Adversarial Attacks on Link Prediction Algorithms
Based on Graph Neural Networks

Wanyu Lin
University of Toronto
wanyu.lin@mail.utoronto.ca

Shengxiang Ji
University of Toronto
shengxiang.ji@mail.utoronto.ca

Baochun Li
University of Toronto
bli@ece.toronto.edu

ABSTRACT
Link prediction is one of the fundamental problems for graph-structured data. However, a number of applications of link prediction, such as predicting commercial ties or memberships within a criminal organization, are adversarial, with another party aiming to minimize its effectiveness by manipulating observed information about the graph. In this paper, we focus on the feasibility of mounting adversarial attacks against link prediction algorithms based on graph neural networks. We first propose a greedy heuristic that exploits incremental computation to find attacks against a state-of-the-art link prediction algorithm, called SEAL. We then design an efficient variant of this algorithm that incorporates the link formation mechanism and $\gamma$-decaying heuristic theory to design more effective adversarial attacks. We used real-world datasets and performed an extensive array of experiments to show that the performance of SEAL is negatively affected by a significant margin. More importantly, our experimental results have shown that our adversarial attacks mounted based on SEAL can be readily transferred to several existing link prediction heuristics in the literature.

CCS CONCEPTS
- Security and privacy → Network security.

KEYWORDS
Adversarial Attacks; Graph Neural Networks; Link Prediction

ACM Reference Format:

1 INTRODUCTION
Link prediction refers to the problem of identifying the existence of a link between two nodes in a network [22]. It is an important problem with practical applications in a diverse set of research fields, including friend recommendation in social networks [14], prediction and ranking algorithms in complex networks (e.g., co-authorship graphs) [25], and criminal networks [7]. In criminal networks, for example, links between entities indicate that potential connections between these entities exist, such as having commercial ties or memberships in the same criminal organization. These potential links provide useful underlying information about network structures, and may be readily detected by link prediction algorithms.

A large number of link prediction algorithms have been introduced in the literature. Existing approaches can be categorized into two classes. The first is heuristic methods which use predefined similarity functions to measure the likelihood of links [2, 17, 19, 22]. Although they worked well in practice, these heuristics make strong assumptions on when links may exist, and none of them performs consistently well across complex networks [22]. The second is learning-based methods, which automatically learn a mapping function from the network [3, 31, 33, 34]. Specifically designed for graph-structured data, graph neural networks have shown to achieve state-of-the-art performance when solving link prediction problems [34].

However, as effective as they may be, recent studies have shown that neural networks, in general, are vulnerable to malicious adversaries, who are able to craft specific sets of adversarial examples so that neural network models will generate desired outputs of their choice. Typically, these selected adversarial inputs are derived from regular inputs by introducing minor — yet carefully selected — perturbations. Such adversarial attacks have been widely demonstrated with high success rates in the contexts of image recognition [8] and malware detection [15]. Interestingly, it remains unclear how effective such adversarial attacks may be for link prediction algorithms based on graph neural networks.

Adversarial perturbations on the graphs underlying complex networks, especially on social networks, are easily conceivable and quite common in practice. As link prediction may reveal connections which associated parties prefer to keep hidden — either for the sake of profit, or to evade the law enforcement. For example, in online recommendation systems, fraudsters frequently manipulate online reviews to affect reader opinion in recommendation networks [12]. In a criminal network, criminals may try to hide their links to bypass the detection of criminal groups [7, 23].

In order to systematically study the ability of an "adversary" to manipulate link prediction, we mount adversarial attacks on link prediction via applying existing evasion attacks in adversarial machine learning. For this purpose, we first formulate the problem of crafting adversarial examples to deceive GNN-based link prediction models as an optimization problem. In particular, we focus on evasion attacks against a state-of-the-art link prediction algorithm,
called SEAL [34], which learns missing/unobserved links from local enclosing subgraphs. Essentially, SEAL is proposed based on a \(Y\)-decaying heuristic theory, which shows that graph structure features can be well approximated from the local subgraphs and is able to unify a wide range of heuristics in a single framework. In this regard, we can envision that the mounted attacks may be transferred to the heuristics which can be well incorporated into the \(Y\)-decaying heuristic framework.

Attacking the graph in a complex network effectively involves several non-trivial challenges. First, due to the inherent learning characteristics of SEAL, our problem of crafting perturbations on graph data contains dependent variables as the adjacency matrix and node information matrix are coupled, and existing solutions of gradient-based approaches are not applicable. Second, unlike existing adversarial attacks in the domain of image recognition [8] consisting of continuous data, graph-structured data are typically discrete. With the data’s discreteness and the large number of model parameters of SEAL, solving the optimization problem for a complex network graph is highly challenging. Finally, adversarial perturbations — such as adversarial images in the context of image recognition — should not be noticeable by humans in general. Yet, in complex network graphs, the notion of “unnoticeable changes” needs to be clearly defined first.

Inspired by Zaguner et al. [37], we first propose a greedy heuristic that perturbs the network graph incrementally by manipulating the graph structure. We then propose an efficient variant that utilizes the link formation mechanism and the \(Y\)-decaying heuristic theory. To validate the effectiveness of our crafted attacks, we use real-world datasets to perform an extensive array of experiments. Our results have shown convincing evidence that the performance of link prediction in SEAL has been negatively affected by a significant margin using our adversarial attack, even with very limited adversarial perturbations — such as adversarial images in the context of image recognition. Table 1 summarizes eight popular adversarial heuristics and associated heuristic formula, which will be used to analyze the transferability of our mechanisms. Note that due to the large literature, we could not analyze to every heuristic, but to some popular ones.

