RESEARCH ARTICLE

Can GNNs Learn Link Heuristics? A Concise Review and Evaluation of Link Prediction Methods

Shuming Liang[1], Yu Ding[2], Zhidong Li[1], Bin Liang[1], Siqi Zhang[3], Yang Wang[1], Fang Chen[1]

1 University of Technology Sydney, Australia
2 University of Wollongong, Australia
3 Zhejiang University, China

Abstract

This paper explores the ability of Graph Neural Networks (GNNs) in learning various forms of information for link prediction, alongside a brief review of existing link prediction methods. Our analysis reveals that GNNs cannot effectively learn structural information related to the number of common neighbors between two nodes, primarily due to the nature of set-based pooling of the neighborhood aggregation scheme. Also, our extensive experiments indicate that trainable node embeddings can improve the performance of GNN-based link prediction models. Importantly, we observe that the denser the graph, the greater such the improvement. We attribute this to the characteristics of node embeddings, where the link state of each link sample could be encoded into the embeddings of nodes that are involved in the neighborhood aggregation of the two nodes in that link sample. In denser graphs, every node could have more opportunities to attend the neighborhood aggregation of other nodes and encode states of more link samples to its embedding, thus learning better node embeddings for link prediction. Lastly, we demonstrate that the insights gained from our research carry important implications in identifying the limitations of existing link prediction methods, which could guide the future development of more robust algorithms.

Corresponding author: Shuming Liang, shuming.liang@uts.edu.au

1. Introduction

Graph Neural Networks (GNNs) have demonstrated powerful expressiveness in graph representation learning[1][2]. However, what structural information can be learned via GNN remains an open question[3][4][5][6][7][8]. Particularly, scant attention has been directed towards this question in terms of structural information specific to two nodes. A prominent task relying on such the information is link prediction. Although a number of GNN-based link prediction models have been introduced[9][10][11][12][13][14][15][16], many of them lack thorough investigations into whether their models effectively learn pair-specific structural information.

For example, SEAL[9] and its successors[17][18][19][20] are a family of link prediction methods that attempt to use GNNs to learn pair-specific structural information represented by traditional link heuristics such as Common Neighbors, Katz index[21], etc. SEAL[9] has proven that most link heuristics between two nodes in a graph can be computed approximately within an enclosing subgraph specifically constructed for that two nodes. In essence, SEAL-type methods perform link prediction by using GNNs to classify such enclosing subgraphs, with an expectation that GNNs could learn the structural information equivalent to link heuristics. However, a critical evaluation of this expectation is lacking in the literature. We demonstrate that this expectation does not hold completely.

In this paper, we mainly study the link prediction capability of GNNs, with a focus on three aspects. 1), we explore whether GNNs can effectively learn the pair-specific structural information related to the number of common neighbors for link prediction. 2), we present our experimental observation: incorporating trainable node embeddings can improve the performance of GNN-based link prediction models, and the denser the graph, the greater the improvement. This observation, not extensively revealed in the prior literature, has significant practical implications for selecting appropriate methods based on graph density in real-world link prediction problems. 3), we leverage insights derived from our research to provide a limitation analysis of existing link prediction methods, thereby contributing valuable perspectives for their potential improvements.

First, the majority of GNNs follow a neighborhood aggregation scheme, where each node's representation is recursively updated by aggregating the representations of that node and its neighbors[22][15][2]. The learned representations are node-wise. It has been recognized that every node's representation can hardly capture information related to the number of its neighbors. This is due to the nature of the set-based pooling of the aggregation scheme, which inherently ignores the size of the neighborhood set of each node[3][23].

A general strategy for applying node-wise representations learned by GNNs to downstream multiple-node tasks (e.g., link prediction, graph classification, etc.) is to combine the representations of the nodes involved in these tasks. For link prediction, we find that the combination of two nodes' representations essentially lacks the ability to capture information related to the number of common neighbors. This is mainly because node-wise representations learned by GNNs inherently lack information about the number of neighbors of each node, and most operations of combining two nodes' representations (e.g., concatenation, Hadamard production, etc.) also do not contain any behaviors of counting how many common neighbors between two nodes. To empirically verify the above, we examine the link prediction performance of an approach that incorporates traditional link heuristics (e.g., Common Neighbors) into the GNN. The approach yields results either superior or comparable to those obtained by using only GNNs, experimentally supporting our analysis.

Moreover, in our experiments, we find that trainable node embeddings (different from pre-trained node embeddings, we refer to trainable node embeddings as those embeddings that can be optimized during the model training) can enhance the performance of GNN-based link prediction models, and the denser the graph, the stronger the enhancement. In particular, by only utilizing node embeddings in GCN[24] or GAT[25], we are able to surpass many link prediction specific methods on two dense graphs, i.e., ogbl-ddi and ogbl-ppa[26]. Our explanation is as follows. Compared to the model weights of a GNN that are shared across all nodes[24][22][2], each trainable node embedding is unique to its respective

node. This characteristic of node embeddings can benefit the model. When the training is supervised by positive and negative link samples (i.e., two nodes are not linked), the link state of two nodes in every link sample could be encoded into the node embeddings of that two nodes and their neighboring nodes by the neighborhood aggregation algorithm of the GNN. This would enable each node embedding to remember the relationships of that node to other nodes, allowing the model to know better which two nodes are more likely to be linked or not. Moreover, in the neighborhood aggregation of the GNN, the denser graphs would allow each node to see more other nodes, leading to better learning of node embeddings for link prediction.

The insights gained in this study can help identify and interpret the limitations of existing link prediction methods, potentially directing the search for more robust algorithms. To demonstrate this, we present two case studies: first, we show that SEAL-type methods[9][18][19][20] could not effectively learn information about the number of common neighbors. Second, we show that NBFNet[27] lacks the algorithmic capability to train powerful node embeddings for link prediction. Additionally, we compare the empirical performance of various link prediction methods on OGB datasets. The results can be explained with our insights.

2. Notations and Problem Definition

Without loss of generality, we demonstrate our work on homogeneous graphs. Let $G = (V, E, \mathbf{X})$ denote a graph G with N nodes, where V is the set of nodes, $|V| = N$, E is the set of edges, and $\mathbf{X} \in R^{N \times f}$ is the feature matrix of nodes. The i-th row of $\mathbf{X}$ (i.e., $\mathbf{x}_i \in R^f$) is the feature vector of node i. The adjacency matrix is $\mathbf{A} \in R^{n \times n}$ in which the i, j-th entry (i.e., $a_{i,j}$) is 1 if an edge exists from node i to j and 0 otherwise. The degree of node i is $\deg(i) = \sum_{j \in V} a_{i,j}$. The degree of the graph G is the average degree of all nodes. A set of nodes connected directly to a node $v \in V$ is the first-order or 1-hop neighborhood set of v and is denoted by Γ_v.

Link prediction is a node-pair-specific problem, aiming to estimate the likelihood $\hat{y}_{v,u}$ of the existence of an unknown edge $E_{v,u} \notin E$ between two nodes $v, u \in V$. Herein we refer to v, u as *two target nodes* in the *candidate link* $E_{v,u}$.

3. Statistical Link Heuristics

In the long history of link prediction research, especially prior to the emergence of neural networks, a variety of statistical link prediction methods have been proposed[28][29][30]. These statistical methods, known as heuristic methods or link heuristics, often rely on intuitive rules or empirical observations, and often extract structural information specific to the target node pair. Therefore, we can concrete abstract pair-specific structural information using tangible link heuristics.

