network by temporal random walk.

Adversarial Attacks. A bunch of works have explored the field of adversarial attack on graph data. Community membership anonymization is realized by connecting the target user to the one of high centrality [18]. Another method focuses on disconnecting certain neighbors while adding links between different communities, with regards to the centrality of degree, closeness and betweenness [19]. In fact, community deception can be achieved by only rewiring the links inner the target community [3]. On the other hand, the emerging network embedding techniques, such as the graph convolutional network (GCN) [9], have drawn wide attention these days. And Nettack [20], [21] is proposed to generate adversarial examples with respect to graph structure and node feature to fool the GCN model. Another gradient-based method called fast gradient attack (FGA) [22] makes full use of the gradients information to choose candidate links that need modification when performing attack. The above methods are called evasion attacks [20] which aim to generate adversarial examples to deceive target models without changing models' parameters. Zügner *et al.* [23] apply meta-gradient to achieve training-time attack which lowers the global accuracy in training process. Not limited to the manipulation on links, adding fake nodes [24] could also minimize the classification accuracy of GCN. To increase the scalability of attack methods, SGA [25] carries out adversarial attack on a small subgraph consisting of k-hop neighbors of the target node rather than the whole network. Though most existing adversarial attacks target at GCN, some unsupervised embedding methods are also concerned, e.g., Bojchevski *et al.* [26] analysis the adversarial vulnerability on random walk-based embedding methods and Sun *et al.* [27] propose to use Projected Gradient Descent (PGD) based attack lower the accuracy of DeepWalk [28] and LINE [29] on link prediction task.

Graph adversarial examples are designed not only for fooling GNNs but also for improving the robustness of the models through adversarial training. GraphAT [30] trains a GCN by additionally minimizing a graph adversarial regularizer to improve the performance and robustness. It adds perturbations to node features along with the training process. In addition, Jin *et al.* [31] propose to add noise to the latent node representations to achieve the same goal. Like the adversarial training methods for image classification models [32], [33], those focusing on GNNs could only generate continuous but not discrete perturbations and thus could not modify the network structure.

As innovative as they are, existing adversarial attack approaches are still limited to static networks while the networks in real world are always time-evolving.

3 Preliminary and Problem Formulation

In this section, we present the definition of DNLP, as well as the adversarial attacks on it.

3.1 Dynamic Network Link Prediction

A network structure could be represented by $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, where $\mathbf{V} = \{v_1, v_2, \ldots, v_n\}$ denotes the set of network nodes, and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ represents the set of links. A directed link from v_i pointing to v_j is denoted by an ordered pair of nodes (v_i, v_j). In this paper, we focus on the dynamic networks with fixed node set but temporal links. Such a dynamic network could be modeled as a sequence of graphs $\{\mathbf{G}_{t-N}, \ldots, \mathbf{G}_{t-1}\}$, where $\mathbf{G}_k = \{\mathbf{V}, \mathbf{E}_k\}$ represents the network's structure at the k^{th} interval. With this, the definition of DNLP goes as follows.

- *Given a sequence of N graphs* $\mathbf{S} = \{\mathbf{A}_{t-N}, \ldots, \mathbf{A}_{t-1}\}$, *where* $\mathbf{A}_k$ *denotes the adjacent matrix of* G_k, *the task of dynamic network link prediction is to learn a mapping* $\mathbf{S} \rightarrow \mathbf{A}_t$, *from historical snapshots to future network structure.*

Specifically, DNLP algorithms capture latent spatial features and temporal patterns from historical information, i.e., previous N adjacency matrices, and then are able to infer the adjacency matrix of next snapshot. A link exists between v_i and v_j at time t if the probability $P(\mathbf{A}_t(i,j))$ given by the DNLP algorithm is larger than some threshold.

3.2 Adversarial Attack on DNLP

The idea of the adversarial attack has been extensively explored in computer vision, which is typically achieved by adding unnoticeable perturbation to images in order to mislead classifiers. Similarly, adversarial network attack on DNLP generates adversarial examples by adding or deleting a limited number of links from the original network, so as to make DNLP algorithms fail to predict target linkages. Intuitively, the goal of the generated adversarial example is to minimize the probability of the target link predicted by the DNLP algorithms, which could be formalized as

$$\begin{aligned} &\min P(\mathbf{A}_t(i,j)|\hat{\mathbf{S}}) \\ &\text{subject to} \quad \hat{\mathbf{S}} = \mathbf{S} + \triangle\mathbf{S}, \end{aligned} \tag{1}$$

where $\hat{\mathbf{S}}$ denotes the generated adversarial example, and $\triangle\mathbf{S}$ is the perturbation introduced into $\mathbf{S}$, i.e., the amount of links that need modification. Usually, $\triangle\mathbf{S}$ is small enough to make the attack unnoticeable.

Different from the attack strategies on static network algorithms, the chosen links added to or deleted from historical snapshots are associated with temporal information. One same link on different snapshots may contribute differently in the prediction of target link, not to mention that the linkages are time-varying. Therefore, it is crucial to take *time* into consideration when designing the attacks.

4 Time-Aware Gradient Attack

In order to generate adversarial dynamic network with optimal link modification scheme, a naive idea is to search through permutation and combination, which however is extremely time-consuming. Fortunately, deep learning based DNLP methods produce abundant information when making predictions, i.e., the gradients, which may assist adversarial example generation. It assumes that the attacker could access every detail of DDNE, including the model structure and the parameters, indicating that TGA is a white-box attack method. Here, we first briefly introduce DDNE and then show how it can help to generate adversarial examples. The framework is shown in Fig. 1. It should be noted that it does not matter which DNLP model is adopted here, as long as it can achieve a reasonable performance.

4.1 The Framework of DDNE

DDNE [13] has a dual encoder-decoder structure. A GRU could be used as the encoder, which reads the input node sequence both forward and backward, and turns the node into lower representation. The decoder, which consists of several fully connected layers, restores the input node from the extracted features. The original DDNE directly encodes each snapshot with a GRU, making it unsuitable for processing large-scale network. To make DDNE adapt to larger networks, we first encode the network with an embedding layer which significantly reduces the number of parameters of DDNE. For a node v_i, the encoding process is described as

$$\begin{aligned} &\mathbf{S} = Embedding(\mathbf{S}), \\ &\overrightarrow{\mathbf{h}}_i^k = \mathrm{GRU}(\overrightarrow{\mathbf{h}}_i^{k-1} + \overrightarrow{\mathbf{S}}^k(i:,)), \\ &\overleftarrow{\mathbf{h}}_i^k = \mathrm{GRU}(\overleftarrow{\mathbf{h}}_i^{k-1} + \overleftarrow{\mathbf{S}}^k(i:,)), \\ &\mathbf{h}_i^k = [\overrightarrow{\mathbf{h}}_i^k, \overleftarrow{\mathbf{h}}_i^k], \qquad k = \{t-N, \ldots, t-1\}, \\ &\mathbf{c_i} = [\mathbf{h}_i^{t-N}, \ldots, \mathbf{h}_i^{t-1}], \end{aligned} \tag{2}$$

where $\mathbf{h}_i^k$ represents the hidden state of the GRU when processing v_i of the k^{th} snapshot, and $\mathbf{c}_i$ is the concatenation of all $\mathbf{h}_i^k$ in time order. $\mathbf{h}_i^k$ consists of two parts, the forward one $\overrightarrow{\mathbf{h}}_i^k$ and the reversed one $\overleftarrow{\mathbf{h}}_i^k$, which are fed with opposite time sequence, $\overrightarrow{\mathbf{S}}(i,:)) = \{\mathbf{A_{t-N}}(i,:), \ldots, \mathbf{A_{t-1}}(i,:)\}$ and $\overleftarrow{\mathbf{S}}(i:,)) = \{\mathbf{A_{t-1}}(i,:), \ldots, \mathbf{A_{t-N}}(i,:)\}$, respectively. The decoder is composed of multilayer perceptrons, of which the complexity may vary according to the scale of datasets. The decoding process could be formulated as

$$\begin{aligned} &\mathbf{y}_i^{(1)} = \sigma_1(\mathbf{W}^{(1)}\mathbf{c}_i + \mathbf{b}^{(1)}), \\ &\mathbf{y}_i^{(m)} = \sigma_m(\mathbf{W}^{(m-1)}\mathbf{y}_i^{(m-1)} + \mathbf{b}^{(m)}), m = 2, \ldots, M, \end{aligned} \tag{3}$$

