No Pain, No Gain: Improving Graph Learning via Adversarial Link Inference

Sheng Guan Hanchao Ma Yinghui Wu
Case Western Reserve University
{sxg967,hxm382,yxw1650}@case.edu

***Abstract*—Emerging graph models based on message passing such as Graph Neural Networks (GNNs) suffer from unreliable links due to dirty data or adversarial attacks at training time. This paper introduces an adversarially learned framework to automatically infer critical links that can mitigate the impact from adversarial manipulations. Our key idea is to characterize critical links with transition likelihoods that can "reverse" the impact of adversarial links to the model performance. We introduce two approaches to improve graph learning against unreliable links, favoring interpretability and cost, respectively: (1) The *bounded manipulation* strategy configures and explicitly refines a small number of links to improve the targeted GNN model with high interpretability; and (2) The *masked attention* strategy combines transition likelihood with a shared node attention mechanism to learn better models directly from unreliable data without link manipulation. Our methods do not require prior knowledge on attack models or expensive link prediction, and apply to mainstream GNN models with feasible implementation. Using real-world benchmark graphs and classification task, we show that our framework effectively improves GNN-based learning with significant performance gains (*e.g.*, TBF on TBF for TBF) by changing a small amount of links (TBF in TBF), and can provide interpretable results.**

I. Introduction

Graph representation models such as Graph neural networks (GNNs) [6] have been applied to learn and transform graph data to their proper vector representations (*e.g.,* node embeddings). Representative GNNs adopt a label propagation architecture to learn discriminative node embeddings with multiple transformation layers. In each GNN layer, the embedding of a node v is updated by aggregating its own features and those from its neighbors $N(v)$ (a graph convolutional layer). For example, convolutional graph neural network (GCN) [12] generalizes the convolution from grid data to graphs by stacking multiple graph convolutional layers.

Existing GNN-based learning often assume complete and accurate graph data and heavily rely on their inherent topology. Nevertheless, real-world graphs are often unreliable. (1) The neighbors of a node is often partially observable due to missing or incorrect links [20], [23]. (2) An open challenge in GNN design is the impact from the skewed label distribution of the neighbors. For example, a node can be incorrectly labeled simply due to that most of its neighbors have a same label [18]. (3) Malicious edge modifications via graph structural attacks [3], [40] also lead to low-quality GNNs [40].

Can we properly infer and manipulate important links to improve GNN*-based graph learning*? The refined links can reveal and recover useful neighbors to targeted GNNs, thus mitigate the negative impact of unreliable links. Consider the following examples taken from real-world graphs and benchmark learning tasks.

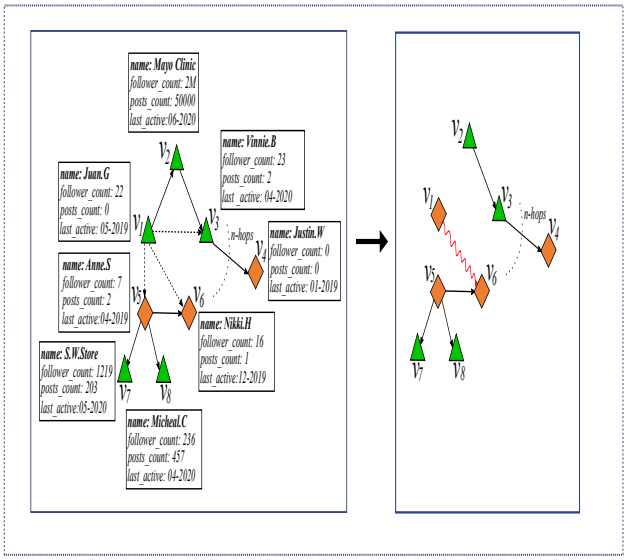

Fig. 1: Identify long-range relations improve GNN-based node classification.

Example 1: Add another example (tweet?) to show impact of adversarial attack. Threat model/cost of attackers/defenders.

Fig. 1(b) illustrates a fraction of a protein-protein interaction network with two disconnected components. There are two nodes v_4 and v_7, representing different types of proteins *FTH* and *FTMT*, respectively. The two proteins reside in cells from different human tissues with quite different direct neighbors, and are not connected via interaction relations. Nevertheless, they have a similar function of iron binding, determined by their common features from gene sequences.

A classifier can be induced from GNNs to infer the unknown functions of v_4 given the learned node embeddings. However, the relevant protein node v_7 cannot be exploited by GNN-based learning to correctly infer the label due to the missing link. Augmenting the missing link between the disconnected v_4 and v_7 helps to perform a correct classification for v_4. □

We are interested in identifying only a small amount of links and manipulation techniques to *improve* GNN-based learning without incurring much cost. There are several options.

(1) Apply link prediction [11], [34] to infer the missing edges from observed distribution of relations as much as possible. However, link prediction targets at data completion instead of identifying critical links that benefit graph representation learning, leaving alone the inference cost over the entire graphs. Indeed, not all the inferred links contribute to GNN-based learning. For example, add more. Moreover, missing links that represent useful "long-range" dependencies between relevant nodes (*e.g.,* the link between v_4 and v_7 in G_2) may not be captured by link prediction.

(2) Optimize GNN architectures. For example, one may add more convolutional layers to GNN such that "long-range" nodes have a chance to contribute their information via message passing. However, when stacking too many layers, the embedding results of each node quickly become indistinguishable, leading to poor classification results [13].

One may also enable "jumping" across multiple stacked layers [10], [30], or assigning weights ("importance") to neighbors via attention mechanism (*e.g.,* graph attention models GAT [24]). These models relax strict neighborhood to important fraction or non-local nodes, but either rely on input topology, or require manually specified pattern (*e.g.,* meta-paths) to connect non-local nodes. For example, the relevant nodes v_4 and v_7 in Fig. 1(b) cannot be linked via meta-paths.

Adversarial defense [35] is to improve model performance by training models against known adversarial samples in a min-max game (minimize loss over adversarial samples that maximizes a closeness measure to true data). Adversarial training requires prior generation knowledge of adversarial samples. The attack models and the "ground truth" graphs (such as knowledge graphs [20]) may not known in practice. Moreover, the negative impact from missing links may not always conform to a particular attack model.

Contribution. This work introduces an adversarially learned framework to improve GNN-based learning against unreliable links. Our key idea is to "learn from attacks": we first learn a generative model $\mathcal{G}$ for adversarial links over an augmented counterpart of input graphs with a goal to degrade the performance of targeted GNNs, and learn to manipulate a small amount of links (not simply "reversing" the adversarial manipulation) to mitigate the negative impact.

A generative model for adversarial links. We characterize unreliable (including missing or adversarially modified) links with a generative model $\mathcal{G}$ that simulates malicious training time link manipulation (edge insertions and deletions) to a targeted GNN model $\mathcal{M}$. Given a graph G, $\mathcal{G}$ is learned by "attacking" an augmented counterpart of G with a goal to reduce the accuracy of $\mathcal{M}$. Unlike adversarial defense, $\mathcal{G}$ can be learned without prior knowledge of attack models or graphs, and can be specialized to several existing attack models.

Adversarial Link Inference. Given the generative link model $\mathcal{G}$, we introduce an adversarial link inference model Ł to identify critical links that are likely to be attacked. By ensuring that Ł suggests links with high likelihood to be reconnected or disconnected in response to injected adversarial edge insertions or deletions, the critical links are used to guide applicable manipulations or better learning strategy. We introduce efficient solution based on supervised random walk to recommend critical links based on their importance quantified by transition probability.

Model Improvement. The critical links are not directly usable to improve GNN models. We introduce two solutions, each addresses a challenging issue as a standalone method.

(a) The first strategy, called *bounded manipulation*, discovers and explicitly modifies a small set of links to satisfy the (un)reachability requirement suggested by critical links. We formulate this as an optimization problem called reachability preserving min-cut problem, and show that the problem is hard to approximate. We present an algorithm to produce manipulations, and show that it ensures approximation ratio for special cases such as "insertion only" or "deletion only".