In this work, based on whether a link heuristic captures information related to the Number of Common Neighbors (**NCN**) between two target nodes or not, we categorize pair-specific link heuristics (structural information) into two types: *NCN-dependent* and *non-NCN-dependent*. With this categorization, we offer a concise review of link heuristics in the literature, which reveals that the majority of these heuristics are NCN-dependent.

3.1. NCN-dependent Link Heuristics

Table 1. NCN-dependent link heuristics between nodes v, u.

Heuristic	Definition
$CN_{v,u}$	$\lvert \Gamma_v \cap \Gamma_u \rvert$
$JA_{v,u}$[31]	$\dfrac{\lvert \Gamma_v \cap \Gamma_u \rvert}{\lvert \Gamma_v \cup \Gamma_u \rvert}$
$AA_{v,u}$[32]	$\sum_{z \in \Gamma_v \cap \Gamma_u} \dfrac{1}{\log \lvert \Gamma_z \rvert}$
$RA_{v,u}$[33]	$\sum_{z \in \Gamma_v \cap \Gamma_u} \dfrac{1}{\lvert \Gamma_z \rvert}$
Sorensen index[34]	$\dfrac{2\lvert \Gamma_v \cap \Gamma_u \rvert}{\lvert \Gamma_v \rvert + \lvert \Gamma_u \rvert}$
Salton index[35]	$\dfrac{\lvert \Gamma_v \cap \Gamma_u \rvert}{\sqrt{\lvert \Gamma_v \rvert \cdot \lvert \Gamma_u \rvert}}$
Hub Promoted index[36]	$\dfrac{\lvert \Gamma_v \cap \Gamma_u \rvert}{min(\lvert \Gamma_v \rvert, \lvert \Gamma_u \rvert)}$
Hub Depressed index[36]	$\dfrac{\lvert \Gamma_v \cap \Gamma_u \rvert}{max(\lvert \Gamma_v \rvert, \lvert \Gamma_u \rvert)}$

Table 1 lists several commonly-used NCN-dependent Link heuristics. As shown, Common Neighbors (CN) is defined as the size of the intersection of the first-order neighborhood sets of two nodes. Jaccard (JA) coefficient[31] normalizes the CN by the size of the union of the two nodes' neighborhood sets. AdamicAdar (AA)[32] and Resource Allocation (RA)[33] suppress the contribution of nodes by penalizing each node with its degree. Sorensen index[34], Salton index[35], Hub Promoted index[36], and Hub Depressed index[36] incorporate the degree of two target nodes with CN. We can see from Table 1 that these heuristics are highly dependent on the number of common neighbors between two nodes v, u (i.e., $\lvert \Gamma_v \cap \Gamma_u \rvert$).

In addition to the above, many other link heuristics are NCN-dependent. Cannistraci et al.[37] suggest that the likelihood of two nodes forming a link increases if their common neighbors are members of a strongly inner-linked cohort, termed local-community-links. They introduce a modified version of CN, JA, AA, and RA, denoted as CAR-based. Furthermore, some another link heuristics consider the clustering coefficient of the nodes in counting common neighbors, such as node clustering coefficient ($\sum_{z \in \Gamma_v \cap \Gamma_u} C(z)$)[38] and node-link clustering coefficient ($\sum_{z \in \Gamma_v \cap \Gamma_u} \frac{\lvert \Gamma_v \cap \Gamma_z \rvert}{\lvert \Gamma_z \rvert - 1} \times C(z) + \frac{\lvert \Gamma_u \cap \Gamma_z \rvert}{\lvert \Gamma_z \rvert - 1} \times C(z)$)[39], where $C(z) = \dfrac{2\lvert E_{j,k} : j, k \in \Gamma_z, E_{j,k} \in E \rvert}{\lvert \Gamma_z \rvert \times (\lvert \Gamma_z \rvert - 1)}$ is the local clustering coefficient of the node z.

An important family of link heuristics is those counting all paths between two target nodes, such as Katz index[21], Leicht Holme Newman index[40], and Rooted PageRank[9]. These heuristics are indeed NCN-dependent, where first-order and

high-order common neighbors are considered. For example, Katz index ($Katz_{v,u} = \sum_{l=1}^{\infty} \beta^l |\{path_{v,u}^{(l)}\}|$)[21] weighted sums the number of all paths between two target nodes v, u, where $|\{path_{v,u}^{(l)}\}|$ is the number of all paths between node v and u with the length of l, and β is a damping factor. For example, if $l = 4$, $|\{path_{v,u}^{(4)}\}|$ can be computed by $|\{path_{v,u}^{(4)}\}| = \sum_{a \in \Gamma_v, b \in \Gamma_u} CN_{a,b}$, where $CN_{a,b}$ is the number of common neighbors between nodes a, b.

3.2. non-NCN-dependent Link Heuristics

According to our categorization strategy, there exist a limited number of link heuristics that are non-NCN-dependent, including Shortest Path Distance (SPD), Preferential Attachment ($PA_{v,u} = |\Gamma_v| \times |\Gamma_u|$)[41], and SimRank[42]. Notably, SPD and PA do not extract information about the number of common neighbors. SimRank, detailed in Algorithm 1, recursively refines similarity scores between every two nodes by considering the neighboring nodes of the two nodes, where the number of common neighbors is ignored essentially. Specifically, the similarity score $s_{i,j}^{(m)}$ between node i, j in the m-th iteration is obtained by averaging the similarity scores between all neighbors of i and j from the $(m-1)$-th iteration, where the information about how many common neighbors between i, j can hardly be encoded into $s_{i,j}^{(m)}$.

1: **Input:** Graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ ($|\mathbb{V}| = N$), decay factor C ($0 < C < 1$), iterations K
2: **Output:** Similarity $\mathbf{S} = (s_{i,j}) \in \mathbb{R}^{N \times N}$
3: **Initialize:** $s_{i,j}^{(0)} = 1$ if $i = j$, otherwise 0
4: **for** $m = 1$ to K **do**
5: $\quad s_{ij}^{(m)} = \frac{C}{|\Gamma_i||\Gamma_j|} \sum_{b=1}^{|\Gamma_j|} \sum_{a=1}^{|\Gamma_i|} s_{\Gamma_i(a)\Gamma_j(b)}^{(m-1)}$, where $\Gamma_i(a)$ is the a-th node in Γ_i
6: **end for**

Algorithm 1. SimRank[42]

