# Inductive Link Prediction with Interactive Structure Learning on Attributed Graph

Shuo Yang, Binbin Hu, Zhiqiang Zhang, Wang Sun, Yang Wang, Hongyu Shan, Jun Zhou [⋆], Yuetian Cao, Borui Ye, and Yanming Fang

Ant Group
{kexi.ys,bin.hbb,lingyao.zzq,sunwang.sw,zuoxu.wy, xinzong.shy, jun.zhoujun, yuetian.cyt,borui.ybr,yanming.fym,jingmin.yq}@antgroup.com

**Abstract.** Link prediction is one of the most important tasks in graph machine learning, which aims at predicting whether two nodes in a network have an edge. Real-world graphs typically contain abundant node and edge attributes, thus how to perform link prediction by simultaneously learning structure and attribute information from both interactions/paths between two associated nodes and local neighborhood among node's ego subgraph is intractable.
To address this issue, we develop a novel **P**ath-**a**ware **G**raph **N**eural **N**etwork (PaGNN) method for link prediction, which incorporates interaction and neighborhood information into graph neural networks via broadcasting and aggregating operations. And a cache strategy is developed to accelerate the inference process. Extensive experiments show a superior performance of our proposal over state-of-the-art methods on real-world link prediction tasks.

## 1 Introduction

Graph-structured data are ubiquitous in a variety of real-world scenarios, e.g., social networks, protein-protein interactions, supply chains, and so on. As one of the most common and important tasks of graph mining, *link prediction*, which aims at predicting the existence of edges connecting a pair of nodes in a graph, has become an impressive way to solve various crucial problems such as friend recommendation [24, 26], supply chain mining [29], entity interactions prediction [28, 23], and knowledge graph completion [17, 7].

In general, current researches towards link prediction can be categorized into three lines: heuristic methods, network embedding based methods and graph neural network based methods. *Heuristic methods* [13, 2] focus on estimate the likelihood of the edge through different heuristic similarities between nodes under certain assumptions, which, unfortunately, may fail when their assumptions do not hold true in the targeted scenario [11]. *Network Embedding* (NE) based methods [3] learn node representation with context (e.g., random walk [14, 5] or neighborhood [22]) in the graph, followed by a well-trained classifier for link

[⋆] Corresponding author

prediction. However, most of them fail to take into account the rich attributes of nodes and edges when learning node representation, so they cannot obtain better performance and are not suitable for inductive link prediction. By making full use of the structure and attribute information in an inductive manner, the emerging *Graph Neural Network* (GNN) based methods [25] achieve the state-of-the-art performance in link prediction [12].

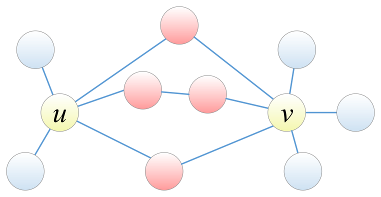


**Fig. 1.** Node roles for link prediction.

Nevertheless, we believe that the effectiveness and efficiency of current GNN-based methods for link prediction are still unsatisfactory. Current GNN-based methods can be categorised into two lines: node-centric and edge-centric. *Node-centric* GNN-based methods [31] learn representations of two targeted nodes via certain GNN architecture independently, followed by a pairwise prediction function (e.g., MLP, dot product, etc). Such a two-tower architecture is good at modeling the surrounding context centered at each targeted node, but fails to perceive **interactions** (or **paths**, red nodes in Fig 1) between two targeted nodes (yellow nodes in Fig 1), which are essential for the *effectiveness* of some real-world link prediction scenarios. Recently, several *edge-centric* GNN-based methods propose to adopt a different technique (e.g., node labeling function [33], between-node path reachability[27], enclosing-subgraph level pooling [20], meta-graph [34], etc) to model such interactions or paths to some extent, and achieve better performance. However, on the one hand, they still do not explicitly integrate the structure and attribute information of interactions between targeted nodes. For example, the node labeling function [33, 20] only models the structure information to some extent, while the between-node path reachability [27] only estimates reachable probability from one node to another via random walk. Both of them neglect the abundant *attributes* of interactions/paths. On the other hand, all of the above approaches are time-consuming in the training or inference phase, thus the *efficiency* becomes a great challenge when scaling up to industrial graphs with billions of nodes and tens of billions edges. For example, Graph learning framework [27] computes the reachability via random walks, which need to be repeated many times until convergence, and this process also has to be redone for any newly emerging edge. Therefore, it is time-consuming when performing inference in huge graphs.

Here, we summarize the challenges facing by current link prediction methods in three aspects:

– *Integrating the structure and attribute information of interaction.* Since most real-world graphs contain abundant node and edge attributes, and most scenarios can benefit from such information, the link prediction model should be able to subtly model the structure information and attribute information simultaneously.
– *Scalability and Efficiency.* Existing works that model interactions by calculating high-order information are usually time-consuming. How to scale up and become more efficient is another challenge when performing over real-world huge graphs.
– *Inductive ability.* In real-world graphs, nodes and edges may emerge at any time. To handle such newly emerging nodes and edges requires the inductive ability of the link prediction model.

Addressing the above challenges, we propose a **P**ath-**a**ware **G**raph **N**eural **N**etwork (PaGNN) towards link prediction. Considering the motivation of both node-centric and edge-centric methods, PaGNN jointly learns the structure and attribute information from both interactions/paths between two targeted nodes (edge-centric) and the local neighborhood of each targeted node (node-centric), through the novel *broadcasting* and *aggregation* operation. The broadcasting operation responds to "send" information from one targeted node to all other nodes in its local neighborhood and generate the *broadcasted embeddings*, while the aggregation operation aims to aggregate information (including to "receive" the broadcasted embeddings) for another targeted node from its local neighborhood and generate its final embedding. Note that the destination node can perceive all paths connecting two targeted nodes via aggregating broadcasted embeddings. Thus, such a broadcasting and aggregation operation can explicitly integrate structure and attribute information of interactions and local neighborhood of targeted nodes. In addition, addressing the poor scalability and efficiency of edge-centric methods in the inference phase, we propose a *cache embedding strategy* that nearly doubles the speed of the inference phase. Note that by leveraging the native inductive power of GNNs, PaGNN can handle newly emerging nodes and edges naturally. At last, We conduct extensive experiments on several public datasets to demonstrate the effectiveness and efficiency of PaGNN compared with state-of-the-art baselines.