(b) When link manipulations are expensive (*e.g.,* modifying social structures or cyber network communication), we provide a second solution called *adversarial masked attention*. This end-to-end solution combines a masked attention mechanism for node updating and adversarially inferred links to a single end-to-end representation learning process. This strategy enables us to directly learn high quality graph representations against unreliable links, without explicitly manipulating links.

Ablation Analysis. Our adversarial link augmentation framework applies to improve representative GNNs with direct neighborhood aggregation (GCN) and directed attention coefficients (GAT). Using real-world graphs and benchmark tasks, we provide both empirical evidence and experimental analysis to verify the effectiveness of our models and algorithms. For example, the bounded inference can enhance GCN with a gain of on averaged **TBF**% on **TBF** for node classification over benchmark knowledge graphs when **TBF** edges are "polluted". The gain on model accuracy can be achieved by manipulating up to **TBF** links, and is not sensitive to the attack models and

the amount of adversarial links. Our framework also suggests intuitive interpretation to the result, as verified by our case analysis.

Related Work. We categorize the related work as follows.

Graph Neural Networks. Extending deep neural networks, GNNs [6] encode graph features to a proper representation as node embeddings, which can be used for downstream tasks such as node classification. A key component in GNNs is the encoder function that aggregate features from the local (direct) neighbors of a node [8], [12], [24] to propagate labels following links. Notable examples include GCN [12], which applies a localized first-order approximation of spectral graph convolutions where the layer serves as an operation of graph convolution (essentially a special form of Laplacian smoothing); and GAT [24], which supports self-attention mechanism to assign different weights to neighbors. It is observed that GNNs often limit to shallow architectures with two or three layers, as stacking multiple layers reduces its discriminative power of node embedding [13], [21]. This in turn suggests a limited exploitation of long-range relations, especially with presence of missing and adversarial links.

Rather than designing yet-another GNN model, our work introduces a general approach that can best improve GNN-based learning and its downstream tasks against unreliable links. Our adversarially learned scheme adapts to improve GNNs against graphs with missing or adversarial links at affordable cost. These are not discussed in GNN-based learning.

Link Prediction. Link prediction has been extensively studied (see [15] for a survey). For example, missing links can be asserted by similarity-based methods [34], [38], random walks [2], [33], or maximum likelihood methods [31]. The goal of these methods are to predict missing links based on their likelihood of existence given the observed graphs. In contrast, rather than making graph data more complete [15], we identify and manipulate a bounded amount of links to improve GNN-learning. For example, we intentionally "bias" the inference to only the important links that contribute to improving GNN learning. Such links direct to relevant neighbors but may not necessarily belong to a missing fraction of the graph. To our best knowledge, link augmentation designed with a goal to improve GNN models has not been studied by prior graph-based learning techniques.

Non-local Learning. Recent study verifies an emerging interest in enabling non-local dependency in deep neural networks [27]. [10], [30] proposed to carefully combine different representations learned from different distances when updating the source node representation, not necessarily from their immediate (first-hop) neighbors. [29] proposed to create a virtual node that aggregates the representation of nodes along a meta-path together as long-range dependencies, yet requires specific meta paths to be provided. Our work differs from these work in the following. (1) The adversarial link augmentation does not require meta-paths. Instead, it is guided by adversarial examples that can be effectively learned over targeted GNN and input graphs. (2) Our models automatically differentiate useful and irrelevant nodes via adversarial learning and finer-grained refinement process, instead of averaging feature vectors from all the nodes within certain hops [29].

II. Background

A. Graphs and Graph Representations

Attributed Graphs. We consider an attributed graph $G = (V, E)$ with a set of nodes V and a set of edges $E \subseteq V \times V$. Each node $v \in V$ carries a vector $(v.A_1, \ldots v.A_n)$ defined on n attributes, where $v.A_i = a_i$ ($i \in [1, n]$) denotes that the attribute A_i of v has value a_i.

We shall use the following notation. (1) The *r-hop neighbor* of a node v (denoted as $N_r(v)$) refers to the set of nodes in V that can reach v or can be reached from v via a path with length no more than r. The direct neighbors of v $N(v)$ refers to the set $\{v' | (v, v') \in E$ or $(v', v) \in E\}$. (2) Given a set of nodes $V' \subseteq V$, a *subgraph induced by V'* of G is a subgraph with nodes V' and the edges among them.

We define a *link* in G as a node pair $(v, v') \in V \times V$. A *manipulation* of a link (v, v') in G is either (1) a deletion (denoted as $((v, v'), '-')$ of an edge $(v, v') \in E$, or (2) an insertion $(((v, v'), '+'))$ of a new link $(v, v') \notin E$. The manipulations ΔG updates G to a graph $G' = G \oplus \Delta G$.

Representing Graphs. A *feature representation* of a graph $G = (V, E)$ is a pair (X_G, A_G), where (1) X_G is a feature matrix ($X_G \in \mathbb{R}^{|V| \times d}$) that records a feature vector $x_v \in \mathbb{R}^d$ for each node $v \in V$; and (2) A_G is the adjacency matrix of G. In practice, one may apply encoding functions to convert node attributes to embedding space $\mathbb{R}^d$, such as word embedding [5]. For nodes with different types that may not be uniformly represented, one may unify node feature representation by concatenating categorical types as one-hot feature [9], or apply meta-paths [37] to induce and represent similar nodes.

Node classification. Given a graph $G = (V, E)$, each node v is assumed to have a class label $L(v)$. A node is *labeled* if its class label is known; otherwise it is *unlabeled*. Given a set of *training nodes* $V_{\mathcal{T}} \subseteq V$ that contains labeled nodes, the problem is to learn a model that accesses G to infer the class labels of a set of unlabeled *test nodes* from $V \setminus V_{\mathcal{T}}$.

We shall make case for node classification as a common graph learning task. Our framework applies to other GNN-based graph learning tasks such as anomaly detection.

B. Graph Representation Learning: A Revisit

Graph Neural Networks (GNNs) [28] are applied for graph representation learning. The goal is to transform input graph feature representation (X_G, A_G) to proper representation to support downstream tasks such as node classification.

A GNN with n layers over graph $G = (V, E)$ adopts a neighborhood aggregation architecture to distill information about the neighborhood of a node v to its embedding as follows. The ℓ-th layer ($\ell = 1, \ldots, L$) takes the embedding $\vec{h}_v^{\ell-1}$ of each node $v \in V$ as input ($\vec{h}_v^0$ refers to the input

Symbol	Notation
$G = (V, E)$	Attributed graph with nodes V and edges E
$V_{\mathcal{T}}$	Labeled training nodes
(X_G, A_G)	Feature representation of G: (X_G: feature matrix; A_G: adjacency matrix)
$\vec{h}^{\ell}(\mathbf{x}_v)$	the embedding of node v at ℓ-th layer
$H^{(n)}$	graph embedding matrix at layer n
$\mathcal{M}$	targeted GNN model
$\mathcal{G}$	Generative model for adversarial links
$G' = (V, E')$	Augmented counterpart of G for link inference
Ł	Biased Link inference model

TABLE I: Table of Notations

feature $x_v \in X_G$). For each node v, it updates the embedding $\vec{h}_v^{\ell}$ by aggregating from the embeddings of its neighbors $\mathcal{N}$(v). As the updated $\vec{h}_i^{\ell}$ becomes the input to the $(\ell + 1)$-th layer, the aggregation procedure propagates from layer 1 to n in a principled manner. As such, a node v utilizes the information up to its n-hop neighbors in learning its own representation. The principled update process contains two steps [19]:

(1) Transform neighborhood embeddings:

$$\vec{h}_{\mathcal{N}(v)}^{\ell} \leftarrow \text{TRAN}(\text{AGG}_{\mathcal{N}(v)}(\{(\vec{h}_{v'}^{\ell-1}, w_{v,v'}^{\ell}) \,|\, v' \in \mathcal{N}(v)\}))$$

where (1) AGG is an aggregation *e.g.,* a max-polling operation (reducing dimensions of input matrix with *e.g.,* max filter over its non-overlap sub-regions), and is performed with learnable weights $w_{v,v'}^{\ell}$ from a weight matrix W^{ℓ} shared by all the nodes; and (2) TRAN is a transformation function over AGG results, *e.g.,* a multiplication between W^{ℓ} and AGG results followed by a nonlinear activation function $\sigma(\cdot)$.