4. Aggregation-based GNNs

Most GNNs follow a neighborhood information aggregation algorithm, where the representation of each node in a graph is iteratively updated by aggregating the representations of its neighbors and its own[22][2]. Formally, the representation of a node i updated by the l-th layer of a GNN is

$$\hat{\mathbf{h}}_i^{(l)} = \text{AGG}^{(l)}\left(\left\{\mathbf{h}_j^{(l-1)} \mid \forall j \in \Gamma_i \cup \{i\}\right\}\right),$$

$$\mathbf{h}_i^{(l)} = \hat{\mathbf{h}}_i^{(l)} \mathbf{W}^{(l)},$$

where $\mathbf{h}_i^{(0)}$ is initialized with the feature vector of node i, $\text{AGG}^{(l)}(\cdot)$ is instantiated as a set-based pooling operation such as MAX, MEAN[43], or attention-based SUM[25][15], $\mathbf{W}^{(l)}$ is a weight matrix for the l-th GNN layer, which is shared across all nodes and used for representation transformation (i.e., if $\hat{\mathbf{h}}_v^{(l)} \in \mathbb{R}^f, \mathbf{W}^{(l)} \in \mathbb{R}^{f \times f}$, then $\mathbf{h}_v^{(l)} \in \mathbb{R}^f$). For simplicity, we omit the residual connections, activation functions, etc. In this paper, we use the term GNNs to refer to such aggregation-based GNNs unless otherwise stated.
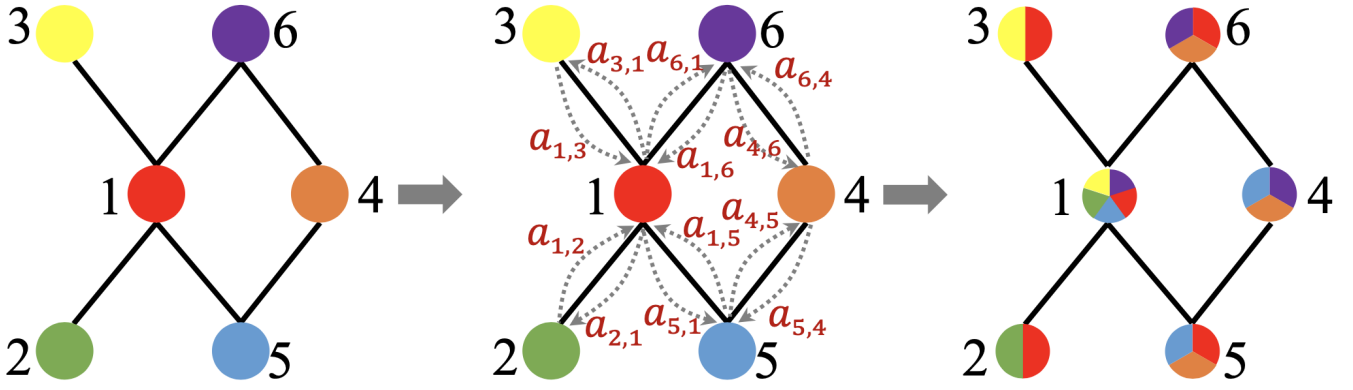
Figure 1. An illustration of neighborhood information propagation and aggregation in GNNs, where $a_{i,j}$ can be an edge weight or attention weight from node j to i.

Fig. 1 illustrates the neighborhood information propagation and aggregation process in GNNs. As shown, $\hat{\mathbf{h}}_i^{(l)}$ in Eq. 1 can be computed by

$$\hat{\mathbf{h}}_i^{(l)} = \sum_{j \in \Gamma_i \cup \{i\}} \frac{a_{i,j}^{(l)}}{\sum_{j \in \Gamma_i \cup \{i\}} a_{i,j}^{(l)}} \mathbf{h}_j^{(l-1)},$$

where $a_{i,j}^{(l)}$ is the weight for the message (i.e., $\mathbf{h}_j^{(l-1)}$) from node j to i. For MEAN-pooling in GNNs like GCN[43], it can be $a_{i,j}^{(l)} = 1, \forall \mathrm{E}_{i,j} \in \mathrm{E}$. For attention-based GNNs like GAT[25], $a_{i,j}^{(l)}$ is an attention coefficient that is computed dynamically based on $\mathbf{h}_i^{(l-1)}$ and $\mathbf{h}_j^{(l-1)}$.

Remark 1. *The node representations learned by aggregation-based GNNs are node-wise.*

Analysis. As shown in Eq. 1, the input, intermediate, and output representations of GNNs are node-wise.

Remark 2. *Each node's representation learned by neighborhood aggregation-based GNNs lacks information about the number of neighboring nodes of that node.*

Analysis. As shown in Eq. 1, GNNs update the representation $\mathbf{h}_i^{(l)}$ by aggregating the representations of node i and its neighbors. In this process, the number of neighbors of node i can hardly be encoded into $\mathbf{h}_i^{(l)}$. This is due to the inherent nature of neighborhood aggregation scheme in GNNs, i.e., the $\mathrm{AGG}^{(l)}(\cdot)$ in Eq. 1 is set-based pooling, which is originally designed to handle irregular sizes of neighborhood sets of different nodes in a graph. For example, if the aggregation is MEAN pooling, then the set of node-wise representations (i.e., $\{\mathbf{h}_j^{(l-1)} | \forall j \in \Gamma_i \cup \{i\}\}$ in Eq. 1) will be averaged and the result could hardly contain information about the size of that set. Note that attention-based pooling also cannot address this inherent issue of the neighborhood aggregation scheme. As shown in Eq. 2, attention-based GNNs[25] essentially replace the original edge weight with attention weight. Despite this modification, the set of representations is weighted averaged, and consequently, the resulting representation still lacks information related to the size of the neighborhood set.

Remark 2 shows that the neighborhood aggregation algorithm of GNNs inherently cannot effectively learn information

about the number of neighbors of each node. Essentially, we can address this issue by, for example, adding the node degree as a feature to each node. We note that several previous works have pointed out this[3][23]. We present it using Remark 2 for better presenting our following analysis.

5. Can GNNs Completely Learn NCN-dependent Structural Information?

5.1. Analytical Study

What can we do when applying the node-wise representations learned by GNNs to downstream graph tasks that involve multiple nodes, such as link prediction or graph classification? A general way is to combine the representations of the involved nodes into one representation and pass it into the subsequent model components[3][44][15][2]. For such a combination, we have the following insight:

Remark 3. *The combination of two or more nodes' representations learned by GNNs cannot effectively capture NCN-dependent structural information.*

Analysis. According to Remark 2, due to the inherent nature of the neighborhood aggregation algorithm, node-wise representations learned by GNNs cannot effectively capture information related to the number of neighbors of each node (i.e., the size of its neighborhood set), much less to the number of common neighbors between two nodes. The operation of combining representations of two or more nodes also cannot effectively extract NCN-dependent structural information. For example, we can combine the representations of two nodes by concatenation, Hadamard production[44][2], and combine more nodes' representations by MEAN pooling[3], Sort pooling[45], etc. These combination operations on node-wise representations learned by GNNs are unlikely to contain the behavior of counting the common neighbors between two nodes, thereby falling short in extracting NCN-dependent structural information.

GNNs might learn little structural information related to the number of common neighbors. However, the neighborhood aggregation algorithm of GNNs learns node-wise representations by passing the messages of neighboring nodes of each node to that node and set-based aggregates them[43][25][23][15]. Such set-based aggregation operation inherently washes out the information related to the number of nodes in the set, including the number of common nodes between two nodes. For instance, as shown in Fig. 1, the nodes 1 and 4 have common neighboring nodes $5, 6$. In GNN learning based on Eq. 1, the node 1 will receive the messages from $2, 3, 5, 6$, where the number of common neighbors (i.e., $CN_{1,4} = 2$) can hardly be captured in the aggregation of five representations (i.e., representations of nodes $1, 2, 3, 4, 5$). Note that in this example, attention-based aggregation also cannot effectively learn $CN_{1,4} = 2$ from the aggregation of five representations. The reason is the same as the analysis of Remark 2. In fact, it is difficult to interpret what information the GNN has learned in a rigorous mathematical format. Nevertheless, we could say that GNNs cannot completely capture NCN-dependent structural information.