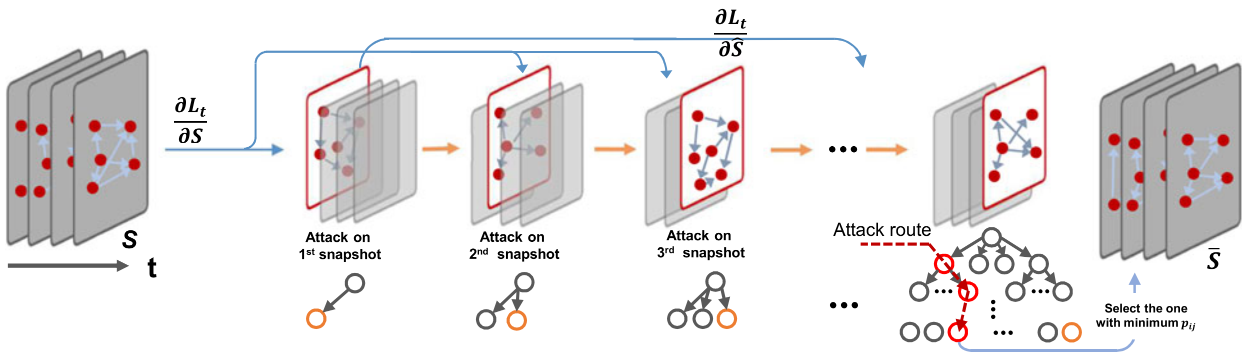

$\frac{\partial L_t}{\partial S}$
$\frac{\partial L_t}{\partial S}$
S
t
Attack on 1st snapshot
Attack on 2nd snapshot
Attack on 3rd snapshot
Attack route
Select the one with minimum p_{ij}
$\bar{S}$

Fig. 1. The framework of TGA-based methods. The nodes in the trees below represent the generation of adversarial example sets and each one in orange corresponds to the set generated in the current iteration.

where M represents the number of layers in the decoder, and σ_m denotes the activation function applied in the m^{th} decoder layer. Here, $\sigma_m = \mathrm{ReLU}(\cdot)$ when $m < M$ and $\sigma_M = \mathrm{sigmoid}(\cdot)$. In the training process, DDNE minimizes objective function $\mathcal{L}_{all}$ which consists of three parts: An adjusted L_2 loss $\mathcal{L}_s$ between predicted snapshot and the true one to learn the transition pattern, an adjusted L_2 loss $\mathcal{L}_c$ between the two embeddings to capture interaction proximity and a regularization term $\mathcal{L}_{reg}$ to avoid overfitting. And $\mathcal{L}_{all}$ is defined as

$$\mathcal{L}_{all} = \mathcal{L}_s + \beta\mathcal{L}_c + \gamma\mathcal{L}_{reg}. \tag{4}$$

Here, $\mathcal{L}_s$ adds an additional weight $\mathbf{Z}(i,:)$ to L_2 loss in order to ease the impact of sparsity, with $\{\mathbf{Z}(i,j)\}_{j=1}^{n} = 1$ if $\mathbf{S}^t(i,j) = 0$ and $\{\mathbf{Z}(i,j)\}_{j=1}^{n} = \alpha > 1$ otherwise. $\mathcal{L}_s$ is defined as

$$\begin{aligned}\mathcal{L}_s &= -\sum_{i=1}^{n} \mathbf{Z}(i,:) \odot [\mathbf{A}_t(i,:) - \hat{\mathbf{A}}_t(i,:)]^2 \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{Z}(i,j)[\mathbf{A}_t(i,j) - \hat{\mathbf{A}}_t(i,j)]^2.\end{aligned} \tag{5}$$

On the other hand, $\mathcal{L}_c$ imposes N_{ij}, the amount of links between v_i and v_j in historical snapshots, to L_2 loss. It addresses the influence of historical connections, and is defined as

$$\mathcal{L}_c = \sum_{u,v=1}^{n} N_{ij} \parallel \mathbf{c}_i - \mathbf{c}_j \parallel_2. \tag{6}$$

4.2 Time-Aware Link Gradient

When training DDNE, we minimize $\mathcal{L}_{all}$ through gradient descent. And in general, the lower $\mathcal{L}_{all}$ is, the better performance that DDNE has. After the training process of DDNE finishes, $\mathcal{L}_{all}$ is dependent on $\mathbf{S}$ as the parameters of the model are fixed. In adversarial network generation, we can update $\mathbf{S}(i,:)$ by taking $\partial\mathcal{L}_{all}/\partial\mathbf{S}(i,:)$ with $\mathbf{S}(i,:)$ being the variable. $\mathcal{L}_{all}$ integrates the information of the entire network, which makes the links that contribute the most in prediction covered among all links in $\mathbf{S}(i,:)$. To find out the most valuable links in target link prediction, we design $\mathcal{L}_t$ to only take the target link into consideration. Its definition goes as follows:

$$\mathcal{L}_t = -[1 - \hat{\mathbf{A}}_t(i,j)]^2, \tag{7}$$

with $\hat{\mathbf{A}}_t(i,j)$ equal to $P(\mathbf{A}_t(i,j))$, the probability generated by DDNE. This can make the time-aware link gradient, $\partial\mathcal{L}_t/\partial S(i,:)$, more concentrated when it is applied in target link attack. The calculation of $\partial\mathcal{L}_t/\partial\mathbf{S}(i,:)$ follows the chain rule and the partial derivative can be obtained by calculating $\partial f(\mathbf{S}(i,:))(i,j)/\partial\mathbf{S}(i,:)$ with DDNE regarded as f, which is described as

$$\begin{aligned}\frac{\partial\mathcal{L}_t}{\partial\mathbf{S}(i,:)} &= 2[1 - \hat{\mathbf{A}}_t(i,j)]\frac{\partial\hat{\mathbf{A}}_t(i,j)}{\partial\mathbf{S}(i,:)} \\ &= 2[1 - \hat{\mathbf{A}}_t(i,j)]\frac{\partial f(\mathbf{S}(i,:))(i,j)}{\partial\mathbf{S}(i,:)}.\end{aligned} \tag{8}$$

Note that $\partial\mathcal{L}_t/\partial\mathbf{S}(i,:)$ is a tensor with the same shape of $\mathbf{S}(i,:)$, and the element $g_k(i,j)$ represents the gradient of linkage (i,j) on the k^{th} snapshot.

4.3 Traversal Search Based TGA

Given a dynamic network $\mathbf{S}$, we aim to find $\triangle\mathbf{S}$ such that the target link could be prevented being predicted from DDNE at a minimum cost. Possible solutions are myriad since there are tens of thousands of nodes and multiple snapshots in $\mathbf{S}$, making it hard to find a proper $\triangle\mathbf{S}$. Chen *et al.* propose to modify links according to Eq. (9) to disturb GCN on the task of node classification. The modification involves both the magnitude and sign of $g(i,j)$, which decides the candidate linkages and how they should be modified, respectively

$$\hat{\mathbf{A}}(i,j) = \mathbf{A}(i,j) + sign(g(i,j)). \tag{9}$$

Considering a simple function $f(x) = ax$ where a is the weight and x denotes the independent variable, the $f(x)$ changes faster if a is larger and the *sign* of a determines the direction in which $f(x)$ changes. This small example gives us insights of the relationship between $\mathcal{L}_t$ and the gradients. To effectively lower $\mathcal{L}_t$ and mislead DDNE subsequently, we need to find the link with maximum magnitude of gradient. The assumption follows that the link with maximum magnitude of gradient contributes the most when making inference. And the sign of gradients, denoted as $sign(g(i,j))$, determines whether the effect is positive or negative. However, DDNE is not a convex function as simple as $f(x)$. And neither GCN or DDNE could learn network features and make prediction flawlessly, which might result in the deviation of gradients. It means that modifying the