(2) Combine neighborhood embeddings for local update:

$$\vec{h}_i^{\ell} \leftarrow \text{COMBINE}\left(\vec{h}_i^{\ell-1}, \vec{h}_{\mathcal{N}(i)}^{\ell}\right)$$

where COMBINE is a function that merges the TRANS result with the representation of node i, *e.g.,* concatenation.

GNNs can be specified by different layer-wise propagation rules. Among the mainstream GNNs (see [28] for a survey) are: (1) GCN (as aforementioned), which stacks multiple graph convolutional layers; (2) GraphSage [8] permits general aggregation functions over sampled fixed-size neighborhood of each node; and (3) GAT [24], which integrates self-attention mechanism to learn a weight coefficient for each neighbor in order to average features from neighborhood of nodes.

Unreliable links: A pain point. GNNs assume complete data and often follow the inherent topology of input graph G for propagation and updates. It is verified that the quality of GNNs can be significantly degraded when links (which specify $N(v)$) are missing or unavailable [25], [40]. Moreover, it is often challenging but desirable to clarify which nodes and links from $N(v)$ are "responsible" for the degraded performance.

In response to these challenges, we next introduce our adversarial link augmentation framework.

III. Framework Overview

We introduce the cornerstone models in our framework. Each model tackles a challenge as a standalone approach.

Algorithm ALGen

Input: Augmented graph G', perturbation budget Δ, training iterations T, trainin

Output:

1.
2.
3.

Fig. 2: Algorithm ALGen

Adversarial Links Generative Model ($\mathcal{G}$). While there is no prior knowledge on attack models or complete graphs, we advocate to learn a generative model to characterize important links for GNN-learning. Given input graph G and targeted GNN model $\mathcal{M}$, the generative model $\mathcal{G}$ approximates a set of malicious manipulations ΔE^a of adversarial links E^a applicable to an augmented counterpart G' of G ($\hat{G} = G' \oplus \Delta E^a$), such that ΔE^a leads to a small amount of change to G' ($\hat{G}$ and G' are similar), and the accuracy of $\mathcal{M}^a$ learned from $\hat{G}$ is maximally reduced, compared with the model $\mathcal{M}'$ learned from G' (Section IV).

Link Inference Model (Ł). Given the generative model $\mathcal{G}$ and adversarial links E^a, the link inference model Ł learns and infers from E^a a set of important links E^p.

such that the accuracy of the model $\mathcal{M}^c$ learned from $G \oplus \Delta E^c$ can be maximally improved, compared with its counterpart $\mathcal{M}$ learned from graph G (Section V).

Model Improvement. The learning phase of adversarial link augmentation optimizes $\mathcal{G}$ and Ł respectively over input graphs. In the inference phase, it consults $\mathcal{G}$ and Ł to infer a small amount of manipulation to improve GNN-based learning.

We only consider link manipulations in this work. Moreover, the manipulations are applied in an "non-destructive" manner: we perform auxiliary modifications to improve learning process without actual graph modifications.

IV. Generative Model for Adversarial Links

The generative model $\mathcal{G}$ aims to identify potential adversarial links E^a applied to input graph G [**Sheng**]:G' (augmented counterpart of G) that reduce the performance of a target GNN $\mathcal{M}$ learned from G [**Sheng**]:$\hat{G}$.

Characterizing adversarial links. Following adversarial attacks [40], we assume that $G = (V, E)$ is obtained by applying adversarial manipulations ΔE^a to its high quality counterpart G^* ($G^* = G'$[**Sheng**] : $G \oplus \Delta E^a$). If G^* is known, $\mathcal{G}$ can be learned from the known distribution of the symmetric difference between the edges from G^* and G, respectively. Nevertheless, these adversarial links are not known a priori as G^* is not available.

Our key idea is to first "augment" G to a counterpart G' with link insertions that make possible adversarial deletions applicable (G' may contain links that are not useful for learning $\mathcal{M}$). We then learn $\mathcal{G}$ by "attacking" G', treating its adjacency matrix $A_{G'}$ as a hyperparameter.

We define the augmented graph G' as [**Sheng**]:an augmented graph based on G by adding a set of "active" edges, where we define an edge (u, v) to be active if (1) at least one of these two nodes lies in the the the *training node* set with labels, *e.g.,* $v \in V_{\mathcal{T}}$; (2) $(u, v) \notin E$; and (3) the closeness measurement between u and v is larger than a deterministic threshold of ϵ_v.

[**Sheng**]:There are two facts in the GNN-learning: (1) graph suffers from unreliable links, *e.g.,* missing links, and (2) neighboring nodes in a graph are supposed to be similar to each other [18]. Naturally, we need to complete missing fraction of graph G first by augmenting edges that connect two nodes with similar vector representations to facilitate GNN-learning. In this way, nodes can aggregate information from their similar neighbors to learn discriminative node embeddings via augmented links.

[**Sheng**]:In learning $\mathcal{M}$, the misclassification cost is defined only on labeled nodes $V_{\mathcal{T}}$. What is more, most GNN-based methods aggregate and update node embeddings of v starting from its direct neighbors $N(v)$, and then propagate along the edges to further consider nodes at the far-away hops. The embedding results of node v would be dominant by node embeddings of $N(v)$. Hence, We pay attention to "augmenting" the fraction of graph G that is related to the labeled nodes $V_{\mathcal{T}}$. For any node pair (v, v'), if there is at least one end node *e.g.,* $v \in V_{\mathcal{T}}$ and their closeness measurement, *e.g.,* semantic closeness, $s(v, v') > \epsilon_v$, we augment this edge (v, v') to G' if $(v, v') \notin E$. In the experiment, we specify one way to determine the threshold ϵ_v. The similarity threshold ϵ_v for each node $v \in V_{\mathcal{T}}$ is defined as the maximum closeness value $\max_{v' \in N(v)} s(v, v')$, respectively. If (v, v') involves two thresholds ϵ_v and $\epsilon_{v'}$, the smaller one will be chosen as the threshold for this edge.

[**Sheng**]:A complete graph that connects all the nodes in G with each other would not be applied as G' directly since it increases augmented edges exponentially and limits the scalability of the generative model for large graphs. The generative model needs more computation time to manipulate additional edges if G' is complete. These additional edges indeedly have minor impact on the GNN-learning task since they are not directly connected with $V_{\mathcal{T}}$.

Learning Generative Model. Given an augmented graph G' and targeted model $\mathcal{M}$, we define the learning problem as: Give formal definition of the learning problem; give the learning objective; define parameters as needed.

[**Sheng**]:The quality of node embeddings can be measured by the loss $\mathcal{L}(Z_{\hat{G}}, A_{\hat{G}})$ of the model $\mathcal{M}$ under attack ($\hat{G} = G' \oplus \Delta E^a$). The lower loss represents the higher quality. $Z_{\hat{G}} \in \mathbb{R}^{N \times K}$ is the embedding matrix containing the embeddings of all nodes.

[**Sheng**]:The attacker aims to decrease the generalization performance of the targeted model $\mathcal{M}$ by minimizing the loss of $\mathcal{L}_{\text{atk}}(Z_{\hat{G}}, A_{\hat{G}})$ (maximizing generalization loss). Given an augmented graph G' and targeted model $\mathcal{M}$, We can formalize the learning problem as the following bi-level optimization problem:

$$\begin{aligned} &\min_{\widehat{G}=G' \oplus \Delta E^a} \quad \mathcal{L}_{\text{atk}}\left(Z^*_{\hat{G}}, A_{\hat{G}}\right), \\ &s.t. \quad Z^* = \arg\min_{Z_{\hat{G}}} \mathcal{L}_{\text{train}}(Z_{\hat{G}}, A_{\hat{G}}) \end{aligned}$$

Use 3-4 sentences to explain the intuition of the learning objective.