1: **Input:** Graph $\mathcal{G} = (\mathbb{V}, \mathbb{E}, \mathbf{X})$ $(|\mathbb{V}| = N)$, $\mathbf{X} \in \mathbb{R}^{N \times f}$, trainable node embeddings $\mathbf{E} \in \mathbb{R}^{N \times d}$, trainable heuristic embeddings, ground truth $y_{v,u}$ for link sample (v, u), GNN layers L, epochs K

2: **Output:** Link likelihood $\hat{y}_{v,u} \in \mathbb{R}$ for node pair v, u

3: **Initialize:** node embeddings $\mathbf{E}$, trainable heuristic embeddings, model weights, etc.

4: **for** $i = 0$ **to** K **do**

5:     **for** $l = 1$**to** L **do**

6:         $\hat{\mathbf{h}}_i^{(l)} = \mathrm{AGG}^{(l)} \left(\left\{ \mathbf{h}_j^{(l-1)} \mid \forall j \in \Gamma_i \cup \{i\} \right\} \right)$

7:         $\mathbf{h}_i^{(l)} = \hat{\mathbf{h}}_i^{(l)} \mathbf{W}^{(l)}$

8:     **end for**

9:     $\mathbf{h}_{vu} = \mathrm{COMBINE} \left(\mathbf{h}_v^{(L)}, \mathbf{h}_u^{(L)} \right)$

10:     $\mathbf{e}_{vu} = \mathrm{CONCAT} \left(\mathbf{e}_{vu}^{(\mathrm{CN})}, \mathbf{e}_{vu}^{(\mathrm{JA})}, \cdots, \mathbf{e}_{vu}^{(\mathrm{RA})} \right)$

11:     $\hat{y}_{vu} = \mathrm{PREDICTOR} \left(\mathrm{CONCAT} \left(\mathbf{h}_{vu}, \mathbf{e}_{vu} \right) \right)$

12:     Calculate loss$(y_{v,u}, \hat{y}_{v,u})$

13:     Update $\mathbf{E}$, trainable heuristic embeddings, model weights, etc.

14: **end for**

15: Herein $\mathbf{h}_i^{(0)}$ is initialized based on the feature and node embedding of node i (i.e., $\mathbf{x}_i$ and $\mathbf{e}_i$). $\mathbf{h}_{vu}$ is the link representation for (v, u). $\mathbf{e}_{vu}^{(\mathrm{CN})}, \mathbf{e}_{vu}^{(\mathrm{JA})}, \mathbf{e}_{vu}^{(\mathrm{RA})}$ are trainable heuristic embeddings by encoding CN, JA, RA between nodes v, u, respectively. $\mathrm{COMBINE}(\cdot, \cdot)$ can be Hadamard production, concatenation, etc. CONCAT is the operation of concatenation. $\mathrm{PREDICTOR}(\cdot)$ is a predictor like MLP.

Algorithm 2. Link prediction by integrating statistical heuristics into the GNN

5.2. Empirical Study

5.2.1. Experimental Design

If Remark 3 holds, we expect that properly integrating NCN-dependent heuristics into a GNN could improve the link prediction performance. To this end, we design our experiments as detailed in Algorithm 3. As shown, given two nodes v, u, the link prediction is performed by combining the two nodes' representations from the last GNN layer into a pair-specific link representation $\mathbf{h}_{vu}$, then concatenating it with heuristic encodings, and lastly passing the concatenation into a predictor like MLP. During the training stage, the node pair (v, u) can be a positive or negative link sample, where a negative sample can be two distant nodes that are not connected to each other.

In Algorithm 3, $\mathbf{h}_i^{(0)}$ can be initialized using feature vector $\mathbf{x}_i \in \mathbf{X}$, node embedding $\mathbf{e}_i \in \mathbf{E}$ or the concatenation of both $\mathbf{x}_i$ and $\mathbf{e}_i$. Some may refer to node embedding as an intermediate representation of a node in GNNs. In this work, we clearly distinguish node embeddings from node representations. We consider node embedding as a type of node-wise input feature. The embedding of a node can be viewed as encoding a unique node id into a trainable embedding vector, which is like encoding a unique word id into the word embedding in natural language processing[46]. Note that we can encode any feature into a trainable embedding vector (e.g., encoding node degree to an embedding). The main difference between node embeddings and the embeddings of other node features is that each node embedding vector is unique to that node. For example, an embedding of a node degree is not unique to that node (different nodes could have the same node degree).

We also encode link heuristics into trainable embedding vectors. To verify Remark 3, we only need to encode NCN-

dependent link heuristics. The methodology of encoding heuristics is as follows. For heuristics that are discrete integer values (e.g., CN), we assign a trainable embedding vector to each integer. In the case of heuristics that are continuous floating-point values (e.g., AA), we partition the value range into small bins and subsequently allocate each bin a unique embedding vector. Encoding heuristics into embeddings is mainly because if we directly use heuristics as features, we find that the model optimization is challenging, where the model is more likely to get stuck in a local optimum. This issue could arise due to the high correlation between the heuristic features and the link samples. Encoding heuristics into trainable embeddings can address this issue successfully.

5.2.2. Datasets

Table 2. Statistics of OGB link prediction datasets used in our experiments.

Dataset	#Nodes	#Edges	#Degree
OGBL-DDI	4,267	1,334,889	500
OGBL-COLLAB	235,868	1,285,465	8
OGBL-PPA	576,289	30,326,273	73
OGBL-CITATION2	2,927,963	30,561,187	21

All of our experiments are conducted on four OGB link prediction datasets: *ogbl-collab*, *ogbl-citation2*, *ogbl-ppa*, and *ogbl-ddi*[26]. The statistics of datasets are summarized in Table 2. All these datasets are constructed based on real-world data, covering diverse realistic applications and spanning different scales (4K - 3M nodes). OGB provides an official evaluation protocol. We completely follow it in the data splits and evaluation metrics (i.e., Hits@50, MRR, Hits@100, and Hits@20 on ogbl-collab, ogbl-citation2, ogbl-ppa and ogbl-ddi, respectively). We report the result on the test set, with mean and standard deviation computed across 10 trials.
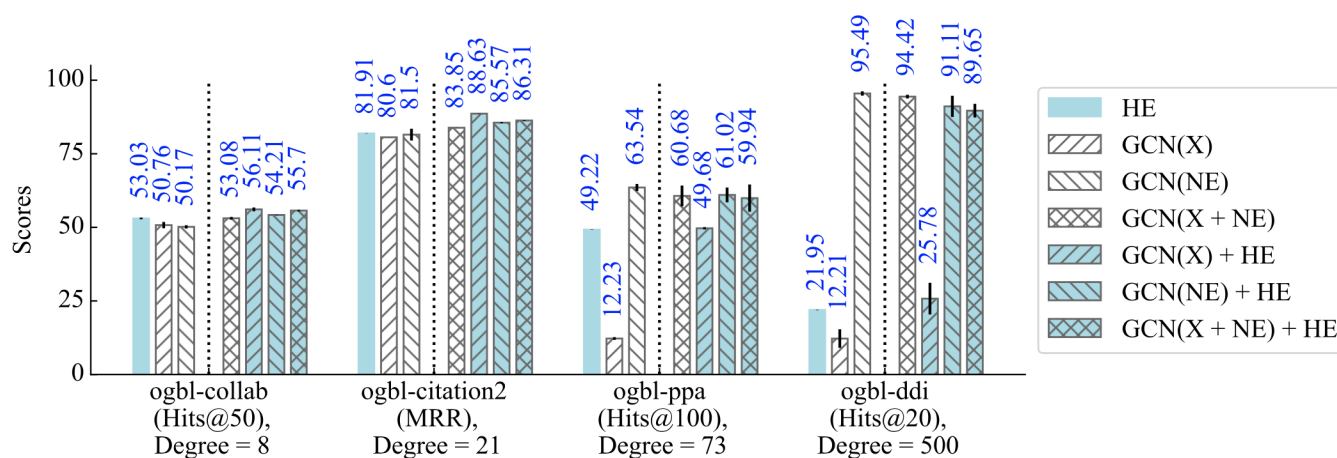