## 2 Model Formulation

In this section, we introduce the proposed path-aware graph neural network (PaGNN) model towards link prediction. First, we briefly exhibit the overall architecture of PaGNN. Then, we elaborate the detail implementation of PaGNN, including the broadcasting operator, the aggregation operation, edge representation learning, and at last the loss function.

### 2.1 Notations and Definitions

Before diving into PaGNN, we first give the definition of link prediction.

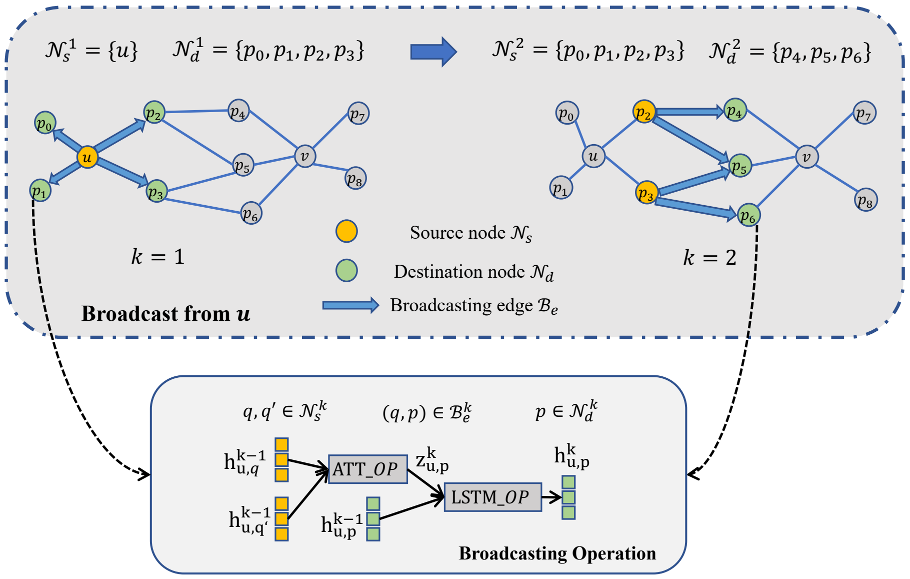

$\mathcal{N}_s^1 = \{u\}$ $\mathcal{N}_d^1 = \{p_0, p_1, p_2, p_3\}$
$\mathcal{N}_s^2 = \{p_0, p_1, p_2, p_3\}$ $\mathcal{N}_d^2 = \{p_4, p_5, p_6\}$
$k = 1$
$k = 2$
Source node $\mathcal{N}_s$
Destination node $\mathcal{N}_d$
Broadcasting edge $\mathcal{B}_e$
Broadcast from $u$
$q, q' \in \mathcal{N}_s^k$
$(q, p) \in \mathcal{B}_e^k$
$p \in \mathcal{N}_d^k$
ATT_OP
LSTM_OP
Broadcasting Operation

**Fig. 2.** Broadcasting Operation

**Definition 1.** ***Link Prediction.*** *Given an attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of observed edges, and $\mathbf{X} \in R^{|\mathcal{V}| \times d}$ consists of d-dimensional feature vectors of all nodes, as well as a set of labeled edges $\mathcal{L} = \{(\langle u, v \rangle, y) | u, v \in \mathcal{V}, y \in \{0, 1\}\}$, $y = 1$ denotes that there exists an edge between u and v (i.e., $(u, v) \in \mathcal{E}$), otherwise $y = 0$, for an unlabeled edge set $\mathcal{U} = \{(\langle u, v \rangle | u, v \in \mathcal{V}\}$, the goal of link prediction is to predict the existence probability of edges in $\mathcal{U}$.*

## 2.2 Overview of PaGNN

PaGNN aims to learn the representations on the centralized subgraph of two associated nodes, Fig 2 and 3 show the overall workflow of PaGNN. By leveraging the broadcasting and aggregation operations, PaGNN can model all interaction (i.e., paths) and neighborhood information between two associated nodes of the targeted edge, and generate their embeddings for link prediction. PaGNN first performs broadcasting and then aggregation operation on both two targeted nodes. More specifically, one node broadcasts information (called broadcasted embedding) to nodes in its local neighborhood, then the other node aggregates information from its neighborhood If there is any overlap between the broadcasted neighborhood of two nodes, the information from one node can be aggregated by the other node via the paths between them, which means two nodes can perceive each other. Therefore, the structure and attribute information of the interactions, as well as the local neighborhood, of the two nodes, can be subtly

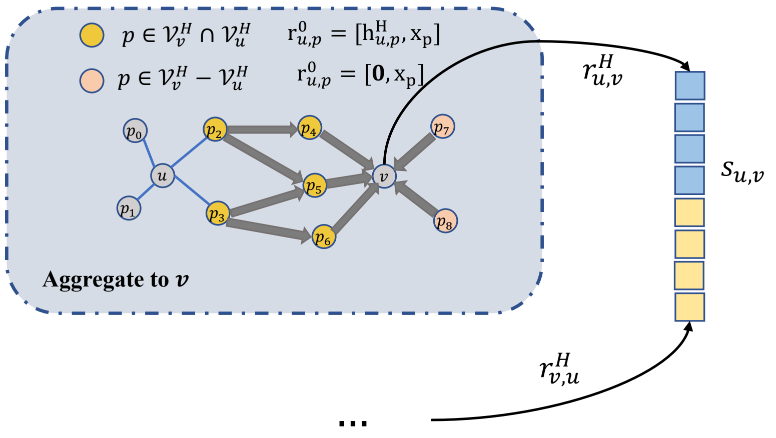

$p \in \mathcal{V}_v^H \cap \mathcal{V}_u^H$ $r_{u,p}^0 = [h_{u,p}^H, x_p]$
$p \in \mathcal{V}_v^H - \mathcal{V}_u^H$ $r_{u,p}^0 = [\mathbf{0}, x_p]$
$p_0$ $p_1$ $u$ $p_2$ $p_3$ $p_4$ $p_5$ $p_6$ $v$ $p_7$ $p_8$
Aggregate to $v$
$r_{u,v}^H$
$s_{u,v}$
$r_{v,u}^H$
...