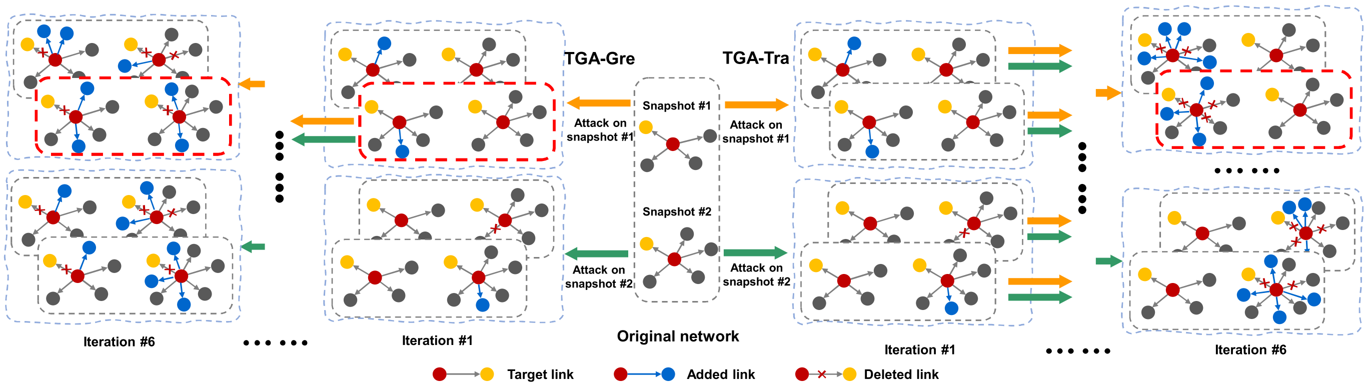


Fig. 2. Illustration of TGA-Tra and TGA-Gre with $n_s = 2$, $\gamma = 6$ and $c = 2$. The network in red dashed box is the optimal one in the corresponding iteration. And the arrow in orange denotes that we modify the link on the first snapshot while the one in green represents that we make attack on second snapshot.

link with maximum magnitude of gradient does not always lead to best attack performance and $sign(g(i,j))$ might express the opposite meaning.

4.3.1 Search Across Snapshots

In TGA-based attack methods, we first group top c links based on the magnitude of the gradients $\partial\mathcal{L}_t/\partial\mathbf{S}(i,:)$ and then find the optimal link to modify within the c candidates. Specifically, within one iteration, we first sort links based on $|g_{ij}|$ with respect to each snapshot, and then select top c links in each snapshot. We then generate candidate adversarial examples by altering the linkage status of the selected links. Specifically, if there is a link between v_i and v_j at the k^{th} snapshot, we then remove e_{ij}; In the reverse case, suppose v_i and v_j are not connected at the k^{th} snapshot, we then can add a link between v_i and v_j on the corresponding snapshot. Note that we do not follow the rule defined in Eq. (9) but just perform attack on the basis of linkage status. When implementing the above steps, we treat the one-link modification on the target snapshot as a basic operation, called *OneStepAttack*, as shown in Algorithm 1.

Algorithm 1. OneStepAttack

```
Input: Original network $\mathbf{S}(i,:)$, the partial derivative
       $\partial\mathcal{L}_t/\partial\mathbf{S}(i,:)$, target snapshot $k$, number of candidate
       adversarial examples $c$;
Output: A set of candidate adversarial networks $\overline{\mathbf{S}}(i,:)$
Initialize $\overline{\mathbf{S}}(i,:) = \mathbf{S}(i,:)$;
Initialize empty candidate adversarial examples set $CA$;
Initialize $Q = \{g_k(i,0),\ldots,g_k(i,n)\}$ and sort it by the magni-
 tudes of their gradient in ascending order.;
for $i = 0$ to $c$ do
  Get the target link based on $Q[i]$;
  Generate a adversarial example and add it to CA;
end
return CA;
```

4.3.2 Search Between Iterations

We iteratively add perturbations to $\mathbf{S}(i,:)$ when performing the attack. In each iteration, the perturbation $\triangle\mathbf{S}$ is designed according to the time-aware link gradients. We have two options for this, one is to use the gradients obtained in the last iteration (for higher efficiency), and the other is to re-calculate them based on the adversarial example generated in the last iteration (for higher effectiveness). Here, we use the second one. We first obtain all possible adversarial examples in γ iterations. After that, we choose the one which could achieve the minimum p_{ij} as the final adversarial example. The procedure of our TGA-Tra is visualized in the left part of Fig. 2. For instance, when we set the number of historical snapshots $n_s = 2$, $\gamma = 6$ and $c = 2$, we will have 2^{12} possible adversarial examples as candidates, and the red dashed box in Iteration 6 is the finally chosen one. Clearly we can get the attack route through backtracking. The details of TGA-Tra are presented in Algorithm 2.

Algorithm 2. TGA-Tra: Attack via Traversal Search

```
Input: A trained DDNE, original network $\mathbf{S}(i,:)$, number of
       modifications $\gamma$, number of candidate adversarial
       examples $c$;
Output: adversarial example: $\hat{\mathbf{S}}(i,:)$
Initialize candidate adversarial examples set $CA = \mathbf{S}(i,:)$;
while $\gamma > 0$ do
  Initialize empty set $\overline{CA}$;
  for each adv_example in CA do
    for $k = 1$ to $n_s$ do
      g = $\partial\mathcal{L}_t/adv\_example$;
      $\hat{\mathbf{S}}$ = OneStepAttack(adv_example, g, k, c);
      Add $\hat{\mathbf{S}}$ to $\overline{CA}$;
    end
  end
  CA = $\overline{CA}$;
  $\gamma = \gamma - 1$;
end
for each adv_example in CA do
  $p(i,j) = DDNE(adv\_example)$;
end
Select $\hat{\mathbf{S}}(i,:)$ as the one with minimum p(i,j) in $CA$;
return $\hat{\mathbf{S}}(i,:)$;
```

4.4 Greedy Search Based TGA

TGA-Tra could be effective since it compares a large number of modification schemes, but it is of relatively high time complexity, especially for large N and n_s. Taking the partial derivatives of the input pairs of nodes (*op1*) and sorting them in descending order (*op2*) are the two most time-consuming steps when performing the attack. For TGA-Tra, we need to repeat the above two steps at most $c^{\gamma}n_s^{\gamma}$ times to hide the target link from being predicted, which is barely affordable in real cases. We thus propose another greedy search method, namely TGA-Gre, as shown in the right part of Fig. 2. Here, in each iteration, we select the one achieving the minimum p_{ij} as the most effective adversarial example, which is further considered as the input of next iteration. The details of TGA-Gre are presented in Algorithm 3.

Algorithm 3. TGA-Gre: Attack via Greedy Search

```
Input: A trained DDNE, original network $\mathbf{S}(i,:)$, target link (i, j), number of modifications $\gamma$, number of candidate adversarial examples $c$;
Output: adversarial example: $\hat{\mathbf{S}}(i,:)$
Initialize $\hat{\mathbf{S}}(i,:) = \mathbf{S}(i,:)$;
while $\gamma > 0$ do
  Initialize empty candidate adversarial example set $CA$;
  for $k = 1$ to $n_s$ do
    g = $\partial\mathcal{L}_t/\hat{\mathbf{S}}(i,:)$;
    $\hat{\mathbf{S}}$=OneStepAttack($\hat{\mathbf{S}}(i,:)$, g, k, c);
    Add $\hat{\mathbf{S}}_{temp}(i,:)$ to CA;
    $p(i,j) = DDNE(\hat{\mathbf{S}})$;
  end
  Select $\hat{\mathbf{S}}(i,:)$ as the one with minimum p(i,j) in $CA$;
  $\gamma = \gamma - 1$
end
return $\hat{\mathbf{S}}(i,:)$
```

For a target link (i,j), TGA-Gre assumes that the lowest p_{ij} in each iteration could lead to the best attack result. It avoids massive comparisons between iterations and thus can significantly accelerate the whole process. Similar to the procedure of TGA-Tra, TGA-Gre also compares g_t across all snapshots in each iteration. The major difference is that TGA-Gre elects a local optimal link in each iteration. It is clear that we only need to repeat *op1* and *op2* at most $\gamma c n_s$ times, which makes the attack much more efficient.

5 Experiments

5.1 Datasets

We perform experiments on six real-world dynamic networks, with their basic statistics listed in Table 1.

- *RADOSLAW* [34]: It is an internal email network between employees in a mid-sized manufacturing company. We focus on the nodes appeared in 2010-01 and construct network using their interactions from 2010-02-01 to 2010-09-01.
- *DNC* [35] It is a directed network of emails in the 2016 Democratic National Committee email leak. We construct the dynamic network using the emails between 2016-04-29 and 2016-05-17 with all the nodes contained in the dataset. The network is divided into 7 snapshots at the interval of 3 days.
- *LKML* [36]: It is also an email network extracted from the linux kernel mailing list. We focus on the users who appeared on the mailing list between 2007-01-01 and 2007-04-01 and slice the data from 2007-04-01 to 2008-10-01 at the interval of 3 months.