Figure 2. The results of Algorithm 2 on four OGB link prediction datasets, using heuristic encoding (HE) only, node features (X) only, node embeddings (NE) only, or their combinations. The data splits and evaluation metrics follow OGB official evaluation protocol[26].

5.2.3. Implementation Details

We implement our Algorithm 2 based on PyTorch and PyTorch Geometric[47]. All embedding vectors are initialized by following the methods in[48][49]. The model is trained with Adam optimizer[50]. The learning rate is decayed using the ExponentialLR method[51]. We conduct experiments on ogbl-ddi and ogbl-collab on a Linux machine with 192G RAM, and NVIDIA Quadro P6000 (24G), and on ogbl-ppa and ogbl-citation2 on a machine with 512G RAM and NVIDIA A100 (40G). Table 3 lists the configurations of the Algorithm 2 for the best performance. We provide our code for reproducing the results at https://github.com/astroming/GNNHE.

Table 3. Configurations of the Algorithm 2 for the best performances.

	ogbl-ddi	ogbl-collab	ogbl-ppa	ogbl-citation2
GNN module	GCN	GCN	GCN	GCN
GNN layers	2	2	2	2
predictor	MLP	MLP	MLP	MLP
predictor layer	4	5	3	4
heuristics	-	SPD,CN,AA	-	SPD,AA
heuristic embedding dim	-	32	-	32
node embedding dim	512	-	256	-
lr	0.003	0.002	0.001	0.001
dropout rate	0.3	0.3	0.3	0.25
gradient clip norm	5	10	5	10
batch size	100000	70000	100000	15000

5.2.4. Experimental Results

Fig. 2 shows the experimental results, where HE is the model that only uses the heuristic encoding $\mathbf{e}_{vu}$ in Line 11 of Algorithm 2. GNN(X), GNN(NE), and GNN(X+NE) denote the models only using the GNN with three different inputs: node features (X) only, node embeddings (NE) only, and concatenation of both X and NE (X+NE). The model GNN(X)+HE uses both $\mathbf{h}_{vu}$ and $\mathbf{e}_{vu}$.

We can see in Fig. 2 that HE outperforms GNN(X) on all datasets, suggesting that NCN-dependent heuristics convey meaningful information that could not be effectively learned by the GNN. Moreover, most of the results of combining GNN and HE are better than those only using GNN. Especially, GNN(X)+HE achieves the best on ogbl-collab and ogbl-ciation2. All these results can support Remark 3.

6. GNNs with Node Embedding in Link Prediction

As shown in Fig. 2, on two relatively sparse graphs, i.e., ogbl-collab (degree 8) and ogbl-citation2 (degree 21), the performance of GNN(NE) is on par with that of GNN(X), and GNN(X+NE) performs better than GNN(NE) and GNN(X). By

comparison, on two denser graphs, i.e., ogbl-ppa (degree 73) and ogbl-ddi (degree 500), GNN(NE) outperforms GNN(X) by a large margin. These results indicate that incorporating node embeddings into GNNs can enhance the link prediction performance. More importantly, it reveals a strong positive correlation between the graph degree and the performance improvement by node embeddings, i.e., denser graphs exhibit greater improvement.

Remark 4. *For GNN-based link prediction models like Algorithm 2, when the training is supervised by positive and negative links, trainable node embeddings could enhance the expressive power of these models.*

Analysis. In Algorithm 2, the parameters optimized by the link samples could include model weights, trainable node embeddings, and other embedding weights. The key difference between node embedding weights and other learnable weights is that the former is unique to each node but the latter is shared across multiple nodes (e.g., the GNN weight matrix $\mathbf{W}^{(l)}$ in Eq. 1 is shared across all nodes). The unique nature of node embeddings can bring benefits. As shown in Algorithm 2, when the model training is supervised by a link sample (v, u), for a GNN using node embeddings, the loss calculated based on $(y_{v,u}, \hat{y}_{v,u})$ would be used to optimize the node embeddings of nodes v, u and their neighboring nodes (i.e., the nodes involved in calculating $\mathbf{h}_v^{(L)}, \mathbf{h}_u^{(L)}$). The link state of (v, u) could be encoded into the node embeddings of these nodes, which would enable those node embeddings to remember the relationships between corresponding nodes. After being trained with sufficient positive and negative link samples, the node embedding of each node could know which nodes (through their node embeddings) in the graph are more likely to be or not to be connected to that node.

If node embeddings are not used in Algorithm 2, link samples will only supervise the optimization of the weights that are shared across multiple nodes. The states of link samples could not be effectively preserved by the model since these shared weights might learn a common pattern for different nodes rather than unique to a node. By comparison, each node embedding is unique to that node and could learn the link information specific to that node. In this respect, trainable node embeddings could enhance the expressive power of the GNN-based link prediction model.

In Remark 4, the requirement of the model training is supervised by positive and negative links is indispensable. Without this prerequisite, the link state between two nodes could not be encoded into node embeddings. Additionally, negative link samples can allow the embeddings of two distant nodes and their neighbors to see each other during the **optimization of the GNN-based model.**

Finding 5. *Following Remark 4, the denser the graph, the more the enhancement by node embeddings.*

Analysis. In GNN-based link prediction models like Algorithm 2, node embeddings in a dense graph could be better learned for link prediction than those in a sparse graph. Our explanation is as follows. In a dense graph, a node often has a lot of neighboring nodes, thereby providing numerous opportunities for that node to meet other nodes and encode link relationships of that node with these other nodes into its embedding during the GNN training. In contrast, a sparse graph typically contains only a limited number of neighbors for each node. For example, in the case where a node v has only one neighboring node w, the optimization of the embedding of node v in a GNN would mainly rely on its neighbor w. As a result, the learned embedding of node v would lack sufficient information to identify the relationships between node v and

the majority of the other nodes in the sparse graph because node v rarely or never sees them during the training process.

Finding 5 indicates that the graph degree significantly influences the effectiveness of trainable node embeddings in GNN-based link prediction models. Interestingly, prior studies[52] have also highlighted the sensitivity of heuristic methods to the graph degree. This underscores the necessity of considering the graph degree when selecting link prediction methods, as their efficacy may vary depending on it. Investigating the influence of different graph degrees on link prediction methods represents a compelling direction for further research.