### Table 1: Popular Heuristics for Link Prediction

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Heuristic Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common neighbors (CN)</td>
<td>(</td>
</tr>
<tr>
<td>Jaccard [20]</td>
<td>(</td>
</tr>
<tr>
<td>Preference attachment (PA) [4]</td>
<td>(</td>
</tr>
<tr>
<td>Adam-Adar (AA) [2]</td>
<td>(\sum_{(x,y) \in \Gamma(x) \cap \Gamma(y)})</td>
</tr>
<tr>
<td>Katz index [19]</td>
<td>(\sum_{i \in \Gamma(x)} \beta^i \text{path}(x,y) = l)</td>
</tr>
<tr>
<td>PAGERANK [6]</td>
<td>(q_{xy} + q_{yx})</td>
</tr>
<tr>
<td>SimRank [17]</td>
<td>(\gamma \sum_{x \in \Gamma(x)} \sum_{y \in \Gamma(y)} \text{score}(ab))</td>
</tr>
</tbody>
</table>

Note: \(\Gamma(x)\) and \(\Gamma(y)\) denote the sets of \(x\) and \(y\)’s one-hop neighboring nodes, respectively; \(\beta\) is a damping factor; \(|\text{path}(x,y) = l|\) represents the number of length-\(l\) paths between nodes \(x\) and \(y\); \(q_{xy}\) is the station probability distribution of \(y\) under the random walk from \(x\).

#### 2.2 Graph Neural Networks

Graph neural networks (GNNs) represent a new type of neural networks that are capable of learning from graphs. A graph neural network for graph classification typically consists of two main components: graph convolutional layers that extract local substructure features for individual nodes, and a graph aggregation layer that aggregates node-level features into a graph-level feature vector.

Deep graph neural networks (DGNN) are GNNs equipped with propagation-based graph convolution layers. They have been shown to achieve state-of-the-art graph classification performance on various benchmark datasets [35]. The aggregation layer in a DGNN is a SortPooling layer, which sorts the final node states to obtain
an isomorphism invariant node ordering, and enables a traditional 1-D convolutional neural network on the node sequence. Its last layer is a fully-connected layer followed by a log-softmax layer.

### 2.3 The SEAL Framework

In this paper, we focus on the problem of crafting adversarial examples for link prediction based on graph neural networks. In particular, we consider a state-of-the-art link prediction framework, called SEAL [34], which learns heuristics from local enclosing subgraphs using a graph neural network. The foundation of this framework is a $\eta$-decaying heuristic theory, which shows that local enclosing subgraphs reserve rich information for link existence prediction.

Particularly in [34], Zhang et al. proposed a $\eta$-decaying heuristic theory that is able to unify a wide range of heuristics in a single framework, and proved that several existing heuristics, including Katz index [19], PAGERANK [6], and SimRank [17] can be well approximated from local enclosing subgraphs. The $\eta$-decaying heuristic for $(x, y)$ has the following form:

$$H(x, y) = \eta \sum_{l=1}^{\infty} \lambda^l f(x, y, l)$$

where $\lambda \in (0, 1)$ is a decaying factor, $\eta > 0$ is either a constant or a function of $\lambda$ bounded by a constant, $f$ is a nonnegative function under the given network.

**The $\eta$-decaying heuristic theory for link prediction:** Given a $\eta$-decaying heuristic, if $f(x, y, l)$ satisfies the following two conditions:

- $f(x, y, l) \leq \lambda^l$ where $\lambda < \frac{1}{\eta}$;
- $f(x, y, l)$ can be achieved from the $h$-hop subgraph of $(x, y)$ for $l = 1, 2, 3, \ldots, g(h)$, where $g(h) = ah + b$, $a, b \in \mathbb{N}$, $a > 0$.

Then the $\eta$-decaying heuristic for $(x, y)$ can be approximated from the $h$-hop enclosing subgraph of $(x, y)$ and the approximation error decreases at least exponentially with $h$.

Following this theory, as illustrated in [34], several existing heuristics inherently share the same $\eta$-decaying heuristic form, which implies that from the small enclosing subgraphs extracted around links, it is able to approximate a wide range of heuristics with small errors.

In this regard, the SEAL framework is designed to automatically learn a ‘heuristic’ function that maps local enclosing subgraph patterns to link existence instead of using predefined ones. It contains three stages: extracting an $h$-hop local subgraph, either for a training or a testing link; constructing the node information matrix $X$ for each link, and then learning with a graph neural network. The input of the graph neural network consists of $(A, X)$ tuples, where $A$ represents the adjacency matrix of the subgraph, and the output of the graph neural network consists of link labels $c$. The optimal model parameters $W$ are learned by minimizing the cross-entropy on the output of the training links. At test time, the link existence of two nodes then can be predicted by applying the trained SEAL model.

For clarity, let’s see a running example as illustrated in Fig. 1. In this example, the SEAL framework learns the ‘heuristic’ function using 1-hop enclosing subgraphs. The learned heuristic may contain information about its graph structure features, such as the number of common neighbours, Jaccard, and Katz index, etc. $(A, B)$ and $(C, D)$ are links with labels and regarded as training links.