TABLE 1
The Basic Statistics of the Three Datasets

Dataset	$\lvert V\rvert$	$\lvert E_T\rvert$	$\bar{d}$	d_{max}	Timespan (days)
RADOSLAW	151	72.2K	27.7	240	242
DNC	1,891	46.7K	1.24	198	731
LKML	2,210	201.7K	7.9	718	731
ENRON	2,628	308.8K	2.23	228	365
WIKI	46,300	66.6K	0.24	14,732	2,191
FLICKR	74,927	90.1K	2.03	1,411	30

$\bar{d}$ represents the average degree of the network and d_{max} is the maximum degree.

- *ENRON* [37]: It is an email network covering decades of the email communication in Enron. We use the records between 2000-04-01 and 2001-10-01 experiments and slice them into 7 snapshots in 3-month increments. Also, we only focus on part of the email address that at least sent an email from 2000-01-01 to 2000-04-01.
- *WIKI* [38]: It is a communication network of the Serbian Wikipedia. The temporal links appeared between 2000-01-01 and 2001-10-01 are first divided into 7 parts at the interval of 3 months and then used for constructing dynamic network.
- *FLICKR* [39]: It records the friendships between users of Flickr. We use the temporal links appeared from 2007-03-01 to 2007-04-05 and focus on the users appeared from 2007-03-01 to 2007-03-06. The network is divided into 7 slices at the interval of 5 days.

As described, all the datasets are divided into 7 snapshots with different intervals. The first snapshot provides the nodes we need to focus and the rest are used for training and inference. All the datasets are available online.[1]

5.2 Baseline Methods

As the first work to study adversarial attack on DNLP algorithms, we design two baseline attacks, RA and CNA, to compare with TGA-Gre and TGA-Tra. Also, we implement another two gradient-based attack methods, FGA and IG-JSMA, as baselines.

- *Random Attack (RA)*: RA randomly modifies γ linkages in all snapshots. In practice, we add b new connections to the target node and remove $\gamma - b$ links between the target node and its neighbors. Here, we use RA to see the robustness of DNLP algorithms under random noises.
- *Common-Neighbor-based Attack (CNA)*: CNA adds b links between node pairs with less common neighbors and remove $\gamma - b$ links between those with more common neighbors. We adopt CNA as a baseline since *common neighbor* is the basis of many similarity metrics between pairwise nodes used for link prediction.
- *Fast Gradient Attack (FGA)* [22]: FGA recursively modify the linkage with maximum absolute gradient obtained by $\partial\mathcal{L}_t/\partial\mathbf{S}(i,:)$, until the attack succeeds, or the number of modifications reaches γ. We use FGA

1. http://konect.cc/

TABLE 2
Attack Performance in Terms of ASR and AML

Dataset	$\mathcal{ASR}$						$\mathcal{AML}$					
	RA	CNA	FGA	IG-JSMA	TGA-Gre	TGA-Tra	RA	CNA	FGA	IG-JSMA	TGA-Gre	TGA-Tra
Attack on top-100 links with the highest existence probability												
RADOSLAW	0.00	0.32	0.86	0.88	0.95	**0.97**	10.00	8.97	7.98	7.58	7.34	**7.10**
DNC	0.00	0.37	0.83	0.85	**0.91**	**0.91**	10.00	8.02	6.33	5.86	5.48	**5.43**
LKML	0.00	0.24	0.68	0.68	**0.77**	**0.77**	10.00	8.47	6.14	6.14	**5.31**	**5.31**
ENRON	0.02	0.33	0.77	0.80	0.82	**0.83**	9.97	8.26	5.56	4.93	4.71	**4.67**
WIKI	0.00	0.40	0.79	0.85	**0.88**	**0.88**	10.00	7.49	5.88	5.07	4.71	**4.71**
FLICKR	0.00	0.19	0.64	0.68	**0.72**	**0.72**	10.00	8.90	6.17	5.88	5.12	**5.12**
Attack on top-100 links with the highest degree centrality												
RADOSLAW	0.12	0.58	0.97	0.99	**1.00**	**1.00**	9.44	6.98	4.27	3.61	3.54	**3.44**
DNC	0.06	0.48	0.89	**0.91**	**0.91**	**0.91**	9.66	7.53	5.66	5.12	4.87	**4.77**
LKML	0.02	0.23	0.56	**0.56**	**0.56**	**0.56**	9.90	8.17	6.78	6.70	**6.49**	**6.49**
ENRON	0.05	0.43	0.80	0.80	0.80	**0.84**	9.65	7.78	6.03	5.96	5.58	**5.47**
WIKI	0.40	0.38	0.58	0.59	0.63	**0.68**	7.02	6.81	5.33	4.99	**4.66**	**4.66**
FLICKR	0.09	0.39	0.95	0.95	0.97	**0.98**	9.50	6.22	3.45	3.43	2.32	**2.25**
Attack on top-100 links with the highest edge betweenness centrality												
RADOSLAW	0.27	0.42	0.98	**1.00**	**1.00**	**1.00**	8.72	5.76	2.89	2.72	2.47	**2.45**
DNC	0.22	0.48	**0.98**	**0.98**	**0.98**	**0.98**	8.66	6.31	3.54	**2.73**	2.83	**2.82**
LKML	0.17	0.37	0.78	0.80	**0.83**	**0.83**	9.01	8.07	5.13	5.08	**4.78**	**4.78**
ENRON	0.13	0.39	0.97	**0.99**	**0.99**	**0.99**	9.28	8.25	4.57	3.94	3.29	**3.21**
WIKI	0.20	0.26	0.65	0.72	0.70	**0.82**	8.75	7.98	5.77	5.02	4.90	**4.43**
FLICKR	0.10	0.31	0.92	0.95	0.95	**0.96**	9.49	8.33	3.09	2.47	2.41	**2.37**

as a baseline to address the importance of utilizing temporal information in attacking DNLP algorithms.

- *IG-JSMA* [40]: IG-JSMA uses integrated gradients to attack GCN under the settings of JSMA (Jacobian-based saliency map attack) [41]. In this paper, we employ part of IG-JSMA which focuses on graph structure manipulation, making it suitable for attacking DDNE. And γ for IG-JSMA is also set to 10.

We set $\gamma = 10$ in all experiments and $b = 5$ for RA and CNA. For TGA-based methods, we set c to 5. We first train the models with original data, and then feed adversarial examples generated by DDNE to each model, to validate the effectiveness of the attacks.

5.3 Evaluation Metrics

We choose *attack success rate* ($\mathcal{ASR}$) and *average attack modification links* ($\mathcal{AML}$) as attack effectiveness criterion.

- $\mathcal{ASR}$: The ratio of the amount of links that are successfully hidden to the total number of target links that can be correctly predicted in the target snapshot.
- $\mathcal{AML}$: The average amount of perturbation to prevent each target link from being predicted. If it needs to modify at least q_i links to hide link i, then $\mathcal{AML}$ is defined as

$$\mathcal{AML} = \frac{1}{N_l}\sum_{i=0}^{N_l-1} q_i, \tag{10}$$

where N_l represents the number of target links. Note that $q_i \leq \gamma$ and the equality holds when the attack fails.

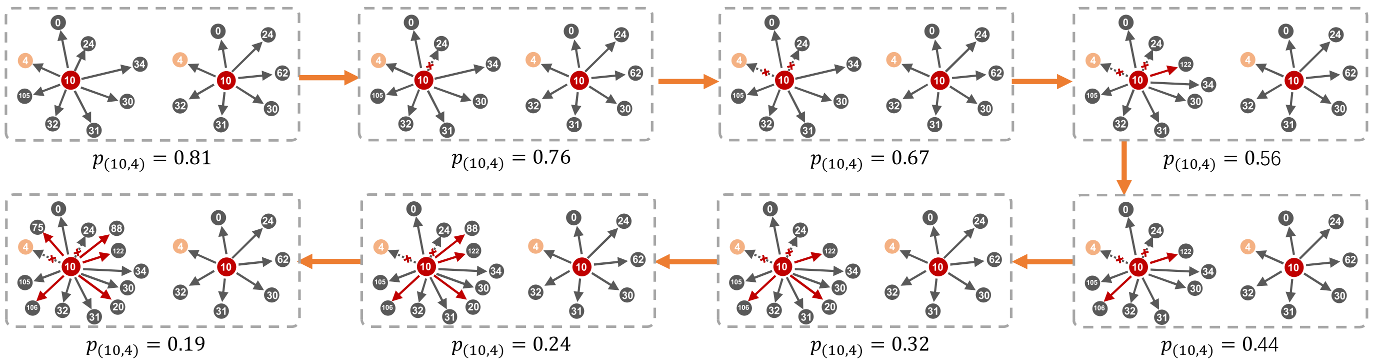
We use $\mathcal{ASR}$ to evaluate attack methods in the first place which reflects the possibility to successfully perform attack and then compare their $\mathcal{AML}$ when their $\mathcal{ASR}$ are close.