7. GNNs in Learning non-NCN-dependent Structural Information

In Section 5, we present that GNNs cannot completely learn the structural information related to the number of common neighbors between two target nodes. When it comes to the question of what non-NCN-dependent pair-specific structural information can be learned via GNNs, it poses a significant challenge. This is due to the potential presence of diverse types of non-NCN-dependent information. Unlike NCN-dependent information directly related to the number of common neighbors, non-NCN-dependent information tends to be abstract and difficult to express in rigorous mathematical terms. For example, our review in Section 3.2 identifies only three non-NCN-dependent link heuristics. In this work, we leave the exploration of this question as a future research endeavor. Nevertheless, by comparing Algorithm 1 and Algorithm 2, we have the following insight.

Remark 6. *The learning styles of SimRank in Algorithm 1, and the GNN-based link prediction model with node embeddings in Algorithm 2, exhibit certain similarities.*[1]

Analysis. Comparing Algorithms 1 and 2, several similarities emerge. Firstly, similarity scores in Line 3 of Algorithm 1 and node embeddings in Line 3 of Algorithm 2 both need to be initialized and can be dynamically trained. Secondly, the updating computations of both algorithms (i.e., Line 5 of Algorithm 1 and Line 6 of Algorithm 2) involve neighboring nodes. Moreover, both the learned results (i.e., s_{ij} in Algorithm 1 and $\hat{y}_{v,u}$ in Algorithm 2) describe the existence likelihood of a link between two nodes. However, compared to Algorithm 2 where the trainable parameters include node embeddings, model weights, etc., the expressive power of SimRank is limited. In SimRank, only the similarity scores between every two nodes can be optimized, with each score always taking the form of a scalar.

Remark 6 implies that although NCN-dependent structural information cannot be effectively learned via GNNs (Remark 3), other types of information (e.g., the information captured by SimRank) might be learned through GNNs.

8. Limitation Analysis of Existing Methods

In this section, we first present a brief survey of existing link prediction methods and then identify their possible limitations.

8.1. A Survey of Link Prediction Methods

8.1.1. Heuristic Methods

As outlined in Section 3, traditional link heuristics are usually defined based on the number of common neighbors or paths between two nodes[53]. Their effectiveness in link prediction has been confirmed in real-world tasks[53][54]. However, many link heuristics are designed for specific graph applications and their performance may vary on different graphs[54]. Also, the expressiveness of these methods is limited compared to graph representation learning[9].

8.1.2. Graph Neural Networks

GNNs have proven their effectiveness in various graph applications[55][56][57][2]. A number of GNN models have been proposed[24][58][59]. GCN[24] learns node representations by summing the normalized representations from the first-order neighbors. GraphSAGE[43] samples and aggregates representations from local neighborhoods. GAT[25] introduces an attention-based GNN architecture. JKNet[58] adds a pooling layer following the last GNN layer and each GNN layer has a residual connection to this layer. Cluster-GCN[60] proposes an efficient algorithm for training deep GCN on large graphs. LRGA[61] incorporates a Low-Rank global attention module to GNNs. Several works such as Mixhop[62] and DEGNN[17] propose techniques to leverage high-hop neighbors. ID-GNN[63] embeds each node by considering its identity.These GNNs have demonstrated promising link prediction performance.

8.1.3. Non-GNN-based Node Embedding Methods

A family of node embedding methods is those built with matrix factorization[64]. MF[65] is a pioneer work employing matrix factorization in link prediction. FSSDNMF[66] proposes a link prediction model based on non-negative matrix factorization. In general, such methods mainly rely on the adjacency matrix and tend to encounter scalability issues when employed on large graphs.

Another family of node embedding methods is those based on relative distance encoding. The similarity of nodes in the embedding space reflects the semantic similarity of nodes in the graph[67]. Such methods would learn more similar embeddings for two close nodes than two distant nodes. Following word embedding[46], methods such as Deepwalk[67], Node2vec[68], and NodePiece[69] learn node embeddings by treating the nodes as words and treating the sequences of nodes generated based on links as sentences. UniNet[70] improves the efficiency of such methods using the Metropolis Hastings sampling technique[71]. Inspired by subword tokenization[72], NodePiece[69] explores parameter-efficient node embeddings.

8.1.4. SEAL-type Methods

SEAL-type methods have shown superior performance among existing link prediction approaches[26][15][15]. SEAL and its subsequent works[9][17][18][19][20] address the link prediction problem by classifying the subgraphs that are extracted specifically for candidate links. SEAL[9] extracts a local enclosing subgraph for each candidate link and uses a GNN[45] to classify these subgraphs for link prediction. GraiL[18] is developed for inductive link prediction. It is similar to SEAL but it replaces SortPooling[45] with MEAN-pooling. DEGNN[17] proposes a distance encoding GNN. Cai et al.[73] transform the

enclosing subgraph into a corresponding line graph and address the link prediction task with the node classification problem in its line graph. Pan et al.[74] follow the subgraph strategy in SEAL while designing a new pooling mechanism called WalkPool. SUREL[19] proposes an algorithmic technique to improve the computational efficiency of subgraph generation in SEAL. SIEG[75] incorporates the structural information learned from the enclosing subgraphs into the GNN for link prediction, which fuses topological structures and node features to take full advantage of graph information for link prediction.

8.1.5. Methods Specific for Link Prediction

Various link prediction-specific methods have been introduced[76][2]. Wang et al.[44] present PLNLP by jointly using the representations learned by a GNN, distance encoding, etc. Neo-GNN[77] weighted aggregates the link prediction scores obtained by heuristics and a GNN. NBFNet[27] generalizes traditional path-based link heuristics into a path formulation. Singh et al.[10] show that adding a set of edges to the graph as a pre-processing step can improve the performance of link prediction models. PermGNN[78] optimizes the neighborhood aggregator directly by link samples. Zhao et al[11] study counterfactual questions about link existence by causal inference. RelpNet[79] aggregates edge features along the structural interactions between two target nodes. Guo et al.[12] propose cross-model distillation techniques for link prediction. Shang et al.[80] propose a negative link sampling method PbTRM based on a policy-based training method. Li et al.[81] study the integration of large language models (LLMs) and prompt learning techniques with graphs, enhancing graph transfer capabilities across diverse tasks and domains.

8.2. Limitation Analysis

We provide two basic insights into the application of GNNs in link prediction. Firstly, we show that aggregation-based GNNs inherently lack the ability to learn NCN-dependent structural information for link prediction (Remark 3). Secondly, we demonstrate that node embeddings can boost the performance of GNN-based link prediction models on dense graphs (Remark 4 and Finding 5). These can serve as effective avenues to identify and interpret the limitations of existing link prediction methods. To illustrate this, we present two case studies.
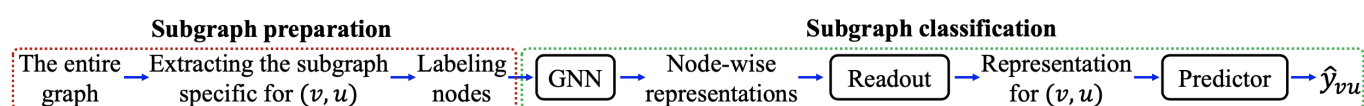
Figure 3. The algorithm flow of SEAL-type link prediction methods.

Case study 1. Can SEAL effectively learn NCN-dependent structural information?