![Figure 1: The SEAL framework: learning graph structure features from 1-hop local enclosing subgraphs: (A, B) and (C, D) are links with labels and regarded as training links.](image)

#### 2.4 Attack Transferability

Existing work in the literature on adversarial machine learning demonstrated that adversarial examples produced to mislead a specific model are highly likely to mislead other models; such property is referred to as transferability. A practical impact of this property is that it leads to oracle-based black-box attacks. More specifically, the adversary is able to use the target model as an oracle to label a synthetic training set for the surrogate, so the adversary need not even observe the full data to mount the attack [21, 26].

The transferability of adversarial machine learning has been extensively studied in the literature [26, 27, 29]. In this paper, the definition of transferability is more general and not only limited among machine learning models. Note that, we aim to offer a comprehensive algorithmic investigation of the problem of attacking link prediction algorithms based on graph neural networks. For this purpose, we focus on evasion attacks against a GNN-based framework, called SEAL, which is proposed based on a $\eta$-decaying heuristic theory. Regarding the $\eta$-decaying heuristic framework, we can envision that the mounted attacks may be transferred to the heuristics.

To illustrate the effectiveness of our attacks, we will empirically analyze their transferability to existing heuristics in Sec. 5.

### 3 Problem Formulation

In this section, we will describe our threat model and explain our attacks as modifications to a graph. In practice, the adversary changes the graph based on the underlying data that are explored for the
prediction of missing (or unobserved) links. For example, in criminal networks, the adversary aims to hide connections between the entities to avoid being detected in criminal investigations.

3.1 Notations

An undirected graph \( G \) is defined by the sets of nodes \( V \) and edges \( E \). \( G \) is such a graph that represents the underlying data – a defender analyses the missing links based on its observed information (e.g., observed graph structure, etc.).

Often when applied, a defender learns a prediction model – described as \( F \) in Sec. 2 – according to its observed graph and seeks to predict the link existence \( e(x, y) \) given any two target nodes \( x, y \in V \). In this paper, an adversary controls an attacker subset \( V_s \in V \). The adversary is capable of accessing and performing perturbations on this subset within a graph \( G = (V, E) \), leading to the graph \( G' = (V', E') \), such that the link prediction model learned is fooled.

3.2 Threat Model

Before describing the adversary’s knowledge, we first discuss the knowledge that is available to the adversary. We assume that the adversary has an active infection set to manipulate, or \( V_s \in V \). This reflects the real-world situation that some of the links are intensively monitored and can be easily detected if an attack occurs.

On the other hand, the adversary has full knowledge of the link prediction algorithms, including the generation of the node information matrix, as well as its learning algorithm. In other words, we assume in this paper that the adversary has complete access to the graph neural networks, including the architecture and model parameters, and can use them in a white-box manner. This is a conservative and realistic assumption: due to the transferability property as illustrated in Sec. 2.4, it is possible to train a surrogate model given black-box access to a target model, and by attacking the surrogate model, the adversary can transfer the attacks to the target [21, 26].

As in the literature [37], we also assume that the adversary has perfect knowledge of the graph \( G \), obtained from the defender. Given the full dataset and the knowledge of the modeling process, the adversary can completely reconstruct the link prediction results as the defender does to evaluate the effectiveness of their attacks. Ideally, this data would be well guarded, making this level of knowledge only realistic for the most sophisticated adversaries. Nevertheless, considering the damage that could be done by a perfectly knowledgeable adversary is important as a security evaluation, since it allows us to find potential weaknesses in link prediction models.

Our attack architecture is shown as Fig. 2. During the testing phase, a testing link \((E, F)\) is predicted as a negative link, while with a perturbation (adding an edge denoted as a blue link in Fig. 2), it is predicted as a positive link. Since \( X \) contains information about a node, including node structural labels, manipulating perturbations on the graph structure (adding or deleting edges) would lead to changes in both \( A \) and \( X \) (coupled variables while performing perturbations). Without loss of generality, attacks induced by adding or deleting edges are referred to as graph structure attacks. Directly manipulating the target link is easy to be detected; thus we also assume that the adversary would not add or delete an edge between the target nodes. To summarize, as the inputs of the prediction model are \((A, X)\) tuples representing the enclosing subgraph of two given target nodes \( x \) and \( y \), perturbations causing changes on \( A \) and \( X \) may lead to an incorrect prediction of the target link \( e(x, y) \).

3.3 Unnoticeability Constraint

In typical application domains, a successful adversarial example is crafted under some simple constraints to ensure its unnoticeability. For example, in the image recognition domain, the perturbation constraint is measured by the distance \((l_0, l_1, l_2, l_{\infty}-\text{norm}, \text{etc.})\) between the adversarial example and its normal example. Its effectiveness can be easily verified by human vision [8, 11]. However, in a complex network graph, manipulating the input data to fool its learning model is much harder.

To quantitatively evaluate unnoticeability, we use the perturbation constraint measured by \( l_1 \)-norm distance of the graph adjacency matrices before and after perturbations. It can be formulated as:

\[
|A - A'| \leq \Delta \tag{2}
\]

where \( A, A' \) are the adjacency matrices of the subgraphs before and after perturbations. It sets the maximum bound that the adversary can change the graph, and with this constraint it is more likely to satisfy the unnoticeability constraint.