5.4 Results

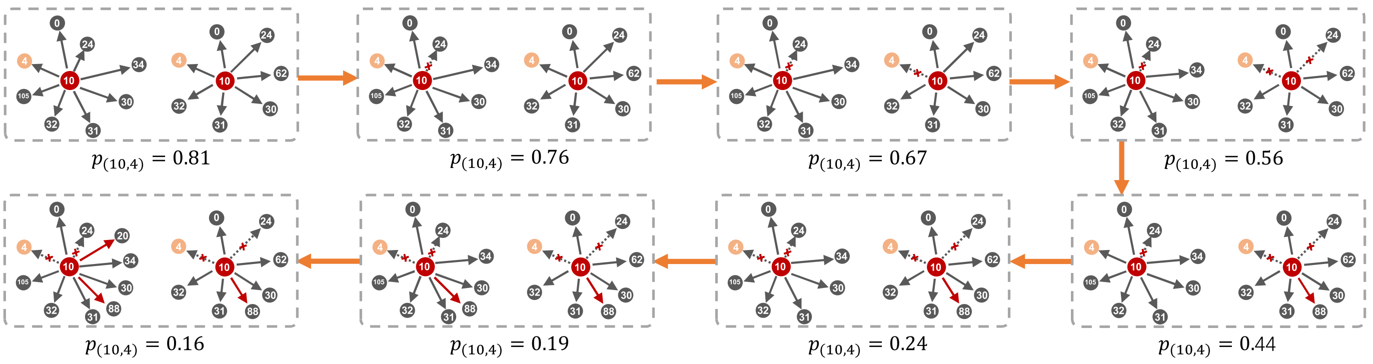
First, we use the generated adversarial examples to fool the DDNE model to prevent target links from being predicted. We set $\gamma = 10$ to ensure the disguise of modification, which also leads the maximum of $\mathcal{AML}$ equaling to 10. The results are presented in Table 2. The two TGA methods outperform FGA and IG-JSMA in terms of both $\mathcal{ASR}$ and $\mathcal{AML}$, while the two gradient-based methods are better than the two heuristic methods, CNA and RA. The results suggest that: 1) the gradients of DDNE is critical to attack different DNLP methods; 2) utilizing temporal information can indeed significantly improve the attack effectiveness. And IG-JSMA is slightly better than FGA which may contribute to the use of integrated gradients. Moreover, we study the attack performance on 3 different types of links: The links that are most likely to exist according to DDNE, the links with highest degree, in terms of the sum of terminal-node degrees, and the links with highest edge betweenness centrality. The other 2 types of links which have physical meaning in real scenarios are easier to be hidden from detection, reflecting the practicability of TGA-based methods.

As expected, TGA-Tra behaves better than TGA-Gre, but the latter is much more efficient and thus more practical in real-world applications. The gap of the performance between TGA-Tra and TGA-Gre overturns the hypothesis that the greatest drop of $\mathcal{L}_t$ in each iteration does not lead to the best attack performance sometimes. This enlightens us to further explore specific meanings behind g_t. Fig. 3 visualizes the attack schemes of TGA-Tra and TGA-Gre performed on $E(10,4)$ of RADOSLAW on #3 snapshot. We find that the performance of TGA-Tra and TGA-Gre are very similar in each iteration, but their routes seem totally different. By investigating these adversarial examples, we have the following two observations:

- First, TGA-Tra is more likely to modify the links on earlier historical snapshots, while TGA-Gre tends to change the links on the most recent ones;



(a) Attack process of TGA-Tra



(b) Attack process of TGA-Gre

Fig. 3. Attack process of TGA-Tra and TGA-Gre on $E(10, 4)$ of RADOSLAW on #3 snapshot.

- Second, TGA-Tra prefers to add rather than delete links, while TGA-Gre has the opposite tendency.

Such observations indicate that TGA-Tra could be more concealed than TGA-Gre, since people tend to pay more attention to recent events, e.g., link change in recent snapshots. On the other hand, TGA-Gre may be preferred if we want to get some short-term attack effect. Besides, TGA-Tra seems to have lower social cost, since adding links are always easier than deleting in our social circle. Since TGA-Gre has similar performance, while is much more efficient, compared with TGA-Tra, we will mainly focus on TGA-Gre in the rest of this paper.

5.4.1 *Attack on Long-Term Prediction*

Besides focusing on the next immediate snapshot, researchers always look into the performance of DNLP algorithms on long-term prediction. That is whether the DNLP algorithms are able to have a good prediction performance on not only the next immediate snapshot but also the snapshots in the remote future. Similarly, we would also like to investigate whether the TGA-based methods are effective for hiding remote future links. To address the problem, we first use DDNE to make predictions for the #3, #4 and #5 snapshot with $n_s = 2$ and then generate adversarial examples for the three snapshots with TGA-Gre. The comparison of the attack performance on the three snapshots are shown in Table 3. We can see that, generally, the performance of TGA-Gre are close in spite that the target snapshots vary. Additionally, there are no significant variation tendency of the attack performance. And we also present the results of TGA-Tra as well as other baselines in Appendix B.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2021.3110580. The results show that TGA-based attack methods are applicable for long-term prediction attack.

5.4.2 *Long-History Attack*

The number of historical snapshots, n_s, is one of the most significant parameters that affect the performance of DNLP

TABLE 3
The Performance of TGA-Gre on Different Snapshots

Dataset	$\mathcal{ASR}$			$\mathcal{AML}$		
	#3	#4	#5	#3	#4	#5
Attack on top-100 links with the highest existence probability						
RADOSLAW	0.95	0.96	0.95	7.34	6.92	6.78
DNC	0.91	0.93	0.92	5.48	4.82	5.14
LKML	0.77	0.94	0.92	5.31	5.52	5.23
ENRON	0.82	1.00	0.95	4.71	3.27	3.28
WIKI	0.88	0.64	0.55	4.71	5.84	5.45
FLICKR	0.72	0.79	0.80	5.12	4.50	4.20
Attack on top-100 links with the highest degree centrality						
RADOSLAW	1.00	1.00	1.00	3.54	3.68	3.55
DNC	0.91	0.95	0.96	4.87	3.73	4.69
LKML	0.56	0.77	0.80	6.49	7.07	6.76
ENRON	0.80	0.86	1.00	3.29	2.84	2.22
WIKI	0.63	0.94	0.59	4.66	2.36	5.12
FLICKR	0.97	0.89	1.00	2.32	2.35	1.25
Attack on top-100 links with the highest edge betweenness centrality						
RADOSLAW	1.00	1.00	1.00	2.47	2.37	2.20
DNC	0.98	0.98	0.98	2.83	2.54	2.68
LKML	0.83	0.92	0.91	4.78	4.53	4.51
ENRON	0.98	1.00	1.00	3.29	2.84	2.22
WIKI	0.70	0.91	0.90	4.90	2.45	2.53
FLICKR	0.95	1.00	0.99	2.41	1.34	1.28

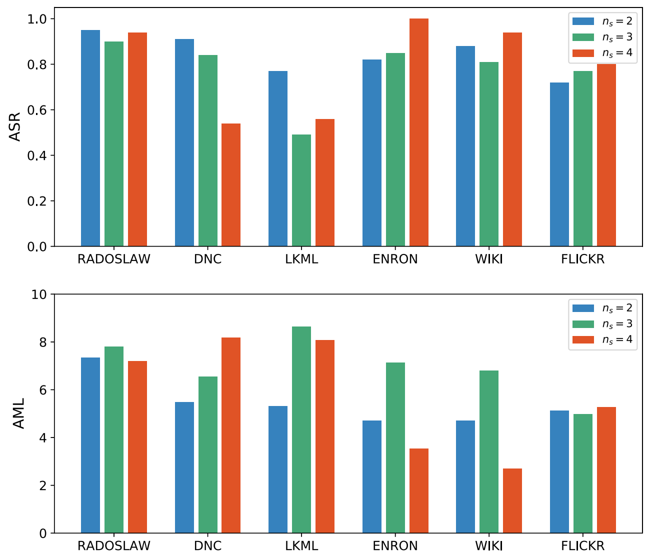


Fig. 4. The performance of TGA-Gre when n_s changes from 2 to 4 (From left to right).

algorithms. Typically, larger n_s means more historical information can be used in prediction, and thus may improve the performance of DNLP algorithms. Higher accuracy will equip the model with more precise gradients. Here, we are interested in whether the increase of n_s will get in the way of adversarial attacks or behave the opposite.