SEAL-type methods have achieved the best performance on several link prediction datasets[9][17][18][19]. SEAL[9] has proven that most link heuristics between two nodes can be computed approximately within an enclosing subgraph extracted specifically for that two nodes. As shown in Fig. 3, most SEAL-type methods employ GNNs for graph representation learning, with the expectation that from such enclosing subgraphs, the GNN can learn the structural

information equivalent to link heuristics including CN, AA, Katz, etc. However, whether this expectation holds true has not been thoroughly investigated in existing works. Herein we present a rough analysis to examine this issue.

First, the GNNs used in SEAL-type methods, e.g., DGCNN[45] in SEAL[9], R-GCN[82] in GraiL[18], still belong to the type of aggregation-based GNNs. According to Remark 3, these GNNs inherently cannot effectively learn NCN-dependent structural information.

Furthermore, we have noticed that SEAL-type methods typically use a labeling technique[23] to add labeling features to each node in the enclosing subgraph. The labeling features of each node describe the relationship of that node to the target two nodes. Fig. 4 illustrates such a labeling method, where the labeling features of a node are the shortest path distances from the node to the target pair of nodes. The work[23] points out that the labeling features can help the GNN learn the structural information about the number of common neighbors. Their explanation is as follows. As shown in Fig. 4, for node v and u, in the first iteration of the neighborhood aggregation of a GNN, only the common neighbors between node v and u will receive the labeling messages from both v and u; then in the second iteration, the common neighbors will pass such messages back to both v and u, which can encode the number of common neighbors into the representations of node v, u.

However, the aforementioned explanation raises questions regarding its validity. In the second iteration, apart from the common neighbors, the non-common neighbors of node v also pass their messages back to v. The messages from all neighbors of v are then aggregated through a set-based pooling (e.g., MEAN or attention-based pooling as shown in Eq. 2). Such an aggregated result for node v would wash out the distinguishable labels' information. We present an example to illustrate this. As shown in Fig. 4, if the pooling method in a GNN is MEAN, then the aggregation of the labeling features of the neighbors of node v would be equal to that of node w, i.e., MEAN($\{10, 3, 2, 1\}$) = MEAN($\{4, 6, 4, 3, 3\}$). This means that the distinct labeling features of the neighbors of a node are not effectively kept in the aggregated result. In other words, the aggregated results for node v and w in the positive and negative link samples become indistinguishable. Note that attention-based pooling in GNNs like GAT[25] also suffers from the above limitation for the same reason as the analysis of Remark 2. The same goes for node u. It should be noted that our example is merely for illustrative purposes. In practice, a GNN layer contains a series of complicated operations such as linear and non-linear transformations, dropout, residual connection, and others. The structural information in the labeling features could be partially kept in the learned representations.
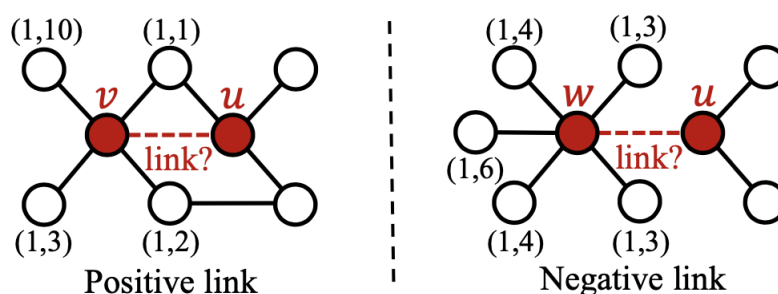

Figure 4. Node labeling in SEAL-type methods. The left is a subgraph specific for a

positive link sample and the right is a negative one. The labeling features are based on the SPDs from every node (here only show the first-order neighbors of node *v* or *w*) to the target pair of nodes. For example, on the left, the node with the labeling (1.0) indicates that the SPD from this node to node *v* and *u* is 1 and 10, respectively.

Table 4. Results on OGB link prediction datasets. Higher scores indicate better performance, with the best results highlighted in bold. Herein, "HE" stands for Heuristic Encoding, "X" represents node attributes, and "NE" denotes Node Embedding.