Instead of verifying by human vision, we employ the graph property preservation technique to ensure its unnoticeable perturbations, which has been discussed by Zagour et al. [37]. Precisely, we use degree distribution preservation to ensure unnoticeable perturbations in the graph – likelihood ratio test for the power-law degree distribution of the two graphs [37]. The intuition is that two highly similar graphs would follow similar power-law behaviour regarding their degree distributions. According to [37], the graph structure perturbations \( G' = (V', E', X') \) can be accepted only when the degree distribution satisfies:

\[
\Lambda(G^{(0)}, G') < \tau \approx 0.004 \tag{3}
\]
where $\Lambda$ denotes the log-likelihood ratio test statistic according to the graphs’ power-law degree distributions; it follows a $\chi^2$ distribution with one degree of freedom. $\tau$ is approximated using the critical $p$-value setting in the $\chi^2$ distribution.

### 3.4 Attacks as an Optimization Problem

As is commonly done on evasion attacks in other application domains, such as image [8] and audio [9], we formulate the problem of generating adversarial perturbations for the link prediction task as follows: given any two target nodes $x, y$, we solve the following problem:

$$
\begin{align*}
& \text{maximize} \quad F(A', X')c - F(A, X')c \\
& \text{subject to} \quad |A - A'| \leq \Delta \quad (4)
\end{align*}
$$

where $F$ is the pre-trained model that the adversary aims to fool, $F(A', X')$ is the output of the log-softmax layer of the graph neural network, $G^{(0)}$ represents the initial graph, $(A, X)$ and $(A', X')$ denote the enclosing subgraphs of the target nodes $(x, y)$ extracted from $G^{(0)}$ and $G'$, respectively. $c$ and $c'$ indicate the predicted labels of $e(x, y)$ before and after perturbations. For simplicity, we use $f(A', X')$ to represent the loss function $F(A', X')c - F(A, X')c$, which is designed to measure how close $(A', X')$ is to a successful attack.

### 4 GENERATING ADVERSARIAL ATTACKS

Solving the optimization problem as illustrated in Sec. 3.4 is non-trivial. While the optimization-based problem for adversarial attacks has been addressed in the literature using gradient-based computations [8, 9, 15], these existing solutions are not applicable in our case.

Except for its discreteness property and non i.i.d. of the graph data, the perturbation variables used to optimize the objective function are not independent. Precisely, as discussed in Sec. 2.3, the node information matrix contains the node structural label, which is one kind of graph structural features. With a graph structure attack, not only the adjacency matrix but also the node information matrix would be changed, which implies that perturbations on $A$ and $X$ are coupled variables in our problem.

In this section, we will describe the algorithms that we use to overcome the challenges of crafting adversarial examples to fool the link prediction model. In particular, the goal of generating adversarial examples in the complex network graph is to mislead the SEAL framework, causing the link predicted results to be incorrect.

#### 4.1 Greedy Graph Structure Perturbation

To cope with data discreteness and variable dependencies, we adopt a locally optimal strategy that perturbs the graph one at a time using an optimal way to manipulate the graph structure (by adding or deleting an edge). To be unnoticeable, the total perturbation magnitude is limited by $\Delta$ as shown in Eq. (2).

For each particular perturbation, we select the one that can achieve the maximum loss function as Eq. (4), from its current feasible graph structure perturbation space $S_{struct}$ (constructed based on Alg. 1 or Alg. 2 which will be explained later). After each effective perturbation, the enclosing subgraph of the target nodes is changed and requires to be re-extracted from the perturbed graph. With this renewed subgraph, a new pair of $(\Lambda_{xy}^{(t)}, X_{xy}^{(t)})$ is generated as the current state of the subgraph and would be the input of the next perturbation. The graph structure perturbation would terminate either due to a successful attack ($f(A', X') > 0$) or the maximum perturbation constraint $\Delta$.

As we mentioned above, the variables $(A', X')$ that are used to optimize the loss function, regarding the structure attack, are dependent. Thus, the typical approaches of using gradient-based search for each perturbation are not applicable in our case. To solve Eq. (4) with dependent variables, the most intuitive way to construct $S_{struct}$ is to employ a heuristic search under its unnoticeability constraint (see line 5 and 8 in Alg. 1).

According to the $\gamma$-decaying heuristic theory [34], given two target nodes, their $h$-hop enclosing subgraphs are very informative for link prediction, which means the perturbations are likely feasible when they lead to changes on its subgraph; $h = 1$ or 2 is typically sufficient for accurate link prediction. Hence, we only have to inspect the optimal perturbations that can make changes to the $h$-hop subgraph. In other words, at least one end of the edge added/deleted should be included in the enclosing subgraph to be a possible feasible perturbation. Precisely, one end of the perturbed edge should be included in its set of $(h - 1)$-hop nodes, denoted as $(\Gamma^{h-1}(x) \cup \Gamma^{h-1}(y))$ (see Alg. 1).

Let’s see an example with $h = 2$ as shown in Fig. 3. $x, y$ are the target nodes and the link in between is requested for link existence prediction. As shown in Fig. 3, only when one end of edge added/deleted (the blue lines) is included in their 1-hop node set (1-hop neighbours of $x$ or $y$), the perturbations can lead to changes to the subgraph. However, the ends of the perturbed edges (the red lines) are not in their 1-hop node set (not 1-hop neighbours of $x$ or $y$), could not change the subgraph.