**Fig. 3.** Aggregation Operation

modeled through such broadcasting and aggregation operations, and encoded into their final embeddings. At last, PaGNN simply employs the concatenation of final embeddings as the edge representation, and performs a MLP on it to do binary classification. Moreover, our framework is operated on the centralized subgraph of associated nodes within fixed steps, thus it naturally provides inductive ability.

## 2.3 Broadcasting Operation

The broadcasting operation aims to “send” a message from a node to other nodes in its local neighborhood. Formally, given a pair of nodes $\langle u, v\rangle$, the goal is to predict whether an edge exists between them. Without loss of generality, we set node $u$ to broadcast information to other nodes within the $H$-hops ego-subgraph centered on $u$, denoted as $\mathcal{G}_u^H$, details of broadcasting operation are shown in Fig 2.

Broadcasting operation is performed in a breadth first search (BFS) style that attempts to “spread out” from the source nodes. At each step, nodes (called *source nodes*) broadcast information to their directed neighbors (called *destination nodes*). Specifically, in the $k$-th step, we maintain a source node set $\mathcal{N}_s^k$, a destination node set $\mathcal{N}_d^k$ and a broadcasting edge set $\mathcal{B}_e^k$. $\mathcal{N}_s^k$ contains the nodes that will broadcast information (the initial $\mathcal{N}_s^1$ only contains node $u$), $\mathcal{N}_d^k$ consists of directed neighbors of nodes in $\mathcal{N}_s^k$, $\mathcal{B}_e^k$ are edges connecting nodes in $\mathcal{N}_s^k$ and $\mathcal{N}_d^k$, in other words, $\mathcal{B}_e^k = \{(q,p)|(q,p) \in \mathcal{E}_u^H, q \in \mathcal{N}_s^k, p \in \mathcal{N}_d^k\}$.

Each node in $\mathcal{N}_s^k$ first *broadcasts* information to their directed neighborhoods in destination nodes set $\mathcal{N}_d^k$ via the edges in $\mathcal{B}_e^k$. Next, each node in $\mathcal{N}_d^k$ *integrates* the broadcasted information together with its own embedding. Then, all destination nodes form the new $\mathcal{N}_s^{k+1}$ of the $k+1$-th step. This process will be repeated $H$ times until all nodes in $\mathcal{G}_u^H$ receive broadcasted information from $u$.

**Algorithm 1** Broadcasting Operation

```
Require: Source node $u$, the number of hops $H$,
    node $u$'s $H$-hop enclosing subgraph $\mathcal{G}_u^H = (\mathcal{V}_u^H, \mathcal{E}_u^H)$,
    original node features $\{\mathbf{x}_p \mid p \in \mathcal{V}_u^H\}$.
Ensure: broadcasted embeddings $\{\mathbf{h}_{u,p}^H \mid p \in \mathcal{V}_u^H\}$.
 1: $\mathbf{h}_{u,p}^0 \leftarrow \mathbf{x}_p, \forall p \in \mathcal{V}_u^H$
 2: $\mathcal{N}_s^1 \leftarrow \{u\}$
 3: for $k = 1...H$ do
 4:     $\mathcal{B}_e^k \leftarrow \{(q,p) \mid q \in \mathcal{N}_s^k, (q,p) \in \mathcal{E}_u^H\}$
 5:     $\mathcal{N}_d^k \leftarrow \{p \mid (q,p) \in \mathcal{B}_e^H\}$
 6:     for $p \in \mathcal{N}_d^k$ do
 7:         $\mathbf{z}_{u,p}^k \leftarrow ATT\_OP(\mathbf{x}_p, \{\mathbf{h}_{u,q}^{k-1} \mid (q,p) \in \mathcal{B}_e^k\})$
 8:         $\mathbf{h}_{u,p}^k \leftarrow LSTM\_OP(\mathbf{h}_{u,p}^{k-1}, \mathbf{z}_{u,p}^k)$
 9:     end for
10:     for $p \in \mathcal{V}_u^H - \mathcal{N}_d^k$ do
11:         $\mathbf{h}_{u,p}^k \leftarrow \mathbf{h}_{u,p}^{k-1}$
12:     end for
13:     $\mathcal{N}_s^{k+1} \leftarrow \mathcal{N}_d^k$
14: end for
```

As shown in Algorithm 1, $\mathcal{V}_u^H$ and $\mathcal{E}_u^H$ are the node set and edge set in $\mathcal{G}_u^H$ respectively. $\mathbf{h}_{u,p}^k \in \mathbb{R}^d$ denotes the broadcasted information (called *broadcasted embeddings*) that starts from $u$ and ends with $p$ at the $k$-th step. The information broadcasted from $u$ to node $p$ is initialized as $p$'s original feature $\mathbf{x}_p$ ($\mathbf{h}_{u,p}^0$, line 1). At the $k$-th step, $\mathcal{B}_e^k$ is updated as the directed edges of $\mathcal{N}_s^k$ and $\mathcal{N}_d^k$ is extracted from $\mathcal{G}_u^H$, which are the directed neighbors of $\mathcal{N}_s^k$ (line 5). In order to handle the situation that different source nodes in $\mathcal{N}_s^k$ broadcast information to the same destination node, we first employ the attention mechanism (ATT_OP in line 7) to aggregate the broadcasted embedding, and then we employs a LSTM-like operator [6] (LSTM_OP in line 8) to combine embeddings of the $k-1$-th step and the $k$-th step. Note that the integrated embedding will become the propagated embedding of the next step. And for nodes in $\mathcal{V}_u^H$ but not in $\mathcal{N}_d^k$, the broadcasted embeddings stay the same as previous step (line 10 to 12). The attention operator ATT_OP of the $k$-th step is defined as:

$$\alpha_{q,p}^k = \frac{\exp({\mathbf{v}_\phi^k}^{\mathrm{T}} \sigma(\mathbf{W}_{\phi 1}^k [\mathbf{x}_p, \mathbf{h}_{u,q}^{k-1}]))}{\sum_{q' \in N_s^k, (q',p) \in \mathcal{E}_u^H} \exp({\mathbf{v}_\phi^k}^{\mathrm{T}} \sigma(\mathbf{W}_{\phi 1}^k [\mathbf{x}_p, \mathbf{h}_{u,q'}^{k-1}]))}, \tag{1}$$