[**Sheng**]:We cannot directly optimize the $\mathcal{L}_{\text{atk}}$ since the labels of the test data is unknown to the attacker. We choose to set $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}}$, that is to maximize the loss on the labeled training nodes $V_{\mathcal{T}}$. We argue that if a model has a high training error, the generalization performance of this model would be poor, which fulfills the need of the attacker. In the semi-supervised node classification task, if we feed the node embedding matrix $Z^*_{\hat{G}}$ to a softmax layer, the output $Z'_{\hat{G}}$ denotes probability of assigning node v to class c. $\mathcal{L}_{\text{train}} = -\sum_{v \in V_{\mathcal{T}}} Y_v \ln Z'_{\hat{G}}$, Y_v is the ground truth of training node $v \in V_{\mathcal{T}}$.

[**Sheng**]: The essential idea to perform graph structure poisoning attack is to use meta-gradients to solve the bi-level optimization problem. Here, to learn a generative model, we treat the graph adjacency matrix as a hyper-parameter to optimize during the training-time attacks. The meta-gradients (*e.g.,* gradients w.r.t adjacency matrix update) can be used to indicate how the attacker loss $\mathcal{L}_{\text{atk}}$ after training will change because of perturbations on the graph structure. We denote this gradients that is related to graph perturbation as ∇^{meta}_G. ∇^{meta}_G can be computed via backpropagation through the learning phase of $\mathcal{L}_{\text{train}}$. When we get the $\mathcal{L}_{\text{atk}}$ after T training steps, the meta-gradient can be computed by unrolling the training procedure. Once we have the meta-gradient ∇^{meta}_G, we define a score function $S : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that assigns every position (u, v) in the adjacency matrix A a numerical score indicating the estimated impact on the attacker objective $\mathcal{L}_{\text{atk}}$ if this edge (u, v) is perturbed (*e.g.,* removing edge if entry (u, v) is 1). We follow a greedy manner that always picks the perturbation with the highest score one at a time. If this perturbation suggests "edge removal", the chosen edge will be removed from G' and marked with a negative flag and vice versa. After Δ modification budget is consumed, all the selected edges by this greedy manner form as manipulations ΔE^a jointly.

We present the learning algorithm ALGen in Fig. 2. .

V. Adversarial Link Augmentation

Our next goal is to compute a small set of link manipulations ΔE^c that can mitigate the negative impact of E^a and maximize the quality of GNN model $\mathcal{M}$ learned from $G \oplus \Delta E^c$. A straightforward method may simply sets ΔE^c to reverse ΔE^a: add (resp. remove) edges E^c that are removed (resp. injected) by ΔE^a. Nevertheless, this limits the quality of GNN models up to those learned from G', leaving alone that some manipulation may not be applicable to G (*e.g.,* deletion of an edge in G' that is not in G).

Instead, we derive a link inference model Ł to discover critical links E^p that are affected by adversarial manipulations

(Section V-A). The critical links E^p are further used to guide, rather than be directly manipulated, the inference of a small set of applicable manipulations E^c in G, by two link augmentation strategies (Sections V-B and V-C).

A. Biased Link Inference

We introduce *biased link inference* model Ł to infer critical links E^p. Our key idea is to first assign transition probabilities to edges in G' with a bias to "reverse" the connectivity of links that are affected by E^a, and infers links E^p that should be connected or disconnected. These links suggests how G should be manipulated to improve message passing of GNNs.

Link inference model (Ł). Given adversarial manipulations ΔE^a, we induce a set of critical links E^p to characterize the desirable manipulations. It contains two sets of links, which are defined as follows:

- positive links $I^+ = \{(v, v') | ((v, v'), -) \in \Delta E^a\}$: the edges (v, v') removed by adversarial manipulation;
- negative links $I^- = \{(v, v') | ((v, v'), +) \in \Delta E^a\}$: the links (v, v') that are injected by adversarial manipulation.

Specifically, $I^+(v)$ (resp. $I^-(v)$) refers to the set $\{v' | (v, v') \in I^+\}$ (resp. $\{v' | (v, v') \in I^-\}$), *i.e.,* a node that *should* be considered as a useful neighbor (resp. *should not* be connected due to malicious manipulation).

Learning objective. Our key idea is to require the biased link inference model Ł to assigns a *strength* $\alpha_{vv'}$ for each link (v, v') in E^c, such that a random walk from v following the strength is more likely (resp. less likely) to visit the nodes in $I^+(v)$ (resp. $I^-(v)$). The strength naturally suggests important links that should be "connected" or "broken" to mitigate the negative impact from adversarial manipulation.

We define Ł as a function $f_\omega(h(v), h(v'))$ over a link (v, v'), which is parameterized by a learnable vector ω. We solve the following optimization problem to learn Ł.

$$\min_{\omega} \mathcal{F}(\omega) = ||\omega||^2 \; s.t. \; \forall v \text{ from } E^a \text{ and } \forall v^- \in I^-(v),$$
$$\forall \; v^+ \in I^+(v) : p_{vv^-} < p_{vv^+}$$

where p refers to the vector of PageRank scores determined by random walks following edge strengths. p_{vv^-} refers to the PageRank score of node v' from v; p_{vv^+} is defined similarly. That is, we learn a parameter vector ω (and Ł as the function $f_\omega(\cdot)$) such that any PageRank score of nodes in $I^+(v)$ is greater than any score of nodes in $I^-(v)$, for any node v involved in adversarial manipulations ΔE^a.

Algorithm. We approach biased link inference by enhancing *supervised random walk* (SRW) [2] with adversarial manipulations, and introduce an algorithm, denoted as DetCritical (Fig. 3). The algorithm learns a mapping f_ω from a pair $(h(v), h(v'))$ of two node features to a corresponding random walk transition probability as the strength value $\alpha_{vv'}$ from v to v'. Taking all links in E^a, we encode the output as a stochastic transition matrix $M \in \mathbb{R}^{|V_D| \times |V_D|}$ with all $|V_D|$ training nodes involved in E^a by default ($V_D \subseteq V_T$ in G).

Algorithm DetCritical has the following steps.

Algorithm DetCritical
Input: Adversarial manipulations ΔE^a, graph G, integer b;
Output: a set of critical links E^p;
1. set E^p:=$\emptyset$; matrix M:=$\emptyset$;
   /* adversarial learning via supervised random walk*/
2. extracts I^+, I^- from ΔE^a;
3. model Ł := SRW (I^+, I^-, G);
   /* inference of critical links from edge transitions*/
4. induce matrix M from Ł over G;
5. E^p := recomLink (I^+, I^-, M, b);
6. Return E^p;

Fig. 3: Algorithm DetCritical (learning and link inference)

(1) Extracts learning instances from E^a, and initializes positive and negative links $I^+(v)$ and irrelevant set $I^-(v)$ for each node v involved in E^a. It also converts node attributes to feature vectors $h(v)$ for model learning.

(2) Invokes a procedure SRW to learn a model "biased" to the preferential links E^c. We specify each node v and its positive and negative sets $I^+(v)$ and $I^-(v)$ as the training examples for supervised random walk [2]. We adopt logistic edge strength computation for function f_ω, and use L-BFGS [14] to solve the optimization problem.

Once the link inference model is learned, the strength of a link (v, v') can be computed as $\alpha_{vv'} = f_\omega(F_u, F_v)$. The edge transition matrix M is then induced as

$$\mathcal{M}_{vv'} = \begin{cases} \frac{\alpha_{vv'}}{\sum_{m \in V_D} \alpha_{vm}} & \text{if } (v, v') \text{ is an edge in G'} \\ 0 & \text{otherwise} \end{cases}$$

(3) The critical links E^p are then induced by a link recommendation procedure recomLink. The parameterized procedure recomLink is configurable to recommend b (and up to $|V| \times |V|$) critical links.

Recommend Critical Links. The procedure recomLink (not shown) recommends b links as follows. It first samples a set of source nodes $V_s \subseteq V$ (see "source sampling"). For each source node $v \in V_s$, it then chooses top $\lfloor \frac{b}{|V_s|} \rfloor$ nodes v' with the highest probability $M'_{vv'}$ from a random walk probability matrix M'. The entry $M'_{vv'}$ encodes the stationary probability that v reaches v' via random walk with restart following the edge transition matrix M. This generates up to b critical links.