The search time complexity in Alg. 1 would be $O(|V_t|\times(|\Gamma^{h-1}(x)|+|\Gamma^{h-1}(y)|))$, where $|V_t|$ is the number of nodes that the adversary is able to manipulate and it can reach $N$ as the capability of the adversary increases. With rapidly growing volumes of data, the size of the graph ($N$) is typically very large; for example, there were

### Algorithm 1: Greedy Graph Structure Perturbation (GGSP)

1. **Input:** Graph $G^{(t)}(\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$, $h$-hop subgraph $c_{xy}^{(t)}$.
2. $S_{struct} \leftarrow \emptyset$.
3. for $\forall u \in \Gamma^{h-1}(x) \cup \Gamma^{h-1}(y)$; do
   4. for $\forall v \in V_t$; do
      5. if $e(u, v) = 1$ and $\Lambda(G^{(t)}, G'^{(t)} - e(u, v)) < 0.004$ then
         6. $G' \leftarrow G^{(t)} - e(u, v)$;
         7. $S_{struct} \leftarrow S_{struct} \cup \{G'\}$;
      8. else if $e(u, v) = 0$ and $\Lambda(G^{(t)}, G'^{(t)} + e(u, v)) < 0.004$ then
         9. $G' \leftarrow G^{(t)} + e(u, v)$;
         10. $S_{struct} \leftarrow S_{struct} \cup \{G'\}$;
   11. end if
   12. end for
5. end for
14. Return: $S_{struct}$
Algorithm 2 Optimized Graph Structure Perturbation (OGSP)

1: **Input:** Graph $G^{(t)}(V^{(t)}, E^{(t)})$, $h$-hop subgraph $G_{xy}^{(t)}$.
2: $S_{\text{struct}} \leftarrow \emptyset$.
3: if $F(A_{xy}^{(0)}, v_{xy}^{(0)}) = 1$ then
4: // decrease common neighbours
5: for $\{v, o \in V_x \cap \{1^h(x) \cap 1^h(y)\} \wedge \{e(u, v) = 1\}$ do
6: if $\Lambda(G^{(t)}, e(u, v)) < 0.004$ then
7: $G' \leftarrow G^{(t)} + e(u, v)$;
8: $S_{\text{struct}} \leftarrow S_{\text{struct}} \cup \{G'\}$;
9: end if
10: end for
11: // increase common neighbours
12: else
13: for $\forall u \in V_x \cap \{1^h(x) \cap 1^h(y)\} \wedge \{e(u, v) = 1\}$ do
14: if $\Lambda(G^{(t)}, e(u, v)) < 0.004$ then
15: $G' \leftarrow G^{(t)} + e(u, v)$;
16: $S_{\text{struct}} \leftarrow S_{\text{struct}} \cup \{G'\}$;
17: end if
18: end for
19: for $\forall v \in V_y \cap \{1^h(y) \cap 1^h(x)\} \wedge \{e(u, v) = 1\} \wedge (u, v) \neq (x, y)$ do
20: if $\Lambda(G^{(t)}, e(u, v)) < 0.004$ then
21: $G' \leftarrow G^{(t)} + e(u, v)$;
22: $S_{\text{struct}} \leftarrow S_{\text{struct}} \cup \{G'\}$;
23: end if
24: end for
25: for $\forall v \in V_x \cap \{1^h(x) \cap 1^h(y)\}$, $\forall i \in 1^{h-1}(x)$, and $\forall j \in 1^{h-1}(y)$ do
26: if $\Lambda(G^{(t)}, e(i, v) + e(v, j) < 0.004$ then
27: $G' \leftarrow G^{(t)} + e(i, v) + e(v, j)$;
28: $S_{\text{struct}} \leftarrow S_{\text{struct}} \cup \{G'\}$;
29: end if
30: end for
31: end if
32: Return: $S_{\text{struct}}$

Even we only consider the search space from its $h$-hop subgraph, the perturbation search time cost is still very high. Can we further improve our attack efficiency? The answer is affirmative.

4.2 Optimized Graph Structure Perturbation

Inspired by the intuition of link formation mechanism — the more common neighbours two target nodes have, the more likely they are connected — we construct $S_{\text{struct}}$ based on the common neighbours that the target nodes share. In this context, a common neighbour is defined as the intersectional neighbours of the two target nodes within their $h$-hop subgraphs.

We consider two different kinds of attacks when constructing $S_{\text{struct}}$. On the one hand, to force a positive link to be a negative link (link hidden), we delete edges to reduce the number of common neighbours in the subgraph (see line 5-10 in Alg. 2). On the other hand, to encourage a negative link to become a positive link, we add edges to force more nodes to become the common neighbours of the target nodes. Precisely, we first consider the nodes in the $h$-hop subgraph but are not common neighbours. For these nodes, we add one edge for each perturbation under the unnoticeability constraint (see line 13-20 in Alg. 2). Besides, we consider the nodes that are not included in the $h$-hop subgraph. In this case, we add two edges simultaneously and force the nodes, outside of the $h$-hop subgraph, to become the common neighbours of the target nodes. As a result, both kinds of attacks are more focusing on hiding links.

5 EXPERIMENTAL EVALUATION

We have conducted an extensive array of experiments to evaluate our proposed methods, including our greedy algorithm (GGSP) and its efficient variation (OGSP). Our results show that both of them are able to reduce the availability of the SEAL framework significantly, achieving strong performance on various datasets.

More importantly, our experimental results have also shown that our adversarial attacks mounted based on SEAL can be readily transferred to several existing heuristics in the literature. We run the experiments 5 times and then use the average attack success rate (ASR) and the average AUC as our evaluation metrics. To make direct comparisons, we use the same model architectures as SEAL shown in Table 3, where $k$ is set to ensure that 60% of the subgraph nodes are larger than $k$ [34, 35].