$$\mathbf{z}_{u,p}^k = \sigma(\mathbf{W}_{\phi 2}^k [\sum_{q \in N_s^k, (q,p) \in \mathcal{E}_u^H} \alpha_{q,p}^k \mathbf{h}_{u,q}^{k-1}, \mathbf{x}_p]) \tag{2}$$

where $\alpha_{q,p}^k$ is the attention value, $\mathbf{z}_{q,p}^k$ is the intermedia embedding after attention calculation and the input of the next LSTM-like operator, $[\cdot,\cdot]$ denotes the concatenation of embeddings, $\mathbf{v}_\phi^k$, $\mathbf{W}_{\phi 1}^k$, $\mathbf{W}_{\phi 2}^k$ are learnable parameters. Next, the

LSTM_OP integrates the information of the $k-1$-th step and the $k$-th step:

$$\begin{aligned}
\mathbf{i}_{u,p}^{k} &= \sigma(\mathbf{W}_{\varphi i}[\mathbf{h}_{u,p}^{k-1}, \mathbf{z}_{u,p}^{k}]), \\
\mathbf{f}_{u,p}^{k} &= \sigma(\mathbf{W}_{\varphi f}[\mathbf{h}_{u,p}^{k-1}, \mathbf{z}_{u,p}^{k}]), \\
\mathbf{c}_{u,p}^{k} &= \mathbf{f}_{u,p}^{k-1} \odot \mathbf{c}_{u,p}^{k-1} + \mathbf{i}_{u,p}^{k} \odot \tanh(\mathbf{W}_{\varphi c}[\mathbf{h}_{u,p}^{k-1}, \mathbf{z}_{u,p}^{k}]), \\
\mathbf{o}_{u,p}^{k} &= \sigma(\mathbf{W}_{\varphi o}[\mathbf{h}_{u,p}^{k-1}, \mathbf{z}_{u,p}^{k}]), \\
\mathbf{h}_{u,p}^{k} &= \mathbf{o}_{u,p}^{k} \odot \tanh(\mathbf{c}_{u,p}^{k}),
\end{aligned}$$

where $\mathbf{i}_{u,p}, \mathbf{f}_{u,p}, \mathbf{o}_{u,p}$ are input gate, forget gate and output gate in LSTM respectively. $\mathbf{W}_{\varphi i}, \mathbf{W}_{\varphi f}, \mathbf{W}_{\varphi o}$ are learnable parameters and $\mathbf{c}_{u,p}$ is the cell state with $\mathbf{c}_{u,p}^{0} = \mathbf{0}$. After $H$ step, the final broadcasted embedding $\mathbf{h}_{u,p}^{H}$ is taken as the output which may contains the structure and attribute information from node $u$ to node $p$.

## 2.4 Aggregation Operation

Different from conventional GCN-based model, the aggregation operation in PaGNN not only recursively aggregates neighbor attributes but also aims to "receive" the broadcasted embeddings from the other node. Suppose information is broadcasted from $u$ and aggregated to $v$, we will introduce details of aggregation operation, as illustrated in Fig 3.

First, node $u$ broadcasts information to nodes among its centralized subgraph $\mathcal{G}_u^H$, then $\mathbf{h}_{u,p}^{H}$, $\forall p \in \mathcal{V}_u^H$ is obtained. Afterwards, nodes in $v$'s centralized subgraph $\mathcal{G}_v^H$ aggregate broadcasted embeddings and initial node attribute to $v$. Before aggregation, initial embedding $\mathbf{r}_{u,p}^{0}$ is set to combination of its broadcasted embedding and initial node attributes. In particular, if there is no path between $u$ and $p$, the broadcasted information is set to $\mathbf{0}$, which is defined as:

$$\mathbf{r}_{u,p}^{0} = \begin{cases} [\mathbf{h}_{u,p}^{H}, \mathbf{x}_p], & p \in \mathcal{V}_v^H \cap \mathcal{V}_u^H \\ [\mathbf{0}, \mathbf{x}_p], & p \in \mathcal{V}_v^H - \mathcal{V}_u^H \end{cases}, \ \forall p \in \mathcal{V}_v^H \tag{3}$$

At last, for each node $p \in \mathcal{V}_v^H$, $p$ aggregates information from its neighbors in a GCN style:

$$\mathbf{r}_{u,p}^{k} \leftarrow AGG(\mathbf{r}_{u,p}^{k-1}, \{\mathbf{r}_{u,i}^{k-1} \mid (i,p) \in \mathcal{E}_v^H\}), \ \forall p \in \mathcal{V}_v^H \tag{4}$$

where $AGG$ is the aggregation function, $\mathbf{r}_{u,v}^{H}$ represents information that broadcasted from $u$ and aggregated to $v$ at H-th step is taken as output.

## 2.5 Edge Representation Learning

With the broadcasting and the aggregation operation mentioned above, edge $\langle u, v\rangle$ is represented from two ways. PaGNN first broadcasts information from $u$ among $\mathcal{G}_u^H$ and aggregates information to $v$ among $\mathcal{G}_v^H$, $\mathbf{r}_{u,v}^{H}$ is obtained. Meanwhile, we broadcast information from $v$ among $\mathcal{G}_v^H$ and aggregate to $u$ among

$\mathcal{G}_u^H$, $\mathbf{r}_{v,u}^H$ is obained. Concatenation of two embeddings is taken as the edge representation:

$$\mathbf{s}_{u,v} = [\mathbf{r}_{u,v}^H, \mathbf{r}_{v,u}^H] \tag{5}$$

***Loss Function***. On the basis of edge representation, we employ a cross entropy loss based on the edge representation:

$$loss = -\frac{1}{|\mathcal{L}|} \sum_{(\langle u,v \rangle, y) \in \mathcal{L}} y \log(\hat{y}) + (1-y) \log(1-\hat{y}), \tag{6}$$

Where $\hat{y} = MLP(\mathbf{s}_{u,v})$ and $MLP(\cdot)$ is a multi-layer perception with two fully-connected layers.