The random walk probability $M'_{vv'}$ of a link (v, v') is assigned by a stationary distribution vector p based on the random walk with restarts following edge transition matrix M. More specifically, p is derived as

$$p^T = p^T M'; \quad \vec{M'}_v = (1 - \beta)\vec{M}_v + \beta(\vec{e}_v)$$

Here β refers to the restart probability, and $\vec{e}_v$ is a binary vector with all '0' entries except the one for v (set to $'1'$).

These critical links E^p represent

are consulted to explicitly recommend the links to be manipulated (Section V-B), or as approximate indicators of the "virtual" attentions (Section V-C).

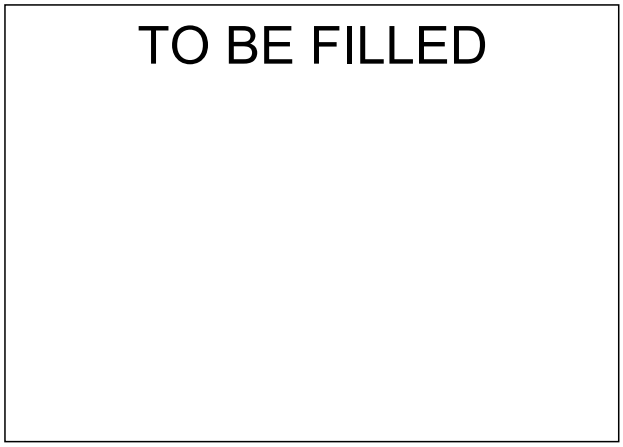

Fig. 4: Bounded Imputation

B. Bounded Link Inference

The adversarially learned stochastic transition matrix M naturally suggests useful link manipulations. It is desirable to manipulate links with a controllable scheme to balance the learning and manipulation cost and model performance. For example, motivation on cost of link manipulation. Our first framework, denoted as BoundInf, applies a configurable *bounded inference* to manipulate at most k links to improve GNN-based learning.

Bounded Inference. An overview of BoundInf is illustrated in Fig. 4. We introduce an algorithm, denoted as BoundInf and illustrated in Fig. 5. It has the following steps.

(1) It first invokes a procedure scope to refine the learning scope $\mathcal{S}$. A corresponding training graph G_M is then refined by S. By default, G_M is induced as defined in Section V-A for inductive and transductive learning tasks, respectively. For large G, scope applies a sampling strategy to induce a set of source node S with $|S| \leq k$ (see 'scope sampling').

(2) BoundInf then learns a biased link inference model and computes a stochastic transition matrix $\mathcal{M}$ (Section V-A).

(3) A procedure augment then selectively samples a set of target nodes and computes a set of links to be imputed for each node. These links and induced new neighbors are explicitly augmented for each node. This induces an augmented training graph G'_M to be accessed for follow-up GNN-based learning.

Procedure augment. For each node v from a set of target nodes, augment computes a set of non-neighbors to be augmented to $N(v)$, as the top b nodes in $V_M \setminus N(v)$ with the highest probability p_u assigned by the stationary distribution p vector based on the random walk with restarts following $\mathcal{M}$. More specifically, p is computed as

$$p^T = p^T \mathcal{M}'$$

where $\mathcal{M}'$ is the random walk probability matrix. For each node pair (v, v') (v is a target node, and $v' \in V_M$), $\mathcal{M}'_v$ is computed as

$$\vec{\mathcal{M}}'_v = (1-\beta)\vec{\mathcal{M}}_v + \beta(\vec{e}_v)$$

Framework BoundInf (specialized for GNN model M)
Input: learning task $\mathcal{T}$=$(\mathcal{D}, G)$, relevance model P, radius r, bound n;
Output: a GNN model M'.
1. scope $\mathcal{S}$:= scope $(\mathcal{D}, G, P, r)$;
2. induce training graph G_M from $\mathcal{T}$;
3. $\mathcal{M}$:= DetCritical $(G_M, \mathcal{T}, \mathcal{S})$;
4. G'_M = augment $(\mathcal{M}, G_M)$;
5. M' = LearnM(D, G'_M).

Fig. 5: Specializing stackBI for GNN model M

where β is the restart probability, and $\vec{e}_v$ is an vector with length of $|V_M|$ with v-th element equal to 1 and others all equal to 0. That is, $\mathcal{M}'_{vv'}$ encodes the stationary probability that v reaches v' with random walk with restart, following the transition probability $\mathcal{M}$.

Feasible implementation. To make the bounded imputation scheme feasible on large graphs, BoundInf is implemented by a two-step sampling-based refinement process. The first aims to shape a proper learning scope for practical biased link prediction in large G. The second is on refining the relevant set to retain the nodes with high likelihood to be linked.

Scope sampling. The scope that involves the entire training graph G_M may still be large. Given the nodes V_M in G_M, we sample a set of source nodes S from V_M with several sampling strategies. One strategy is to start from nodes with training labels as seeds and sample nodes are tend to be either in the similar class with seed nodes or semantically close to the seed nodes. We leverage the stationary distribution learned from the supervised random walk to gradually expand the subgraphs that are induced from the source seeds and their one-hop neighbors. This setting suits under an assumption that the full graph is too large for its global network structure to be known as a whole. As a result, nodes with similar semantic meaning are grouped. Then for an enriched graph, the most relevant node has a higher chance to be identified through this biasing sampling [4].

Relevance refinement. We next sample important nodes in the graph G_M that participate neighborhood augmentation. We make use of several heuristic node importance ranking measures, such as degree, betweenness and PageRank. We are aware of other approaches such as motif-based measures [17], [26], where nodes appear with more subgraph patterns are more likely to benefit learning tasks when motif-based adjacency matrices are used. Since our model shares similarity with PageRank measurement, we adopt PageRank algorithm to rank and select top k nodes, leading to at most $k * b$ new neighbors imputed via link prediction. We defer the exploration of more sampling strategies in future work.

C. Masked Attention Model

While missing neighbors and long-range relations can be explicitly imputed by BoundInf, equal importance are enforced for any follow-up GNN-based learning due to the external inference. It is desirable to incorporate attention mechanism into the process. One option is to stack the biased link inference

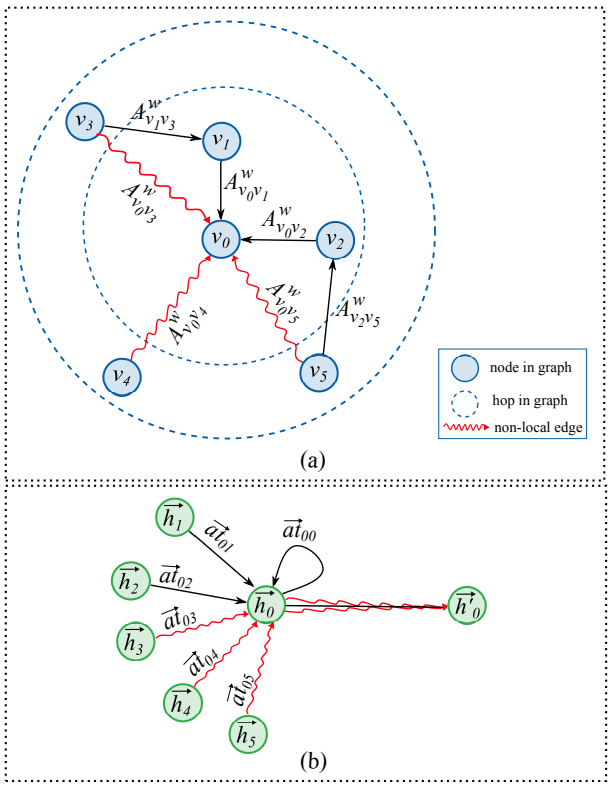

Fig. 6: Masked Attention Model

with graph attention network (GAT). Despite the sampling strategy, the process may still incur considerable augmentation cost due to *e.g.,* computing stationary distribution vectors, independent of the learning of GNN.

We present a second strategy to integrate biased link inference into attention learning without graph augmentation. The main idea is to "enlarge" the first-order neighbors to a *virtual counterpart* that include flexible r-hop relevant sets following a a revised stochastic transition matrix. The proposed model (denoted as MaskInf) preserves the strength of GAT that can implicitly specify different weights to neighbors, while incorporate virtual neighbors including both missing neighbors and long-range relations to receptive fields. MaskInf is applicable to both inductive and transductive tasks.