Datasets. We have selected four datasets as the benchmarks to evaluate our methods. The datasets statistics are given in Table 2. USAir is a network of US Airlines [5], which average node degree is 12.81. NS is a collaboration network of researchers in network science [24], which average node degree is 3.45. Celegans [32] is a neural network of C.elegans, which average node degree is 14.46 and PB is a network of US political blogs [1], which average node degree is 27.36.

Similar to SEAL, we split the existent links randomly into a positive training set (80%) and testing set (10%). As for negative
sets, we randomly sample an equal number of non-existent links as the negative training set and testing set, respectively. We retrain SEAL for 50 epochs for each dataset, and select the model with the smallest loss on 10% validation data; these pre-trained models are used as our target models to mount attacks.

Note that, we remove the edges between the two target nodes in the enclosing subgraphs while we train graph neural network, as did in [34]. This is because these edges would contain the link existence information, while is not available in the enclosing subgraphs of testing links. As observed, we report the model AUC on clean data directly predicted, including

Within the testing set, we select 10 links with the highest prediction margin, including 5 positive links and 5 negative links, i.e., they clearly are correct predictions (best-set). We also select 10 links with the lowest prediction margin (but still correctly predicted), including 5 positive links and 5 negative links (worst-set). Finally, we select 20 links randomly sampled from the links that are correctly predicted, including 10 positive links and 10 negative links, respectively (random-set). These will serve as the target links for our attack. By default, the average ASR and AUC are reported according to the prediction results of the links randomly selected.

**Table 3: Model architecture of SEAL**

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Graph Convolutions + Tanh</td>
<td>32, 32, 32, 1 channels</td>
</tr>
<tr>
<td>Max-K SortPooling</td>
<td>k</td>
</tr>
<tr>
<td>1-D Convolution + ReLU</td>
<td>16 output channels, filter size 2, step size 2</td>
</tr>
<tr>
<td>1-D Convolution + ReLU</td>
<td>32 output channels, filter size 5, step size 1</td>
</tr>
<tr>
<td>Dense Layers</td>
<td>128 units</td>
</tr>
<tr>
<td>Log-Softmax</td>
<td>2 channels</td>
</tr>
</tbody>
</table>

### 5.1 Attacks on SEAL

We start by analyzing both of our two algorithms, GGSP and OGSP, by inspecting their influences on link prediction performance of SEAL (with full knowledge of the network graph). In Table 4, we report the average ASR and average AUC over 5 runs when performing attacks on SEAL. For each run, we use the random-set as our target links. We can see GGSP achieves very high ASR on NS — 100%, and its AUC degrades to 0.000. Even OGSP has an attack performance degradation on this dataset, it still can decline its model AUC to 0.038.

In Fig. 5, we report how our methods affect different testing sets (best, worst and random). We define the prediction margin as the difference between the ground truth label probability with SEAL and the target label probability. The smaller prediction margin indicates better attack performance (margin less than 0 indicates a successful attack). As observed in Fig. 5, most of the average margins are under 0, represented as ‘star’. best-set is harder to attack as its initial prediction margin are large (typically close to 1). Overall, it still can achieve reasonably good attack performance with our algorithms.

**Table 4: Average attack success rate/Average AUC**

<table>
<thead>
<tr>
<th>Data</th>
<th>GGSP</th>
<th>OGSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>USAir</td>
<td>96.3%/0.050</td>
<td>98%/0.000</td>
</tr>
<tr>
<td>NS</td>
<td>100%/0.000</td>
<td>79%/0.038</td>
</tr>
<tr>
<td>CELEGANS</td>
<td>87.5%/0.133</td>
<td>82%/0.166</td>
</tr>
<tr>
<td>PB</td>
<td>82.5%/0.207</td>
<td>78%/0.294</td>
</tr>
</tbody>
</table>

Furthermore, to inspect and compare the time cost of our two attack algorithms, we also report the average attack time per link across different datasets. As Fig. 4 shows, OGSP is way more efficient than GGSP; it can even be approximately 10× faster on Celegans. Note that our time cost is averaged over the target links, containing 50% positive links and 50% negative links. We suspect that OGSP can achieve even better performance at runtime while the adversary is more focused on hiding links.

**Figure 4: Average attack time per link: GGSP vs. OGSP.**
To analyze the performance of our attack with respect to the adversary capability, we run four sets of experiments when $|V_s|$ are in different settings for each dataset. We set the number of nodes that the adversary is capable of manipulating as 25%, 50%, 75% and 100% of the entire node set, respectively. For each run, the node set $V_s$ is selected randomly. As shown in Fig. 5, as $|V_s|$ becomes larger, GGSP produces better results. As observed, even with only 25% perturbable nodes, our algorithm can still achieve a high ASR.

Table 5: Average ASR (GGSP)

| $|V_s|/N$ | USAir | NS | Celegans | PB |
|---------|-------|----|----------|----|
| 0.25    | 66.3% | 97.5% | 72.5% | 69.9% |
| 0.50    | 91.3% | 100% | 81.3% | 79.3% |
| 0.75    | 92.5% | 100% | 84.3% | 81.2% |
| 1.00    | 96.3% | 100% | 87.5% | 82.5% |

We further report how the target link surrounding structure affects the attack performance. We run three sets of experiments in USAir when the adversary can perturb 25% of the entire node set. The target links are categorized according to their link degrees. As seen in Table 6, the target links with higher degrees achieve lower ASRs, indicating that higher-degree links are harder to attack. The ASR of the target links in the range $[1:30)$ can achieve as high as 90.96%.