We compare the attack performance of TGA-Gre on the six datasets with respect to different n_s. In particular, we first apply DDNE with the input sequence varying from 2 to 4, and then generate adversarial examples with $\gamma = 10$. The results are shown in Fig. 4, where we can see that, indeed, the performance of TGA-Gre increases more or less as n_s increases, indicating that larger n_s offers more precise gradient information so that TGA-Gre could be more effective. In the mean while, the $\mathcal{AML}$ also increases. One possible reason is that the increase of n_s expands the solution space and TGA-Gre needs to modify more links to make successful attacks. Different from the performance on the other datasets, the attack performance increase slightly when applying TGA-Gre to DNC and LKML as n_s increases. We argue that it is because larger n_s makes DDNE more robust on these two datasets and $\gamma = 10$ is not enough for TGA-Gre to attack on some specific links. Actually, the performance of TGA-Gre will increase if we set $\gamma = 15$.

5.4.3 *Adding-Link Attack*

In social networks, it is considered that deleting links is of higher social cost than adding. Moreover, temporal networks may also have multiple interactions between a pair of nodes. Therefore, deleting one link on a snapshot always removes all the corresponding interactions in the given interval. And some links may be too important to be deleted in real scenarios. Due to this gap between the cost of deleting and adding links, we would like to investigate how the attack performance of TGA-Gre will be influenced if we just add, rather than rewire, links to the original networks. The results are presented in Table 4, where we find that the performance of TGA-Gre slightly decrease when we perform the attack only by adding links. Such results indicate that,

TABLE 4
The Performance of TGA-Gre With Only Adding Links

Dataset	RADOSLAW	DNC	LKML	ENRON	WIKI	FLICKR
Attack on top-100 links with the highest existence probability						
$\mathcal{ASR}$	0.97	0.91	0.77	0.83	0.88	0.72
GAIN	2.11	0.00	0.00	1.22	0.00	0.00
$\mathcal{AML}$	7.10	5.43	5.31	4.67	4.71	5.12
GAIN	-3.27	-0.91	0.00	-0.84	0.00	0.00
Attack on top-100 links with the highest degree centrality						
$\mathcal{ASR}$	1.00	0.98	0.83	0.99	0.82	0.96
GAIN	0.00	0.00	0.00	1.02	17.14	1.05
$\mathcal{AML}$	2.45	2.82	4.78	3.21	4.43	2.37
GAIN	-0.81	-0.35	0.00	-2.43	-9.59	-1.66
Attack on top-100 links with the highest edge betweenness centrality						
$\mathcal{ASR}$	1.00	0.91	0.56	0.84	0.68	0.98
GAIN	0.00	0.00	0.00	5.00	7.94	1.03
$\mathcal{AML}$	3.44	4.77	6.49	5.47	4.66	2.25
GAIN	-2.82	-2.05	0.00	-1.97	0.00	-3.01

in certain cases, we can perform the *cheap* attack on DNLP by only adding a small number of links, at the cost of losing a little bit attack performance.

Also, we are surprised that the performance of TGA-Gre with only adding links on LKML slight increase. It shows that TGA-Gre actually find the best modification schemes on different networks and rewiring links might be the optimal choice in all cases.

5.5 Attack Under Gradient Disturbance

TGA selects candidate links according to $\mathcal{L}_t$ obtained from DDNE while $\mathcal{L}_t$ is easily affected by the weights of DDNE. The change of hyper-parameters, such as the number of training epochs and the coefficient of regularization term, could result in the change of the weights. It may introduce noise into $\mathcal{L}_t$ but not lower the performance of the model. Would the performance of TGA be affected subsequently? We investigate into the problem by adding disturbance to $\mathcal{L}_t$ manually. We first randomly select $\theta|V|$ nodes and then add noise sampled from $\mathcal{N}(\mu, 1)$ where $\mu = 0.2L_1(\mathcal{L}_t)$. Fig. 5 shows the performance of TGA-Gre and TGA-Tra under different ratios of disturbance. Generally, TGA-Gre and TGA-Tra perform relatively stable when θ varies from 0.1 to 0.2. When θ further increases, the performance drops on some datasets. $\mathcal{ASR}$ on LKML and WIKI stays the same as θ increases, indicating the robustness of TGA. And the increase of $\mathcal{AML}$ means TGA-Gre and TGA-Tra need to modify more links to maintain $\mathcal{ASR}$. On FLICKR, the performance of TGA-Gre and TGA-Tra significantly drops when θ increases to 0.3. We argue that there exists some key links on FLICKR which make TGA more sensitive.

5.6 Runtime Comparison

As mentioned in Section 4, we propose TGA-Gre to reduce the computational complexity to make the attack more practical. To highlight the efficiency of TGA-Gre, we compare the runtime of different attack methods performed on the top-100 links with the highest existence probability of DNC with respect to increasing γ. In the runtime comparison, the number of modified links of each method must reach γ no matter it succeeds or not. And we exclude the runtime of TGA-Tra since it is too time-consuming to finish the attack

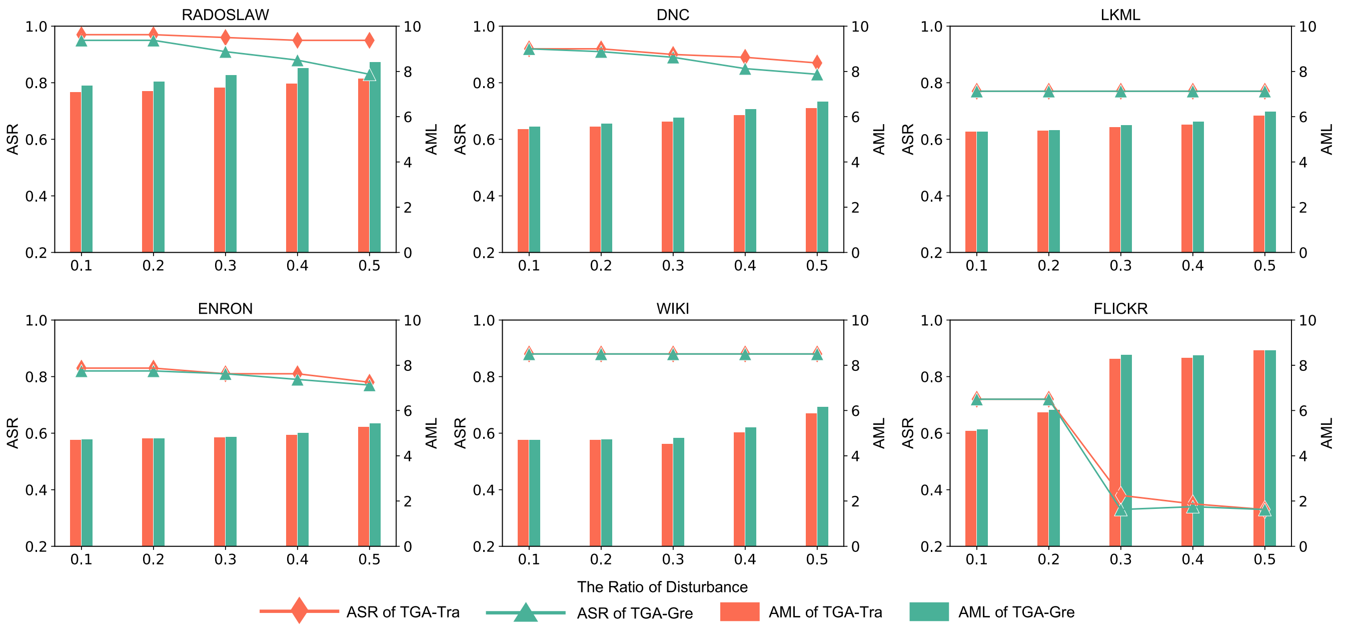


Fig. 5. The performance of TGA-Gre and TGA-Tra under gradient disturbance.

without early stop (stops once the attack succeeds) in an acceptable time. It would take hours for TGA-Tra to attack one link in DNC when $\gamma = 20$. Thus, it is hard to compare the runtime in one figure. In Fig. 6, we can find that the runtime of TGA-Gre, FGA and IG-JSMA are comparable since all of them are gradient-based approaches. Though TGA-Gre is more effective in practice, it is slightly slower for that it needs comparison among different candidate adversarial examples in each iteration. As for RA and CNA, attacking with no gradient information makes them the fastest methods among all the methods despite the poor performance. In fact, if the attack is performed with early stop, TGA-Gre will have shorter runtime than FGA.