Model overview. An overview of the model MaskInf is as illustrated in Fig. 6. There are two major components: *transition learning*, and *attention learning* with virtual neighbors.

Transition Layer. In BoundInf, the strength $\alpha_{vv'}$ in the stochastic transition matrix $\mathcal{M}$ models the transition probability from a node v to a node v' in the training graph. We revise $\mathcal{M}$ to a counterpart $\mathcal{M}^A$ as an approximate indicator of the probability of "virtual neighbors":

$$\mathcal{M}^A_{vv'} = \frac{\alpha_{vv'}}{\sum_{m \in V_D} \alpha_{vm}}$$

Attention layer. The attention layer extends receptive field to a set of virtual neighbors of each node v. The virtual neighbors of v is determined as a set $\{v' | A^w_{vv'} > 0\}$, where the matrix A^w is computed as the element-wise multiplication $A \odot \mathcal{M}^A$, and A is an adjacency matrix extended by a task-specific hyper-parameter r:

$$A_{vv'} = \begin{cases} 1 & \text{if } v' \in N_r(v); \\ 0 & \text{otherwise} \end{cases}$$

The MaskInf hidden layer takes input as a set of node features $\mathbf{h} = \{\vec{h}_1, \ldots \vec{h}_{|V_D|}\}$ ($\vec{h}_i \in \mathbb{R}^d$). The expected output $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, ..., \vec{h}'_{N_t}\}$, $\vec{h}'_i \in \mathbb{R}^{d'}$.

We inject the graph structure into MaskInf by masked attention. We perform linear transformation on a weight matrix $\mathbf{W}$ and self-attention on the nodes–a shared attentional mechanism $\vec{a} : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$ computes attention coefficients as follows:

$$z_i = \mathbf{W} h_i,$$

$$se_{ij} = \text{LeakyReLU}(\vec{a}^T (z_i || z_j))$$

where LeakyReLU is a nonlinear function, $\cdot^T$ represents transposition and $||$ is the concatenation operation.

The attention mechanism is expressed as:

$$at_{ij} = \frac{\exp(se_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(se_{ik})}$$

and the output $\vec{h}'_i$ is computed as:

$$\vec{h}'_i = \sigma(\sum_{j \in \mathcal{N}_i} at_{ij} z_j)$$

Similar as [24], we can extend this mechanism to multi-head attention to stabilize the learning of MaskInf. The learning algorithm (also denoted as MaskInf) is outlined in Fig. 7.

Algorithm MaskInf
Input: learning task T$(\mathcal{D}, G)$, relevance model P, radius r;
Output: a MaskInf model M'.
1. scope $\mathcal{S}$:= scope $(\mathcal{D}, G, P, r)$;
2. induce training graph G_M from $\mathcal{T}$;
3. $\mathcal{M}^A$:= DetCritical $(G_M, \mathcal{T}, \mathcal{S})$;
4. M' = Learn(D, G_M, $\mathcal{M}^A$).

Fig. 7: Algorithm MaskInf

VI. Experiment

Using real-world graphs and established graph-based benchmark tasks (transductive), we perform comparative evaluation to verify: (1) the performance gain achieved by BoundInf and MaskInf, compared with standalone GNN models, and (2) the impact of the parameters to their performance, and (3) [**Sheng**]:the efficiency of BoundInf and MaskInf and the impact of the parameters, and (4) case analysis to illustrate the model interpretability.

	Cora	Citeseer	OAG	DBpedia	Weibo	Yelp
# Nodes	2,708	3,327	11,165	9,595	11,190	3268
# Edges	5,429	4,732	12,990	5,540	9,936	2482
# Features/Node	1,433	3,703	900	900	4,596	900
# Classes	7	6	2	2	2	2
# Training Nodes	140	120	670	576	6,714	196
# Validation Nodes	500	500	1116	960	2,238	327
# Test Nodes	1,000	1,000	3350	2,878	2,238	980

TABLE II: Summary of the data and training set up

A. Experiment Setting

Datasets. We use six real-world graph datasets. (1) Cora and Citeseer are two citation networks where the nodes are documents and edges encodes citation. Each node has features derived from a bag-of-words representation of the document it specifies, and a class denoting the area of the document (*e.g.,* reinforment learning , theory, and so on).(2) DBpedia, a fraction of knowledge graph[1] with 9595 entities and 5540 relations. Each edge encodes semantic relations such as "student of", "colleague". Each node has a type (*e.g.,* 'professor', 'academic advisor'), as well as features derived from entity attributes (*e.g.,* affiliation, title). (3) Weibo[2], a social information network where nodes carries posted microblog text, and edges encode to repost/forward. Node feature consists a keyword list with high frequencies. Features are derived from a bag-of-words representation of these keywords. (4) OAG, a fraction of the open academic graph[3], where each node can be papers and authors, and each edge can be "cite" and "affiliatedTo"; (5) Yelp[4], a business review graph with nodes as users and local services (*e.g.,* plumbers, restaurants, etc.), and edges such as "friendWith".

Learning Setting. We evaluate BoundInf and MaskInf over the following classification tasks.

(1) The benchmark *Multilabel classification* over Cora and Citeseer. For Cora and Citeseer, the task is to correctly predict the relevant topics of a single published document (with cases of multidisciplinary research). We use the benchmark setting with the original classes as in [32]. [**Sheng**]:add the detailed setting of the experiment settting, GCN,GAT, MixHop and MaskInf.

(2) *Erroneous Detection* over OAG, DBpedia, Weibo and Yelp. For OAG, DBpedia and Yelp, the binary classification task is to identify entities with erroneous information by learning from known "dirty" fraction of knowledge bases. We produce training examples and ground truth with established benchmark tool [1] to inject errors (*e.g.,* typos, bogus values, null values, and outliers) that violate verified data constraints to entities. For Weibo, the task identifies misinformation for given ground truth text. We simulated propagation of misinformation by incorporating message disturbance model into an information propagation model [7] from multiple known sources and derived ground truth and training examples. [**Sheng**]:add the detailed setting of the experiment setting,GCN,GAT, MixHop and MaskInf.

Edge Poisoning Attacks [**Sheng**]:add the edge poisoning attack model description here.

[**Sheng**]:It is well-studied that only **TBF** % edge perturbations are sufficient to lead a strong decrease in performance for GNN models [40]. We set a bound B as **TBF** % edge perturbations of each dataset due to attacks or missing edges, *i.e.,* limit the number of changes $\|A - \hat{A}\|_0 \leq B$. Furthermore, during the attack, we guarantee that no node becomes disconnected and we employ unnoticeability constraint on the degree distribution as [39] suggests that graph attack should remain unnoticeable and the degree distribution of the graph can only marginally be modified by the attacker.

Models. We evaluate BoundInf framework and MaskInf with the following representative GNN models:

- GCN: a class of GNN in semi-supervised setting;
- GAT: a class of GNN with self-attention mechanism; and
- MixHop:[**Sheng**]:a state-of-the-art GNN with non-local learning.

In accordance, we implemented BoundInf specifications with model M (denoted as BoundInf (M)), which include: BoundInf (GCN), BoundInf (GAT) and [**Sheng**]:BoundInf (MixHop). [**Sheng**]:At the same time, to show the effectiveness of BoundInf, we add representative link prediction model [2] **TBF** as LP and sequentially combine with model M, which include: LP +GCN, LP +GAT, LP +MixHop.

Configuration. We adopt the following relevance function P to decide relevance and irrelevance sets in BoundInf. (1) For Cora, Citeseer, OAG and Yelp, we adopt semantic closeness measurements; (2) For DBpedia, we adopt association measurements and ontological similarity [16], [22]; (3) For Weibo, we adopt geo-social closeness measurements [36]. [**Sheng**]: We set the same bound B for edge imputation as edge poisoning attack tool did (in total $\leq B$ edges are explicitly inserted or removed to training graphs).