Table 6: ASR: Target link surrounding structure complexity

<table>
<thead>
<tr>
<th>Degree range</th>
<th>$[1 : 30)$</th>
<th>$[31 : 90)$</th>
<th>$[90 : \infty)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>#target links</td>
<td>188</td>
<td>93</td>
<td>78</td>
</tr>
<tr>
<td>Attack success rate</td>
<td>90.96%</td>
<td>64.52%</td>
<td>43.58%</td>
</tr>
</tbody>
</table>

Inspecting an adversarial example. Fig. 6 illustrates a real adversarial example mounted for NS. We first randomly select a pair of nodes as the target nodes (grey); the link (dash line) is to be predicted using the pre-trained SEAL. Fig. 6a shows its 1-hop enclosing subgraph; the link is initially predicted as $c = 0$ by SEAL, indicating the link is negative. Fig. 6a shows its 1-hop enclosing subgraph with our attack method. The edge in blue is suggested to be added by our attack. Even just with this edge addition, the link is predicted as positive.

Transferability of attacks. Note that, our overall objective is to offer a comprehensive study on the ability of an “adversary” to manipulate link prediction via adversarial machine learning. For this purpose, we analyze the transferability of the adversarial attack, generated based on SEAL, to existing heuristics, including common neighbors (CN), Jaccard [20], preference attachment (PA) [4], Adam-Adar (AA) [2], resource allocation (RA) [36], Katz index [19], PAGERANK [6], SimRank [17] and WLK [28]. We report the associated average model AUC over 5 runs, including the model AUC tested on clean graph data and model AUC tested on attacked graph data.

Note that, to be comparable, we trained SEAL purely using graph structure features (the node information matrix contains structural node labels only as did in [34]). Then we mount the adversarial attacks using our efficient design — OGSP. As shown in Table 7, the performance of most existing heuristics is even worse than random guessing (the model AUC is less than 0.5), indicating that our attacks can be readily transferred to several existing heuristics in the
versarial training would force the model to assign both clean and eliminate the problem. As often applied in the image domain, ad-
theory and link formation mechanism, it is very hard to completely mental assumptions used for link predictions, i.e., the \( \Upsilon \) link formation mechanism we considered.

neighbour information, which is not consistent with the typical individual node degree of the target node and does not contain any because the heuristic used by preference attachment only considers
attacks failed transfer to preference attachment. We suspect that is perform extremely well for NS. We can observe that the mounted
literature. Noticeably, among all the datasets, the mounted attacks perform extremely well for NS. We can observe that the mounted attacks failed transfer to preference attachment. We suspect that is because the heuristic used by preference attachment only considers individual node degree of the target node and does not contain any neighbour information, which is not consistent with the typical link formation mechanism we considered.

Defense against attacks. The mounted attacks violate the fundamental assumptions used for link predictions, i.e., the \( \Gamma \)-decaying theory and link formation mechanism, it is very hard to completely eliminate the problem. As often applied in the image domain, adversarial training would force the model to assign both clean and adversarial examples to the same output labels [18]. This raises the idea of adding adversarial examples into the training set while we are training the model, which we leave for our future work.

<table>
<thead>
<tr>
<th>DATA</th>
<th>SEAL</th>
<th>CN</th>
<th>Jaccard</th>
<th>PA</th>
<th>AA</th>
<th>RA</th>
<th>Katz</th>
<th>PAGERANK</th>
<th>SimRank</th>
<th>WLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>USAir</td>
<td>Clean Graph</td>
<td>0.967</td>
<td>0.926</td>
<td>0.902</td>
<td>0.843</td>
<td>0.943</td>
<td>0.947</td>
<td>0.910</td>
<td>0.920</td>
<td>0.790</td>
</tr>
<tr>
<td></td>
<td>Attacked Graph</td>
<td>0.002</td>
<td>0.139</td>
<td>0.346</td>
<td>0.827</td>
<td>0.591</td>
<td>0.246</td>
<td>0.780</td>
<td>0.187</td>
<td>0.083</td>
</tr>
<tr>
<td>NS</td>
<td>Clean Graph</td>
<td>0.987</td>
<td>0.934</td>
<td>0.934</td>
<td>0.631</td>
<td>0.934</td>
<td>0.934</td>
<td>0.934</td>
<td>0.934</td>
<td>0.935</td>
</tr>
<tr>
<td></td>
<td>Attacked Graph</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.178</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.100</td>
<td>0.170</td>
</tr>
<tr>
<td>Celegans</td>
<td>Clean Graph</td>
<td>0.872</td>
<td>0.834</td>
<td>0.782</td>
<td>0.745</td>
<td>0.849</td>
<td>0.853</td>
<td>0.849</td>
<td>0.881</td>
<td>0.751</td>
</tr>
<tr>
<td></td>
<td>Attacked Graph</td>
<td>0.233</td>
<td>0.263</td>
<td>0.094</td>
<td>0.703</td>
<td>0.187</td>
<td>0.152</td>
<td>0.300</td>
<td>0.367</td>
<td>0.190</td>
</tr>
<tr>
<td>PB</td>
<td>Clean Graph</td>
<td>0.940</td>
<td>0.908</td>
<td>0.859</td>
<td>0.899</td>
<td>0.913</td>
<td>0.916</td>
<td>0.919</td>
<td>0.934</td>
<td>0.766</td>
</tr>
<tr>
<td></td>
<td>Attacked Graph</td>
<td>0.277</td>
<td>0.564</td>
<td>0.151</td>
<td>0.881</td>
<td>0.510</td>
<td>0.400</td>
<td>0.590</td>
<td>0.603</td>
<td>0.082</td>
</tr>
</tbody>
</table>