## 2.6 Cache Strategy in Inference

As PaGNN is operated on subgraph of each candidate edge, it's more time-consuming than node-centric model. To address this, we design a cache mechanism to accelerate the inference.

For two associated nodes $\langle u, v \rangle$, since $u$ only broadcasts information to neighbors among its ego subgraph $\mathcal{G}_u^H$, it's obviously that no matter where $v$ is, the broadcasted embeddings of $u$ are the same. In other words, different $v$ doesn't affect any broadcasted embeddings of $u$. Base on this property, we can pre-calculate broadcasted embeddings $\mathbf{h}_{u,p}^H$ for $u$ and cache it in storage. When $u$ appears as a targeted node in inference, the cached broadcasted embeddings can be reused. As a result, only aggregation operation is necessary by leveraging cache strategy.

***Time Complexity***. For PaGNN, suppose information is broadcasted from $u$ and aggregated to $v$, the time complexity is $O(H(|\mathcal{E}_u^H| + |\mathcal{V}_u^H|))$ for broadcasting and $O(H(|\mathcal{E}_v^H| + |\mathcal{V}_v^H|))$ for aggregating. Based on above analysis, the overall time complexity for training and inference stage is $O(H(|\mathcal{E}_u^H| + |\mathcal{V}_u^H| + |\mathcal{V}_v^H| + |\mathcal{V}_v^H|))$.

With the cache strategy, the inference time complexity decreases to $O(H(|\mathcal{E}_v^H| + |\mathcal{V}_v^H|))$, since only aggregation operation is required. As a result, our cache strategy has a nearly two times speed-up for PaGNN.

## 2.7 Summary

Comparing with other GNN models, PaGNN integrates the broadcasted embeddings into the aggregation process (Equation 3 and 4). The broadcasting mechanism guarantees that information is broadcasted from a subset of nodes $\mathcal{N}_s$, and other nodes in $\mathcal{N}_d$ can only absorb information from nodes in the subset. This mechanism guarantees that we only aggregate the information from the path between two target nodes.

On one hand, nodes on the overlap of two ego-subgraphs ($p \in \mathcal{V}_v^H \cap \mathcal{V}_u^H$) not only receive broadcasted information from $u$, but also aggregate this information to $v$, these nodes are called "bridge nodes". And if $p \in \mathcal{V}_v^H - \mathcal{V}_u^H$, the "interaction information" is set to $\mathbf{0}$, indicating that node $p$ isn't on any path between $u$ and

$v$, which is also useful to label the node role. Therefore node $v$ ***integrates all the structure and attribute information from interaction*** between $u$ and $v$ through the "bridge nodes". On the other hand, all nodes in $\mathcal{G}_v^H$ aggregate its attributes to $v$, thus ***representation of node $v$ also embeds the structure and attribute information from its local neighborhood***.

In summary, PaGNN jointly learns structure and attribute information from both interaction and local neighborhood. And since PaGNN is operated on subgraphs of associated nodes, it provides inductive ability.

## 3 Experiments

### 3.1 Experiment setup

**Data Protection Statement** (1) The data used in this research does not involve any **P**ersonal **I**dentifiable **I**nformation (PII). (2) The data used in this research were all processed by data abstraction and data encryption, and the researchers were unable to restore the original data. (3) Sufficient data protection was carried out during the process of experiments to prevent the data leakage and the data was destroyed after the experiments were finished. (4) The data is only used for academic research and sampled from the original data, therefore it does not represent any real business situation in Ant Financial Services Group.

**Datasets** We adopt four real-world dataset from different domains to evaluate the effectiveness of our model, consisting of **Collab** [1] and **PubMeb** [30] from bibliographic domain, **Facebook** [2] from social domain and **SupChain** [29] from E-commerce domain. The statistics of these datasets are illustrated in Table 1.

**Table 1.** The Statistics of the Datasets.

| **Datasets** | **PubMed** | **Facebook** | **Collab** | **SupChain** |
|---|---|---|---|---|
| Nodes | 19.7K | 4.0K | 235.8K | 23.4M |
| Edges | 44.3K | 66.2K | 1.2M | 103.2M |
| Node features | 500 | 161 | 53 | 95 |

**Baseline** We compare our proposal with following three categories of link prediction methods:

***Heuristic methods***. Two heuristic methods are implemented: the **C**ommon **N**eighbors (CN) [13] and the Jaccard [19].

[1] https://snap.stanford.edu/ogb/data/linkproppred
[2] https://snap.stanford.edu/data/egonets-Facebook.html

***Network embedding methods***. We include two network embedding methods, DeepWalk [14] and Node2vec [5]. Network embedding based methods are implemented based on open source code [3], which can not be applied to large scale graph, as a consequence, experiment on dataset *SupChain* is unable to be conducted.

***Node-centric GNN methods***. We take **PinSage** [31] as baseline, which encodes node information via a GNN model, and then predicts the edges based on a pairwise decoder (a MLP layer). In our experiment, **GCN** and **GAT** [21] models are chosen as encoders, which are represented as $\mathbf{PinSage}_{GCN}$, $\mathbf{PinSage}_{GAT}$.

***Edge-centric GNN methods***. We also compare our model with link prediction models that learn high order interactive structure on targeted edge's subgraph, such as SEAL [33], and GraIL [20]. Note that, SEAL and GraIL not only label node roles but also *integrate it with node attributes.*

***Ablation studies***. To study the effectiveness of broadcasted embeddings, we remove it before information aggregation, i.e. Equation 3 is set as $\mathbf{r}^0_{u,p} = [\mathbf{0}, \mathbf{x}_p]$. This method is represented as $\mathbf{PaGNN}_{broadcast}$. We further quantitatively analyze whether only broadcasting information from one node is enough for predicting the edges, i.e., Equation 5 is changed to $\mathbf{s}_{u,v} = \mathbf{r}^H_{u,v}$, it's represented as $\mathbf{PaGNN}_{two\_way}$. And in order to demonstrate effectiveness of LSTM_OP in broadcasting, we change LSTM_OP to concatenation, i.e. $\mathbf{h}^k_{u,p} = [\mathbf{h}^{k-1}_{u,p}, \mathbf{z}^k_{u,p}]$ in Algorithm 1 (line 8), and it is represented as $\mathbf{PaGNN}_{lstm}$.