For transductive learning, we evaluate BoundInf(GCN) and BoundInf(GAT) with GCN and GAT, respectively. [**Sheng**]:To consider non-local learning, we further study BoundInf(MixHop) with MixHop. For a fair comparison, all the configuration of the models GCN, GAT and MixHop are calibrated to yield consistent performance over benchmark metrics as in [10], [24] over benchmark tasks and datasets (Cora, Citeseer). We adopt the exact setting that are aligned with their experiments description. For the BoundInf(GCN), BoundInf(GAT), and BoundInf(MixHop), we still follow the exactly same setting.

Metrics. For a fair comparison using benchmark metrics and tasks [10], [24], we report the average mean classification accuracy (acc.) for transductive tasks over Cora and Citeseer. We report both accuracy and macro-averaged precision (m.prec) for the binary classification over DBpedia, Weibo, OAG and Yelp.

[1]https://wiki.dbpedia.org/develop/datasets
[2]https://www.aminer.cn/data-sna
[3]https://www.openacademic.ai/oag/
[4]https://www.kaggle.com/yelp-dataset

B. Experimental Results

Exp-1: Effectiveness of Bounded Imputation. In this set of experiments, we evaluate the gain of performance of BoundInf over GNN-learning with GCN, GAT and MixHop, respectively.

[**Sheng**]:We treat the original datasets as "clean" datasets to pretend that they are not suffered from any edge attacks. We firstly show that even for the "clean" datasets, applying BoundInf over GNN-learning boosts performance. BoundInf methods would treat "clean" datasets as if they have missing or unreliable links. some results need to be added here. [**Sheng**]: We add "perfect-" as the prefix for the method that achieves the best performance on the "clean" datasets. By selecting the best result for each dataset, we can get an approximate optimal performance result for this dataset. Then, we apply edge poisoning attack tool on all datasets with **TBF** % perturbed edges.

Performance gain (transductive, perturbed datasets).[**Sheng**]: Table III reports the node classification results under the transductive setting where all datasets suffer from edge poisoning attack and bounded link imputations are essential and necessary to adversarially boost the performance. Comparing BoundInf (GCN), BoundInf (GAT) and BoundInf (MixHop) with GCN, GAT and MixHop, respectively. BoundInf framework improves the benchmark metrics of standalone GNN models and simpler mechanisms as LP + GNN in all cases. For example, the accuracy gain is **TBF** % for error detection task in **TBF**. more analysis needs to be filled

Performance of MaskInf. Using the same setting [**Sheng**]:may expand more, the result of MaskInf is reported in Tables III. MaskInf achieves comparable performance with BoundInf (GAT). By exploiting the non-local relations inferred by biased link inference, it also outperforms GAT in all the cases. more analysis to be added here We observe that MaskInf improves GAT better over **TBF**, which may be benefiting from more non-local semantically relations.

Exp-2: Impact of edge perturbation ratio. We next investigate the impact of the number of attacked edges to the performance of all the models. We modify training graphs by "attacking" edges from the original graph G [39], controlled by the size that is proportional to the size of G. Specifically, (1) for dense **TBF**, we afford to simulated a simple structure-preserving attack [39] such that the degree distribution is not significantly changed; (2) for more sparse graphs (Cora, Citeseer, DBpedia and Weibo), we randomly selected nodes and perturbed edges for transductive learning.

Fixing other parameters as default, we varied the percentage p_r of edges perturbed from the training graph over dataset **TBF** from 10% to 50%. Fig 8 reports the impact of the percentage of edges perturbed from the training graph to the performance of the models (*e.g.,*GCN), for multilabel classification over Citeseer, and error detection (binary classification) over DBpedia. The performance of GCN degrades as more edges are perturbed over Citeseer, while BoundInf (GCN) achieves a quite stable accuracy gain. The performance of GCN over DBpedia (in macro-precision) is less sensitive, while BoundInf (GCN) can achieve a gain on precision at **TBF**% when **TBF**% edges are missing from the training graph, and is insensitive to the amount of removed edges.

TO BE FILLED

Fig. 8: Change in accuracy of **TBF** on **TBF** for edge perturbation

TO BE FILLED

Fig. 9: Change in accuracy of **TBF** on **TBF** for increasing number of edge imputation

Exp-3: Impact of edge imputation ratio. Fixing other parameters as default, we varied edge imputation percentage p_i of fixed imputation budget B over dataset **TBF** from 10% to 90% and report the result in Fig. 9 more analysis will be added here. LP+GCN vs StackBI(GCN), remember to mention LP here is modified to support bounded edge imputation.

Exp-4: Impact with limited knowledge about the graph. more will be added here In Fig. 10 we compare **TBF**.

Exp-5: Efficiency of Bounded Imputation. We also evaluated the time cost of BoundInf and MaskInf (not shown). The link inference in BoundInf (M) does not incur much additional cost compared with the standalone learning cost for model M, with M ranges over GCN, GAT and MixHop. For example, it introduces up to 60% of the learning time of GAT over cora dataset, due to the scope sampling and relevance refinement process. The learning of MaskInf is also quite feasible: it takes up to 40% to learn MaskInf over cora with $2,708$ nodes and

dataset	Cora		Citeseer		DBpedia			Weibo			OAG			Yelp		
metrics	acc.	time	acc.	time	acc.	m.prec.	time	acc.	m.prec.	time	acc.	m.prec.	time	acc.	m.prec.	time
GCN	$65.6 \pm 1.8\%$	**TBF**	$45.9 \pm 0.7\%$	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
LP + GCN	$66.2 \pm 2.0\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
stackBI(GCN)	$68.1 \pm 1.2\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
GAT	$63.8 \pm 0.6\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
LP + GAT	$63.0 \pm 0.6\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
stackBI(GAT)	$64.5 \pm 0.5\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
MixHop	**TBF** %	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
LP + MixHop	**TBF** %	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
stackBI(MixHop)	**TBF** %	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
maskBA	$71.0 \pm 0.6\%$	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**
perfect-	**TBF** %	**TBF**	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**	**TBF** %	**TBF** %	**TBF**

TABLE III: Results for node classification with **TBF** % perturbed edges. Bold: best result; Underlined: second best.

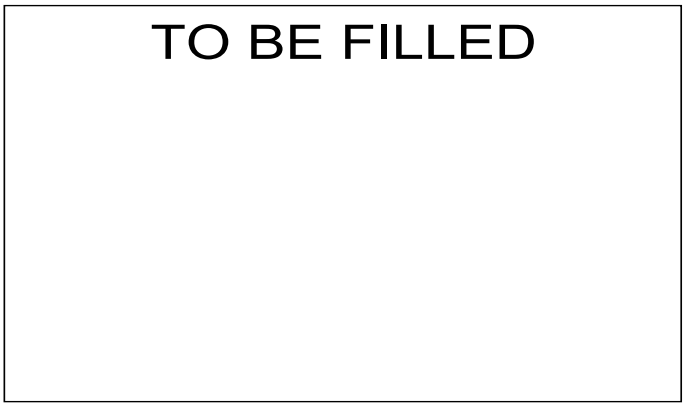

Fig. 10: Bounded imputation with limited knowledge about the graph (i.e. on a subgraph) after **TBF**