5.7 Case Study: Ethereum Transaction Network

Above experiments conducted on six benchmark datasets have shown the effectiveness of TGA-Tra and TGA-Gre on DDNE. In this section, we apply TGA-Gre on the Ethereum transaction network to hide specific transactions from detection. Ethereum is a public blockchain-based platform with the support of smart contract. With around 300 billion USD market capitalization and over 500 billion USD monthly transaction volume, it becomes the largest virtual currency trading platform second to Bitcoin. A bunch of researchers have mined the valuable data with the help of graph analysis [42], [43], [44] among which the analysis of temporal links, i.e., the transactions, is one of the research emphases [43].

We use the data provided by XBlock[2] and extract the transaction records between 2016-02 and 2016-06. The data are sliced into 5 snapshots at the interval of 1 month and modeled as a dynamic network with 2866 nodes which represent the transaction addresses. We focus on two types of accounts: Normal accounts and those belong to Ethereum pool which are identified according to the records on Etherscan.[3] Fig. 7 visualizes #3 snapshot of the Ethereum transaction network from which we can find clusters centered at those Ethereum pool accounts, such as *DwarfPool* and *CoinMine*. To hide some target links in the transaction network from the detection of DDNE, we apply TGA-Gre and TGA-Tra to generate corresponding adversarial examples. Table 5 compares the attack performance of TGA-based methods and the baselines. TGA-Gre and TGA-Tra have the best performance among the five methods. Though IG-JSMA has the same $\mathcal{ASR}$ as TGA-Gre and TGA-Tra does, it needs to modify more links. In practice, adding a link in the transaction network could be costly and thus $\mathcal{AML}$ matters in this occasion.

As we can observe in Fig. 7, most links in the network are those between normal addresses and Ethereum-pool-belonged addresses which are the way normal users making profits from the pools. In real scenarios, we do not pay attentions to these links. Instead, the links between normal addresses are noteworthy. Suppose a user want to make a vital transaction recondite in the near future, he or she could

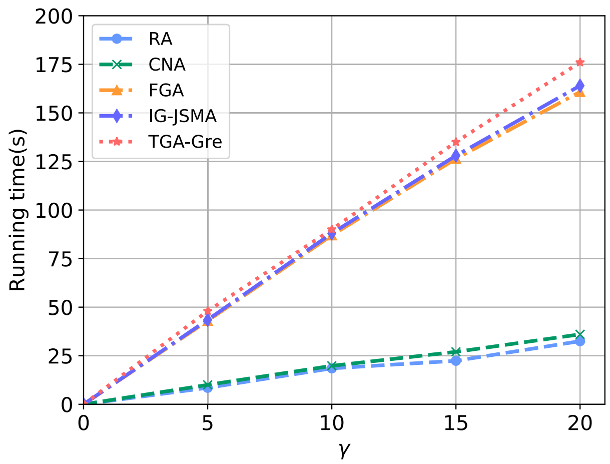


Fig. 6. Runtime of different attack methods performed on the top-100 links with the highest existence probability of DNC.

[2] http://xblock.pro
[3] https://etherscan.io/

TABLE 5
Attack Performance on Ethereum Transaction Network

	RA	CNA	FGA	IG-JSMA	TGA-Gre	TGA-Tra
$\mathcal{ASR}$	0.00	0.13	0.91	0.94	0.94	0.94
$\mathcal{AML}$	10.00	9.67	8.96	8.55	8.02	8.02

make transfers to certain addresses with the guidance of TGA. Then the transaction will not be discovered from current time scope even with the help of DDNE.

6 Conclusion

In this paper, we present the first work of adversarial attack on DNLP, and propose the time-aware gradient, as well as two TGA methods, namely TGA-Tra and TGA-Gre, to realize the attack. Comprehensive experiments have been carried out on six benchmark networks and also the Ethereum transaction network. The results show that our TGA methods behave better than the other baselines, achieving the state-of-the-art attack performance on DNLP. Besides, we investigate into the performance of TGA-Gre in several typical occasions in DNLP, including long-term prediction and long-history prediction. Interestingly, we also find that long-term prediction seems to be more vulnerable to adversarial attacks, while using longer historical information can enhance the robustness of DNLP algorithms. The results of adding-link attack also prove the practicability of TGA-Gre.

Currently, TGA methods rely on the gradients and it requires researchers know every detail of the target DNLP model which limits its application scenarios. In the future, we would like to study the problem of *black-box attack* on DNLP and further propose better strategies to improve their attack performance; on the other hand, we will also seek for methods to defend against such adversarial attacks, to achieve more robust DNLP algorithms. Besides, we would investigate into efficient adversarial attack methods on large-scale networks to extend its practicability.

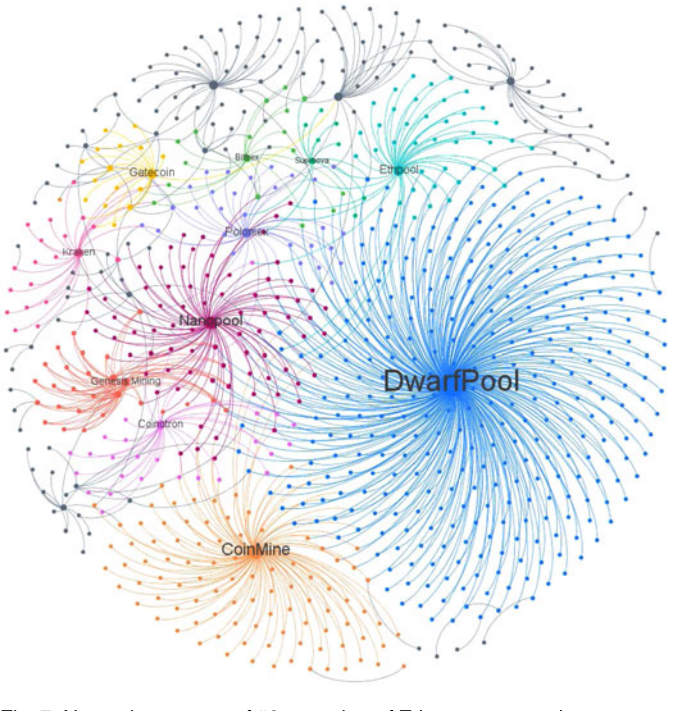


Fig. 7. Network structure of #3 snapshot of Ethereum network.

Acknowledgments

The authors would like to appreciate all the members in the IVSN Research Group, Zhejiang University of Technology for the valuable discussion. This work was supported in part by the National Natural Science Foundation of China under Grants 62072406, 61572439 and 61973273, in part by the Zhejiang Provincial Natural Science Foundation of China under Grants LR19F030001 and LY19F020025.