Specifically, *GAT* is chosen as the aggregation function for all edge-centric GNN methods.

**Parameter Setting** For all GNN models, we set the scale of the enclosing subgraph $H$ to 2, embedding size to 32, and other hyper-parameters are set to be the same. For all labeled candidate edges, we randomly sampled 75% of the them as the training set, 5% as the validation set, and 20% as the test set. The *negative injection trick* mentioned in previous work [1] is also employed, i.e. the negative samples are also inserted into original graph $\mathcal{G}$. We adopt Adam optimizer for parameter optimization with an initial learning rate 0.001. The other hyper-parameters are set to be the same. All GNN based models are trained on a cluster of 10 Dual-CPU servers with AGL [32] framework.

**Metric** In the link prediction task, we adopt the Area Under Curve (AUC) and F1-score as metrics to evaluate all these models, as done in many other link prediction work.

## 3.2 Performance Comparison

Table 2 summarizes the performance of the proposed method and other methods on four datasets. Based on the results of the experiment, we summarize the following points:

[3] https://github.com/shenweichen/GraphEmbedding

**Table 2.** Performance comparison on four Datasets

| Types | Model | AUC | | | | F1 | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Facebook | PubMed | Collab | SupChain | Facebook | PubMed | Collab | SupChain |
| Heuristic | **CN** | 0.927 | 0.662 | 0.771 | 0.601 | 0.869 | 0.492 | 0.703 | 0.593 |
| Heuristic | **Jaccard** | 0.938 | 0.630 | 0.801 | 0.622 | 0.874 | 0.527 | 0.719 | 0.601 |
| NE | **DeepWalk** | 0.884 | 0.842 | 0.864 | - | 0.826 | 0.826 | 0.811 | - |
| NE | **Node2vec** | 0.902 | 0.897 | 0.857 | - | 0.855 | 0.867 | 0.821 | - |
| Node | **PinSage**$_{GCN}$ | 0.924 | 0.823 | 0.851 | 0.941 | 0.900 | 0.766 | 0.910 | 0.763 |
| Node | **PinSage**$_{GAT}$ | 0.917 | 0.832 | 0.833 | 0.968 | 0.902 | 0.774 | 0.909 | 0.822 |
| Edge | **SEAL** (attributed) | 0.963 | 0.898 | 0.909 | 0.977 | 0.915 | 0.841 | 0.923 | 0.862 |
| Edge | **GraIL** (attributed) | 0.971 | 0.904 | 0.947 | 0.979 | 0.928 | 0.848 | 0.964 | 0.864 |
| Edge | **PaGNN**$_{two\_way}$ | 0.909 | 0.762 | 0.902 | 0.976 | 0.820 | 0.720 | 0.921 | 0.861 |
| Edge | **PaGNN**$_{broadcast}$ | 0.940 | 0.866 | 0.853 | 0.970 | 0.917 | 0.790 | 0.910 | 0.848 |
| Edge | **PaGNN**$_{lstm}$ | 0.969 | 0.934 | 0.958 | 0.978 | 0.932 | 0.852 | 0.976 | 0.868 |
| Edge | **PaGNN** | **0.972** | **0.944** | **0.967** | **0.987** | **0.933** | **0.878** | **0.979** | **0.897** |

***Heuristic-based methods*** achieve considerable performance on the social network dataset (i.e., Facebook), but poor performance on the other three datasets. It indicates that the assumptions of heuristic methods are not applicable in many types of network data. Compared with the heuristic method, the ***network embedding methods*** achieve better results, but the effectiveness are worse than ***node-centric GNN models***. One reason is that the GNN method straightforwardly leverages the node attributes and learns representations in a supervised manner.

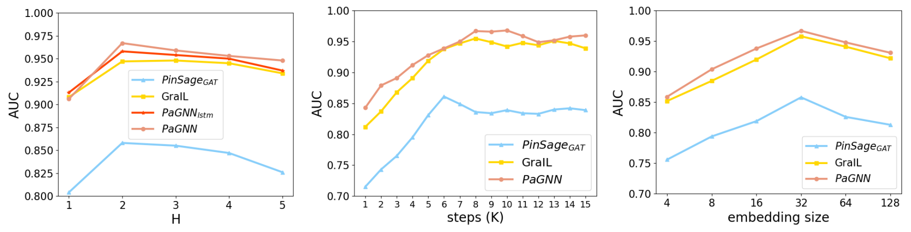
All ***edge-centric GNN models*** achieve better performance since they take the interaction structure into account. And **PaGNN** performs better than the other edge-centric GNN models: **SEAL** and **GraIL**, although they have combined high-order interactions with node attributes. This is because **PaGNN** explicitly incorporates interaction and node attribute information into GNN. Labeling nodes with the structure may not achieve satisfied performance when the attribute on path is crucial.

From the experimental results of ***ablation studies***, **PaGNN**$_{two\_way}$ is worse than **PaGNN** and it's unstable on some datasets. It is due to when there is no path between two nodes, the final representation only contains local neighborhood information of one node, performance of **PaGNN**$_{two\_way}$ drops significantly. For example, PubMed data is relatively sparse and only 18.2% samples are connected, the performance of **PaGNN**$_{two\_way}$ is poor. Comparing the performance of **PaGNN** and **PaGNN**$_{\mathbf{lstm}}$, it can be observed that LSTM_OP is effective to integrate the interactions and node attributes, since LSTM forgets useless information. In particular, by observing results of **PaGNN**$_{broadcast}$ and **PaGNN**, the broadcasted embeddings significantly improve the performance, which verify the effectiveness of the broadcasting operation.

On average, **PaGNN** improved upon the best baseline of every dataset by 1.7% in AUC and by 1.4% in F1. In summary, **PaGNN** outperforms all baselines, and every studied component has a positive impact on final results.

**Parameter Sensitivity** We also evaluate model performance on the Collab dataset as $H$ ranges from 1 to 5, shown in Fig 4(a). All models achieved the best performance when H=2. The AUC value decreases when $H > 2$, which indicates a large $H$ may bring in noises from distant nodes. **PaGNN**$_{lstm}$ shows better performance than **PaGNN** when H=1, since the LSTM_OP has no advantage when path is short. When $H$ becomes larger, **PaGNN** shows better performance, since LSTM forgets useless path information of distant nodes.