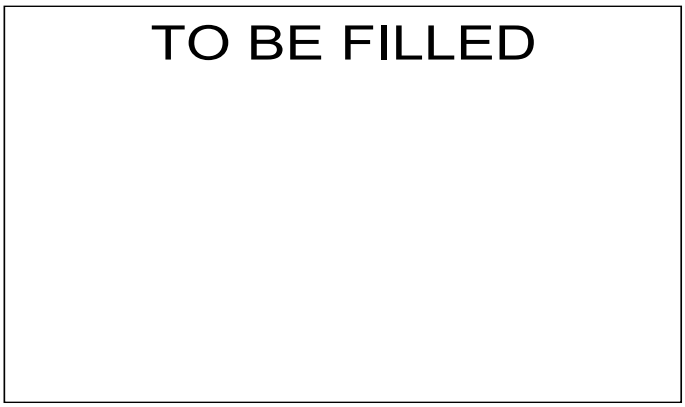

Fig. 11: Efficiency for edge perturbation

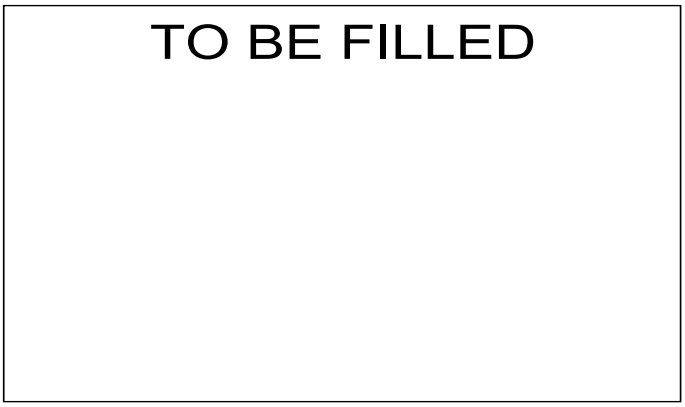

Fig. 12: Efficiency with limited knowledge about the graph

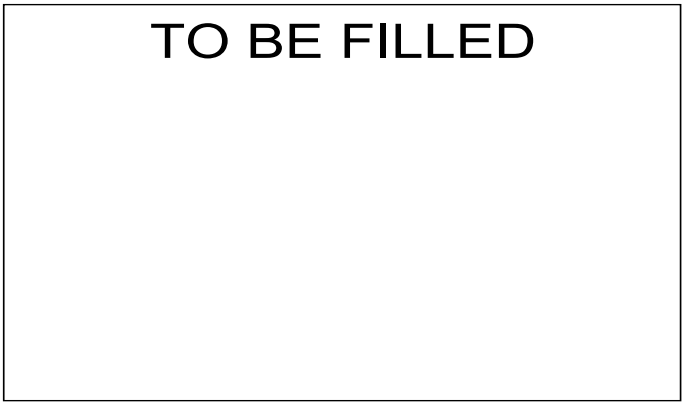

Fig. 13: Ablation analysis

5429 edges, and is comparable with the learning cost of GAT.

Factor edge insertion bound B, graph size [**Sheng**]:overview of efficiency

Efficiency for edge perturbation[**Sheng**]:change the percentage of edge perturbation. In Fig. 11 we compare **TBF**.

Efficiency for limited knowledge of the graph[**Sheng**]:change the percentage of nodes in the subgraph we can access. In Fig. 12 we compare **TBF**.

Exp-6: Ablation analysis. In Fig. 13 we compare **TBF**.

Exp-7: Visual Analysis. We provide a more intuitive comparison naturally enabled by the augmentation phase in BoundInf framework. Fig. 14 illustrates a source node v_6 in the left subgraph of PPI dataset and a set of inferred non-local neighbors (*e.g.,* v_2, v_5) that are augmented to the nodes. For this case, an instance v_2, v_6 of the training nodes has relevant sets that include the common 32 attributes out of total 50 attributes. We found that GraphSage achieves a worse result if no augmentation is added, while BoundInf learns to link more relevant non-local nodes (*e.g.,* v_2, v_5) that share lots of common features with node v_6. A similar case is illustrated in right-hand subgraph. Interestingly, a link is inferred between two disconnected yet similar nodes v_0 and v_7. The features from the neighbors of the latter contribute to a correct classification.

The cases also suggest that intuitive lineage information (*e.g.,* augmented links) can be explicitly suggested by BoundInf specifications, which suggests potential explanation tools for corresponding GNN-based learning. We shall defer the extension in future work.

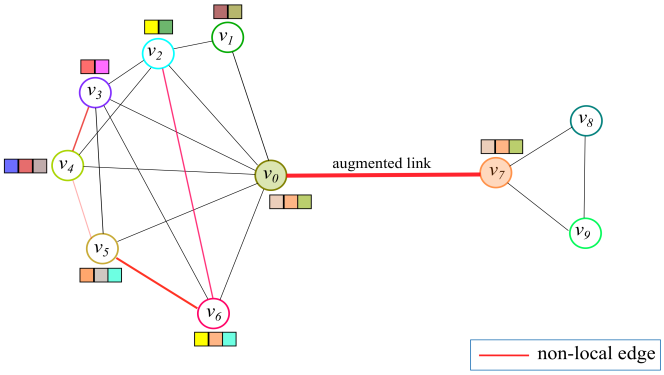

Fig. 14: A visual analysis of BoundInf. Bolder links indicate larger weights assigned by biased link inference.

VII. Conclusion

We have proposed a novel framework to enhance GNN-based learning to exploit non-local relationships from incomplete graphs. We specialize BoundInf, a bounded imputation framework that explicitly suggest new links by a biased link inference process; and MaskInf, a masked attention model to incorporate attention mechanism into virtual neighbors without incurring link augmentation cost. Our experimental study using real-world graph and benchmark tasks verified the effectiveness of the framework. Our general framework and models can be applied to downstream tasks besides node classification. One future topic is to explore the application of our models upon malicious graph manipulation targeted at specific GNNs. Another topic is to investigate effective explanation mechanism by leveraging various task-driven relevance models.

References

[1] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *Proceedings of the VLDB Endowment*, 2015.

[2] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 2011.

[3] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371*, 2018.

[4] M. Fang, J. Yin, and X. Zhu. Supervised sampling for networked data. *Signal Processing*, 124:93–102, 2016.

[5] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*, 2018.

[6] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005.

[7] A. Guille, H. Hacid, C. Favre, and D. A. Zighed. Information diffusion in online social networks: A survey. *Sigmod Record*, 42(2):17–28, 2013.

[8] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.

[9] D. Harris and S. Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[10] A. Kapoor, A. Galstyan, B. Perozzi, G. Ver Steeg, H. Harutyunyan, K. Lerman, N. Alipourfard, and S. Abu-El-Haija. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. 2019.

[11] M. Kim and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM, 2011.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[13] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 1989.

[15] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 2011.

[16] H. Ma, M. Alipourlangouri, Y. Wu, F. Chiang, and J. Pi. Ontology-based entity matching in attributed graphs. *Proceedings of the VLDB Endowment*, 12(10):1195–1207, 2019.

[17] F. Monti, K. Otness, and M. M. Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018.

[18] S. Nandanwar and M. N. Murty. Structural neighborhood based classification of nodes in a network. In *KDD*, 2016.

[19] N. Park, A. Kan, X. L. Dong, T. Zhao, and C. Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[20] H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 2017.

[21] K. Sun, Z. Lin, and Z. Zhu. Adagcn: Adaboosting graph convolutional networks into deep models. *arXiv preprint arXiv:1908.05081*, 2019.

[22] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011.

[23] C. Tran, W.-Y. Shin, and A. Spitz. Community detection in partially observable social networks. *arXiv preprint arXiv:1801.00132*, 2017.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[25] B. Wang and N. Z. Gong. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[26] P. Wang, J. Lü, and X. Yu. Identification of important nodes in directed biological networks: A network motif approach. *PloS one*, 2014.

[27] X. Wang, R. Girshick, A. Gupta, and K. He. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[28] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *NeurIPS*, 2020.

[29] Y. Xiao, Z. Zhang, C. Yang, and C. Zhai. Non-local attention learning on large heterogeneous information networks. 2019.

[30] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[31] Y. Yang, H. Guo, T. Tian, and H. Li. Link prediction in brain networks based on a hierarchical random graph model. *Tsinghua Science and Technology*, 2015.

[32] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[33] Z. Yin, M. Gupta, T. Weninger, and J. Han. A unified framework for link recommendation using random walks. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, 2010.

[34] C. Yu, X. Zhao, L. An, and X. Lin. Similarity-based link prediction in social networks: A path and node combined approach. *Journal of Information Science*, 2017.

[35] X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li. Adversarial examples: Attacks and defenses for deep learning. corr abs/1712.07107 (2017). *arXiv preprint arXiv:1712.07107*, 2017.

[36] J.-D. Zhang and C.-Y. Chow. igslr: personalized geo-social location recommendation: a kernel density estimation approach. In *SIGSPATIAL*, 2013.

[37] Y. Zhang, Y. Xiong, X. Kong, S. Li, J. Mi, and Y. Zhu. Deep collective classification in heterogeneous information networks. In *WWW*, 2018.

[38] T. Zhou, L. Lü, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 2009.

[39] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.
[40] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. *ICLR*, 2019.