References

[1] Y. Gong *et al.*, "Exact-K recommendation via maximal clique optimization," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 617–626.
[2] S. Zhang, J. Ai, and X. Li, "Correlation between the distribution of software bugs and network motifs," in *Proc. IEEE Int. Conf. Softw. Qual. Rel. Secur.*, 2016, pp. 202–213.
[3] V. Fionda and G. Pirro, "Community deception or: How to stop fearing community detection algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 660–673, Apr. 2018.
[4] İ. Güneş, Ş. Gündüz-Öğüdücü, and Z. Çataltepe, "Link prediction using time series of neighborhood-based node similarity scores," *Data Mining Knowl. Discov.*, vol. 30, no. 1, pp. 147–180, 2016.
[5] A. Özcan and Ş. G. Öğüdücü, "Multivariate temporal link prediction in evolving social networks," in *Proc. IEEE/ACIS 14th Int. Conf. Comput. Inf. Sci.*, 2015, pp. 185–190.
[6] C. Fu *et al.*, "Link weight prediction using supervised learning methods and its application to Yelp layered network," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1507–1518, Aug. 2018.
[7] Q. Xuan *et al.*, "Modern food foraging patterns: Geography and cuisine choices of restaurant patrons on Yelp," *IEEE Trans. Comput. Soc. Syst.*, vol. 5, no. 2, pp. 508–517, Jun. 2018.
[8] W. Jin, Y. Li, H. Xu, Y. Wang, and J. Tang, "Adversarial attacks and defenses on graphs: A review and empirical study," 2020, *arXiv:2003.00653*. [Online]. Available: https://arxiv.org/abs/2003.00653
[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl
[10] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, "A deep learning approach to link prediction in dynamic networks," in *Proc. SIAM Int. Conf. Data Mining*, 2014, pp. 289–297.
[11] T. Li, B. Wang, Y. Jiang, Y. Zhang, and Y. Yan, "Restricted Boltzmann machine-based approaches for link prediction in dynamic networks," *IEEE Access*, vol. 6, pp. 29 940–29 951, May 2018.
[12] J. Chen *et al.*, "E-LSTM-D: A deep learning framework for dynamic network link prediction," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 51, no. 6, pp. 3699–3712, 2021.
[13] T. Li, J. Zhang, S. Y. Philip, Y. Zhang, and Y. Yan, "Deep dynamic network embedding for link prediction," *IEEE Access*, vol. 6, pp. 29 219–29 230, May 2018.
[14] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowl.-Based Syst.*, vol. 187, 2020, Art no. 104816.
[15] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 388–396.
[16] X. Wu, J. Wu, Y. Li, and Q. Zhang, "Link prediction of time-evolving network based on node ranking," *Knowl.-Based Syst.*, vol. 195, p. 105740, 2020.
[17] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, "Continuous-time dynamic network embeddings," in *Proc. Companion Web Conf.*, 2018, pp. 969–976.
[18] S. Nagaraja, "The impact of unlinkability on adversarial community detection: Effects and countermeasures," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2010, pp. 253–272.

[19] M. Waniek, T. P. Michalak, M. J. Wooldridge, and T. Rahwan, "Hiding individuals and communities in a social network," *Nat. Hum. Behav.*, vol. 2, no. 2, pp. 139–147, 2018.

[20] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 2847–2856.

[21] D. Zügner, O. Borchert, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on graph neural networks: Perturbations and their patterns," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 5, Jun. 2020, Art no. 57.

[22] J. Chen, Y. Wu, X. Xu, Y. Chen, H. Zheng, and Q. Xuan, "Fast gradient attack on network embedding," 2018, *arXiv:1810.10751*. [Online]. Available: https://arxiv.org/abs/1810.10751

[23] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *Proc. 7th Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bylnx209YX

[24] X. Wang, J. Eaton, C.-J. Hsieh, and F. Wu, "Attack graph convolutional networks by adding fake nodes," 2018, *arXiv:1810.10751*. [Online]. Available: https://arxiv.org/abs/1810.10751

[25] J. Li, T. Xie, L. Chen, F. Xie, X. He, and Z. Zheng, "Adversarial attack on large scale graph," *IEEE Trans. Knowl. Data Eng.*, p.1, 2021.

[26] A. Bojchevski and S. Günnemann, "Adversarial attacks on node embeddings via graph poisoning," in *Proc. 36th Int. Conf. Mach. Learn. Res.*, 2019, pp. 695–704.

[27] M. Sun *et al.*, "Data poisoning attack against unsupervised node embedding methods," 2018, *arXiv:1810.12881*. [Online]. Available: https://arxiv.org/abs/1810.12881

[28] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.

[29] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *Proc 24th Int. Conf. World Wide Web*, 2015, pp. 1067–1077.

[30] F. Feng, X. He, J. Tang, and T.-S. Chua, "Graph adversarial training: Dynamically regularizing based on graph structure," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 6, pp. 3493–2504, Jun. 2019.

[31] H. Jin and X. Zhang, "Latent adversarial training of graph convolution networks," in *Proc. ICML Workshop Learn. Reasoning Graph-Structured Representations*, 2019. [Online]. Available: https://graphreason.github.io/papers/35.pdf

[32] C. Qin *et al.*, "Adversarial robustness through local linearization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 13 847–13 856.

[33] F. Tramèr and D. Boneh, "Adversarial training and robustness for multiple perturbations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 5866–5876.

[34] Manufacturing emails network dataset – KONECT, Apr. 2017. [Online]. Available: http://konect.uni-koblenz.de/networks/radoslaw_email

[35] DNC co-recipient network dataset – KONECT, Sep. 2016. [Online]. Available: http://konect.uni-koblenz.de/networks/dnc-temporalGraph

[36] Linux kernel mailing list replies network dataset – KONECT, Sep. 2016. [Online]. Available: http://konect.uni-koblenz.de/networks/lkml-reply

[37] R. Rossi and N. Ahmed, "Network repository," 2013. [Online]. Available: http://networkrepository.com

[38] J. Sun, J. Kunegis, and S. Staab, "Predicting user roles in social networks using transfer learning with feature transformation," in *Proc. Int. Conf. Data Mining Workshops*, 2016, pp. 128–135.

[39] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the Flickr social network," in *Proc. Workshop Online Soc. Netw.*, 2008, pp. 25–30.

[40] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: Deep insights into attack and defense," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4816–4823, doi: 10.24963/ijcai.2019/669.

[41] R. Wiyatno and A. Xu, "Maximal Jacobian-based saliency map attack," 2018, *arXiv:1808.07945*. [Online]. Available: https://arxiv.org/abs/1808.07945

[42] T. Chen *et al.*, "Understanding Ethereum via graph analysis," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 1484–1492.

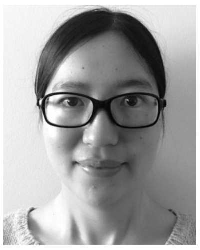
[43] D. Lin, J. Wu, Q. Yuan, and Z. Zheng, "T-EDGE: Temporal weighted multidigraph embedding for ethereum transaction network analysis," *Front. Physics*, vol. 8, p. 204, 2020.

[44] D. Lin, J. Wu, Q. Yuan, and Z. Zheng, "Modeling and understanding Ethereum transaction records via a complex network approach," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 67, no. 11, pp. 2737–2741, Nov. 2020.

Jinyin Chen received the BS and PhD degrees from the Zhejiang University of Technology, Hangzhou, China, in 2004 and 2009, respectively. She studied evolutionary computing with the Ashikaga Institute of Technology, Japan, in 2005 and 2006. She is currently a professor with the Institute of Cyberspace Security and the College of Information Engineering, Zhejiang University of Technology. Her research interests include artificial intelligence security, graph data mining, and evolutionary computing.

Jian Zhang received the BS degree in automation from the Zhejiang University of Technology, Hangzhou, China, in 2017. He is currently working toward the PhD degree with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. He is currently focusing on network analysis and deep learning, especially the intersection of the two fields.

Zhi Chen received the BS and MS degrees in EECS from UC Berkeley, in 2019 and 2020, respectively. He is currently working toward the PhD degree in computer science with the University of Illinois at Urbana-Champaign. At UC Berkeley, he was a research assistant with the Center for Long-Term Cybersecurity from 2019 to 2020, and with BAIR Lab in 2018. His research interests include security and machine learning.



Min Du received the bachelor's and master's degrees from Beihang University, Beijing, China, and the PhD degree from the School of Computing, University of Utah, Salt Lake City, Utah, in 2018. She was a postdoctoral scholar in EECS Department, UC Berkeley from 2018 to 2019. She is currently doing AI security research in Palo Alto Networks. Her research interests include big data analytics and machine learning security.

Qi Xuan (Senior Member, IEEE) received the BS and PhD degrees in control theory and engineering from Zhejiang University, Hangzhou, China, in 2003 and 2008, respectively. He was a postdoctoral researcher with the Department of Information Science and Electronic Engineering, Zhejiang University, from 2008 to 2010, and a research assistant with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, in 2010 and 2017. From 2012 to 2014, he was a postdoctoral fellow with the Department of Computer Science, University of California at Davis, California. He is currently a professor with the Institute of Cyberspace Security, College of Information Engineering, Zhejiang University of Technology, Hangzhou, China. His current research interests include network science, graph data mining, cyberspace security, machine learning, and computer vision.