The convergence of different models is also evaluated. Fig 4(b) records the validation AUC of varying training steps. **PinSage**$_{GAT}$ first achieves the best performance at 6K training steps, as it learns minimum parameters. **GraIL** and **PaGNN** need more time to be converged, which takes about 8K to 10K training steps. Fig 4(c) compares the performance of varying node embedding size, three models are over-fitting when embedding size is larger than 32.

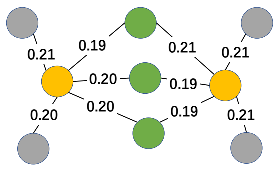


(a)Testing AUC - $H$ (b)Validation AUC - Steps (c)AUC - Embedding Size

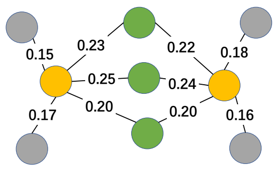
**Fig. 4.** Parameter Sensitivity.

**Case Study** . To examine whether **PaGNN** learns interaction information, we illustrate an example of *Facebook* dataset in Fig 5, the goal is to predict whether two yellow nodes exist an edge, nodes on the path between them are colored with green and other nodes are colored gray, and the value is the attention of corresponding edge in last aggregation step. We can learn that the neighbor attention output by **PinSage**$_{GAT}$ between associated node and its neighbors is almost the same, without regard to where the neighbor is. Attention value of nodes on the path of edge-centric methods is larger, which indicates they have ability to find the pattern that nodes on path are more important. And comparing with **GraIL**, attention value of green nodes for **PaGNN** is larger, shown in Fig 5(b) and (c).

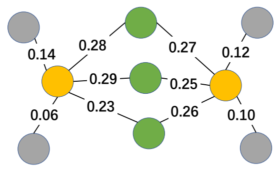
To further prove the impact of paths, in *Facebook* and *SupChain* dataset, we categorize candidate relations into different groups according to the number of paths between two target persons or enterprises, proportion of each group is illustrated in Fig 6. It can be learned that observed friends in social network and enterprises with supply-chain relationship tend to have more paths, which drives to effectively capture fine-grained interactions between target node pairs.



(a) **PinSage**$_{GAT}$

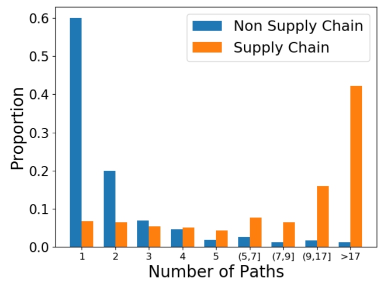


(b) **GraIL**

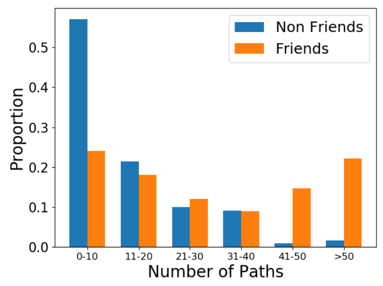


(c) **PaGNN**

**Fig. 5.** Attention of Different Models.



(a) SuppChain



(b) Facebook

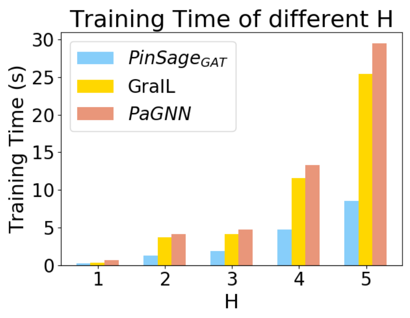
**Fig. 6.** Relation distribution of different path number.

## 3.3 Efficiency Analysis

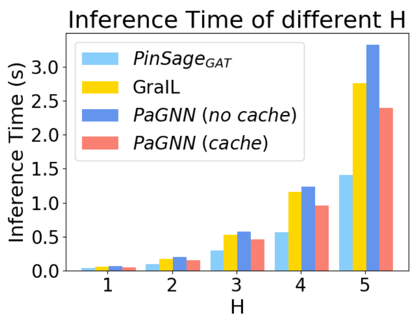
**Training Phase** We calculate the training time of three models with varying $H$ on Collab dataset (Fig 7(a)). The training time increases rapidly as $H$ increases, and all edge-centric models are more time-consuming than node-centric models. **PaGNN** takes about 1.5 longer the training time comparing with **PinSage**$_{GAT}$ when H=1, but about 3.5 times the training time when H=5. It is due to that as $H$ increases, the subgraphs become larger, for node-centric methods, different edges' subgraphs in same batch share more common nodes, which avoids duplicate computation comparing with edge-centric methods.

**Inference Phase** Efficiency evaluation of inference phase is illustrated in Fig 7(b). It can be inferred that edge-centric methods need more time. For example, **PaGNN** takes 2.4 times longer than **PinSage**$_{GAT}$ when H=5, and **GraIL** takes 2.0 times longer. Fortunately, the cache strategy significantly improves the efficiency of **PaGNN**, which takes 1.7 times longer than **PinSage**$_{GAT}$ when H=5 and has a 30% speed-up compared to **PaGNN** without cache strategy. In particular, although the time complexity of cache strategy analysed in Section 2.6 has a two times speed-up theoretically, the statistics reported here are the time cost of the whole inference phase (also including subgraph extraction, input data preparing).

To summarize, comparing with node-centric models (e.g. PinSage), edge-centric models (e.g. PaGNN, SEAL, GraIL) achieve better performance despite



(a) Training Time - $H$.



(b) Inference Time - $H$.

**Fig. 7.** Training and Inference Time per Step.

being more time-consuming, since edge-centric models consider high-order interactive information. Nevertheless, our proposal with the cache strategy takes less time comparing with other edge-centric models (SEAL and GraIL).

## 4 Related Work

Previous work for link prediction can be divided into the heuristic methods, network embedding, and supervised methods. Heuristic methods usually assume that nodes with links satisfy some specific properties. For instance, common neighbor [13] defines the similarity as the number of shared neighbors of two nodes. Katz index [9] calculates similarity by counting the number of reachable paths. The rooted PageRank [2] calculates the stationary distribution from one node to other nodes through random walk. These heuristic methods rely on hand-crafted rules and have strong assumptions, which can not be applied to all kinds of networks.