Figure 6: Adversarial Example from NS: given two target nodes (in grey) randomly selected from NS, the link in between is un-observed (a) its initial 1-hop subgraph was predicted as ‘non-existent’ by SEAL; (b) its 1-hop subgraph after perturbation (just one edge perturbation — blue line — in this case) was predicted as ‘existed’ by SEAL.

6 RELATED WORK
Adversarial attacks on link prediction: Zhou et al. investigated the problem of mounting attacks on several heuristics for link prediction. In this work, they further categorized the heuristics based on the maximum hop of neighbours needed to calculate the similarity score. For each category, they proposed associated attack approaches via deleting edges only. Chen et al. proposed an iterative gradient attack against graph auto-encoder (GAE)-based link prediction [10]. In contrast, our attacks are mounted based on SEAL under “unnoticeability” constraint and are more general, as they perform well in several link prediction heuristics.

Neural networks for link prediction: Weisfeiler-Lehman Neural Machine (WLNM) proposed by Zhang et al. is the first attempt to employ DNNs for link-prediction tasks[33]. In particular, it learns general graph structure features by encoding enclosing subgraphs of the training links into fixed-size adjacency matrices and trains a fully-connected neural network on the adjacency matrices. As the typical neural networks only accept fixed-size tensor, WLNM can only learn the link local patterns from the subgraphs with fixed-size.

Following this work, Zhang et al. proposed the SEAL framework for link prediction using graph neural network, called SEAL. It has been shown its state-of-the-art prediction performance empirically and theoretically. In particular, a \( \Gamma \)-decaying theory has been developed to unify a wide range of existing heuristics for this task, which proved that local subgraphs are informative for link prediction.

Adversarial attacks on machine learning: Huang et al. categorized attacks in adversarial machine learning as either causative or eva-
with the former poisoning the training dataset and the latter evading the pre-trained model by crafting adversarial examples [16]. Following the terminology of Huang et al., our work focuses on the evasion attacks that aim to create adversarial examples deliberately to mislead the state-of-the-art link prediction framework — SEAL, yet still preserving its unnoticeability.
Adversarial attacks have been shown their capability of significantly degrading the performance of deep learning models in various application domains, e.g., image [8], audio [9] and malware detection [15]. These applications consider the data instances to be independent and classify an instance based on features extracted from only that instance. This directly enables evasion techniques such as gradient descent/ascent directly in the feature space. For complex network graph as we considered in this paper, data instances (e.g., links) are interrelated and treated as non i.i.d data.

**Adversarial attacks against graph data.** Works on adversarial attacks against complex network graph for GNN-based graph learning tasks are exceedingly rare, in contrast to other application domains, not to mention adversarial attacks particularly for link prediction. Very recently, Zugner et al. first proposed the work on adversarial attacks against network graph particularly for node classification tasks [37]. As for the classification model, they focused on a transductive learning setting. Zugner et al.’s work is inherently related to the causative/poisoning attacks [16]. Following this work, Zugner et al. investigated the problem of training-time attacks on the overall performance of node classification via the principle of meta learning [38].

Along with the work of Zugner et al., Dai et al. proposed to employ reinforcement learning approach to attack both in the graph-level learning tasks and the node-level classification tasks [15]. Unlike Zugner et al.’s causative/poisoning attacks, Dai et al.’s work focused on the evasion attacks particularly against node/graph classification tasks. Except for Dai et al.’s evasion attacks, Wang et al. studied evasion attacks against collective classification methods via manipulating the graph structure [30]. In contrast, our work focuses on adversarial GNN-based link prediction, which deals with coupled perturbation variables and non i.i.d graph data. We also evaluate the different capabilities of the adversary with various perturbable sets, and analyze its transferability to existing link prediction heuristics.

### 7 CONCLUDING REMARKS

We have shown that adversarial attacks on graphs can break the SEAL framework that uses GNNs for link prediction. These attacks can often achieve significantly higher attack success rates, and can degrade the model performance by a substantial margin, making the prediction error even worse than random guessing. By incorporating the 1-decaying theory and the link formation mechanism, our attacks can be generated efficiently and effectively. The prediction performance is consistently exacerbated, even when the adversary’s capability is restricted to only parts of the network graph. Based on our extensive experiments, we show that the attacks mounted based on GNN-based link prediction can be transferred to several existing heuristics with our design.

### REFERENCES


Networks 25, 3 (2003), 211–230.


[8] Nicholas Carlini and David Wagner. 2017. Towards Evaluating the Robustness of


[23] Yacin Nadji, Manos Antonakakis, Roberto Perdisci, and Weke Lee. 2013. Con-
ected Colors: Unveiling the Structure of Criminal Networks. In Proc. Interna-
tional Workshop on Recent Advances in Intrusion Detection. Springer.


[28] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick Mc-

[29] Binghui Wang and Neil Zhenqiang Gong. 2019. Attacking Graph-based Classifi-