Recently, a number of researchers propose to perform link prediction through constructing node latent features, which are learned via classical matrix factorization [10, 18] and shallow graph representation learning (e.g., Deepwalk [14] and Node2Vec [5]). And there is also research [33, 20] inductively predicts node relations by leveraging graph neural networks, which combine high-order topological and initial features in the form of graph patterns. However, it learns high-order information and node attributes seperately. Some literature [16, 4, 34, 15, 8] also considers integrating target behavior information in a heterogeneous graph, which requires nodes and edges belong to a specific type. Reseach work [27] also employs characteristics of path reachability to represent high-order information, but fails to integrate the path structure and attributes.

## 5 Conclusion

In this paper, we aim at simultaneously leveraging the structure and attribute information from both interactions/paths and local neighborhood, to predict the edge between two nodes. A novel PaGNN model is proposed which first broadcasts information from one node, afterwards aggregates broadcasted embeddings and node attributs to the other node from its ego subgraph. PaGNN inductively learns representation from node attributes and structures, which incorporates high-order interaction and neighborhood information into GNN. And we also employ a cache strategy to accelerate inference stage. Comprehensive experiments show the effectiveness and efficiency of our proposal on real-world datasets.

## References

1. AbuOda, G., Morales, G.D.F., Aboulnaga, A.: Link prediction via higher-order motif features. In: ECML PKDD. pp. 412–429 (2019)
2. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer networks **56**(18), 3825–3833 (2012)
3. Cui, P., Wang, X., Pei, J., Zhu, W.: A survey on network embedding. IEEE Transactions on Knowledge and Data Engineering **31**(5), 833–852 (2018)
4. Feng, Y., Hu, B., Lv, F., Liu, Q., Zhang, Z., Ou, W.: Atbrg: Adaptive target-behavior relational graph network for effective recommendation (2020)
5. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: SIGKDD. pp. 855–864 (2016)
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
7. Hu, B., Hu, Z., Zhang, Z., Zhou, J., Shi, C.: Kgnn: Distributed framework for graph neural knowledge representation. In: ICML Workshop (2020)
8. Hu, B., Shi, C., Zhao, W.X., Yu, P.S.: Leveraging meta-path based context for top-n recommendation with a neural co-attention model. In: SIGKDD. pp. 1531–1540 (2018)
9. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
10. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
11. Kovács, I.A., Luck, K., Spirohn, K., Wang, Y., Pollis, C., Schlabach, S., Bian, W., Kim, D.K., Kishore, N., Hao, T., et al.: Network-based prediction of protein interactions. Nature communications **10**(1), 1–8 (2019)
12. Kumar, A., Singh, S.S., Singh, K., Biswas, B.: Link prediction techniques, applications, and performance: A survey. Physica A: Statistical Mechanics and its Applications **553**, 124289 (2020)
13. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. Journal of the American society for information science and technology **58**(7), 1019–1031 (2007)
14. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: SIGKDD. pp. 701–710 (2014)
15. Sankar, A., Zhang, X., Chang, K.C.C.: Meta-gnn: Metagraph neural network for semi-supervised learning in attributed heterogeneous information networks. In: ASONAM. pp. 137–144 (2019)
16. Sha, X., Sun, Z., Zhang, J.: Attentive knowledge graph embedding for personalized recommendation. arXiv preprint arXiv:1910.08288 (2019)

17. Shi, B., Weninger, T.: Proje: Embedding projection for knowledge graph completion. In: AAAI. vol. 31 (2017)
18. Shi, C., Hu, B., Zhao, W.X., Philip, S.Y.: Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering **31**(2), 357–370 (2018)
19. Shibata, N., Kajikawa, Y., Sakata, I.: Link prediction in citation networks. Journal of the American society for information science and technology **63**(1), 78–85 (2012)
20. Teru, K., Denis, E., Hamilton, W.: Inductive relation prediction by subgraph reasoning. In: ICML. pp. 9448–9457 (2020)
21. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
22. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: SIGKDD. pp. 1225–1234 (2016)
23. Wang, H., Lian, D., Zhang, Y., Qin, L., Lin, X.: Gognn: Graph of graphs neural network for predicting structured entity interactions pp. 1317–1323 (2020)
24. Wang, Z., Liao, J., Cao, Q., Qi, H., Wang, Z.: Friendbook: a semantic-based friend recommendation system for social networks. IEEE transactions on mobile computing **14**(3), 538–551 (2014)
25. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems (2020)
26. Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. arXiv preprint arXiv:1906.00121 (2019)
27. Xu, C., Cui, Z., Hong, X., Zhang, T., Yang, J., Liu, W.: Graph inference learning for semi-supervised classification. arXiv preprint arXiv:2001.06137 (2020)
28. Xu, N., Wang, P., Chen, L., Tao, J., Zhao, J.: Mr-gnn: Multi-resolution and dual graph neural network for predicting structured entity interactions pp. 3968–3974 (2019)
29. Yang, S., Zhang, Z., Zhou, J., Wang, Y., Sun, W., Zhong, X., Fang, Y., Yu, Q., Qi, Y.: Financial risk analysis for smes with graph-based supply chain mining. In: IJCAI. pp. 4661–4667 (2020)
30. Yang, Z., Cohen, W., Salakhudinov, R.: Revisiting semi-supervised learning with graph embeddings. In: ICML. pp. 40–48 (2016)
31. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: SIGKDD. pp. 974–983 (2018)
32. Zhang, D., Huang, X., Liu, Z., Hu, Z., Song, X., Ge, Z., Zhang, Z., Wang, L., Zhou, J., Qi, Y.: Agl: a scalable system for industrial-purpose graph machine learning. arXiv preprint arXiv:2003.02454 (2020)
33. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: NeurIPS. pp. 5165–5175 (2018)
34. Zhang, W., Fang, Y., Liu, Z., Wu, M., Zhang, X.: mg2vec: Learning relationship-preserving heterogeneous graph representations via metagraph embedding. IEEE Transactions on Knowledge and Data Engineering (2020)