|	ogbl-ddi	ogbl-collab	ogbl-ppa	ogbl-citation2
	Hits@20 (%)	Hits@50 (%)	Hits@100 (%)	MRR (%)
MF[65]	13.68 ± 4.75	38.86 ± 0.49	32.29 ± 0.94	51.86 ± 8.43
FSSDNMF[66]	14.62 ± 2.64	37.95 ± 3.25	34.15 ± 1.16	54.71 ± 8.73
DeepWalk[67]	26.42 ± 6.10	50.37 ± 0.34	28.88 ± 1.63	60.11 ± 0.23
Node2vec[68]	23.26 ± 2.09	48.88 ± 0.54	22.26 ± 0.83	61.41 ± 0.11
NodePiece[69]	24.15 ± 3.04	47.88 ± 0.41	22.85 ± 0.94	61.52 ± 2.91
GraphSAGE[43]	83.90 ± 4.74	48.10 ± 0.81	16.55 ± 2.40	82.60 ± 0.36
GAT[25]	95.38 ± 0.94	52.26 ± 0.85	51.33 ± 2.16	83.17 ± 0.54
Neo-GNN[77]	75.72 ± 3.42	55.31 ± 0.53	49.13 ± 0.60	87.26 ± 1.84
PLNLP[44]	90.88 ± 3.13	52.92 ± 0.98	32.38 ± 2.58	84.92 ± 0.29
NBFNet[27]	18.14 ± 2.12	51.15 ± 1.38	23.96 ± 2.03	74.91 ± 2.37
SEAL[9]	30.56 ± 3.86	54.71 ± 0.79	48.80 ± 4.56	87.67 ± 0.32
DEGNN[17]	26.63 ± 6.42	53.74 ± 0.45	36.48 ± 5.38	60.30 ± 0.81
SIEG[75]	31.95 ± 3.93	55.35 ± 0.52	53.35 ± 1.39	89.87 ± 0.10
HE	21.95 ± 0.08	53.03 ± 0.29	49.22 ± 0.06	81.91 ± 0.05
GCN(X)	12.21 ± 3.16	50.76 ± 1.08	12.23 ± 0.47	80.60 ± 0.04
GCN(NE)	95.49 ± 0.73	50.17 ± 0.56	63.54 ± 1.21	81.50 ± 2.01
GCN(X+NE)	94.42 ± 0.63	53.08 ± 0.46	60.68 ± 3.52	83.85 ± 0.03
GCN(X)+HE	25.78 ± 5.38	56.11 ± 0.64	49.68 ± 0.39	88.63 ± 0.05
GCN(NE)+HE	91.11 ± 3.60	54.21 ± 0.07	61.02 ± 2.51	85.57 ± 0.19
GCN(X+NE)+HE	89.65 ± 2.32	55.70 ± 0.24	59.94 ± 4.62	86.31 ± 0.12
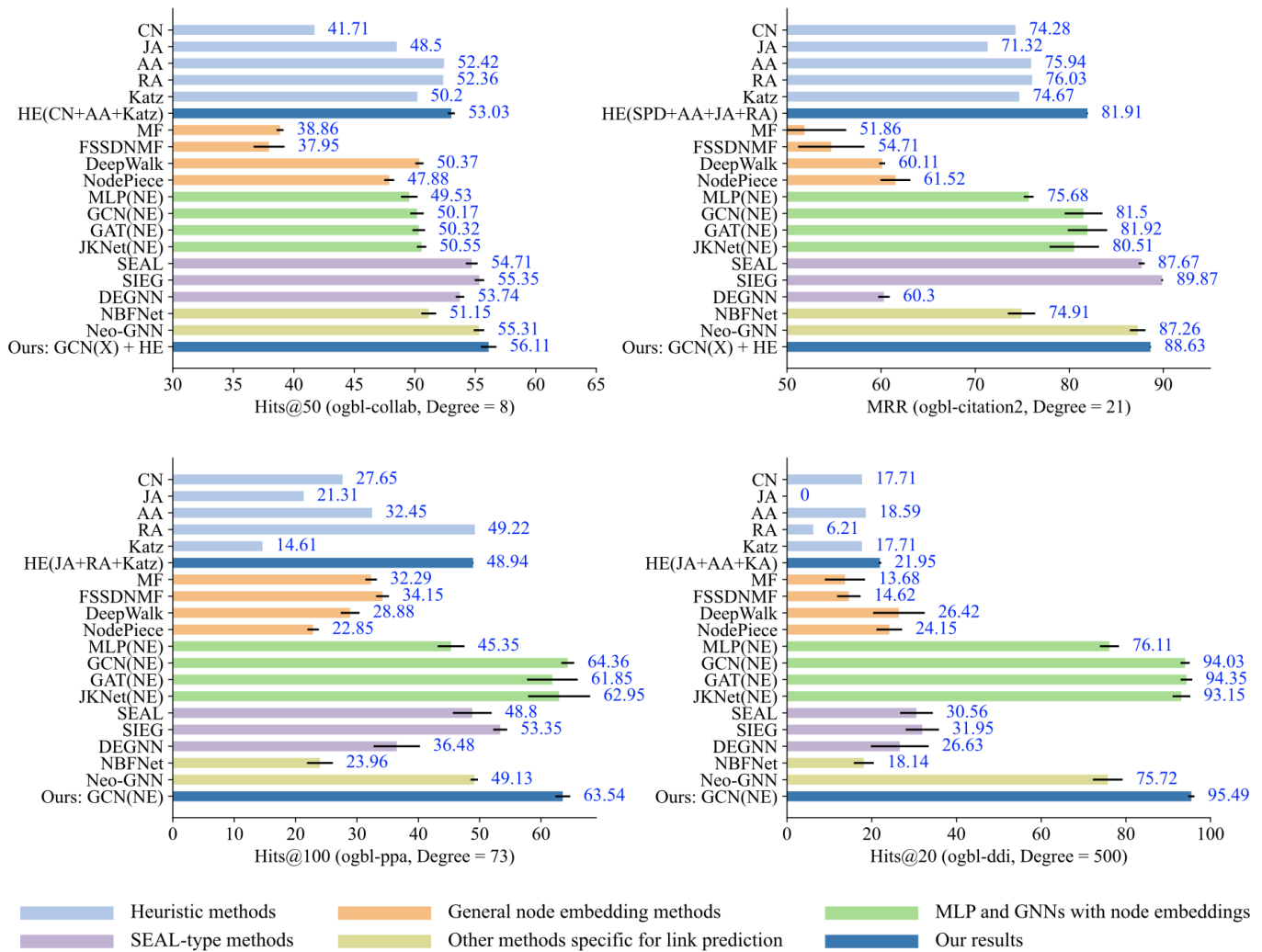
Figure 5. Results of different methods for link prediction on four OGB datasets. For MLP and general GNNs, we present their results obtained by utilizing node embeddings, considering the dominant performance of node embeddings as shown in Fig. 2.
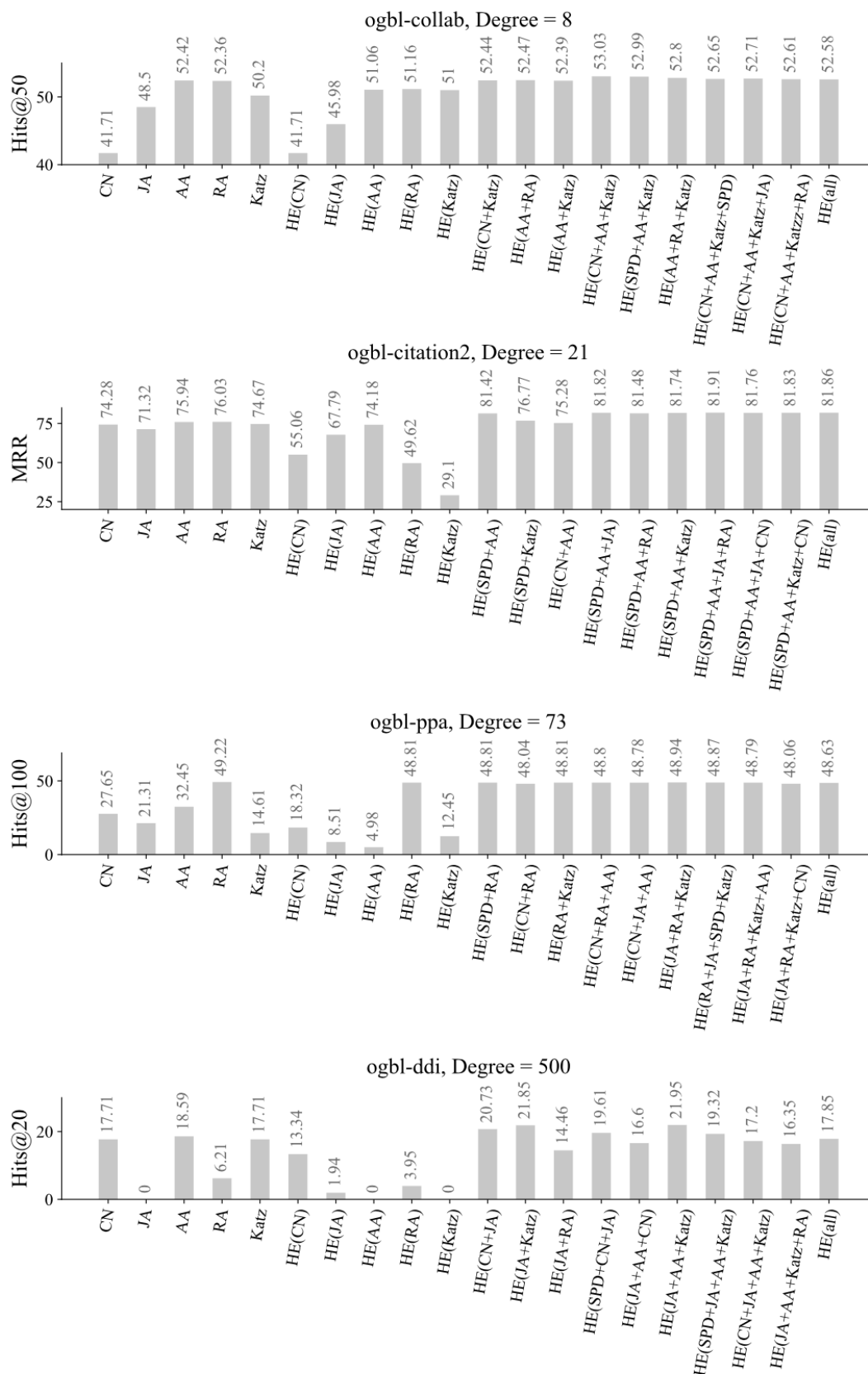
Figure 6. Results of HE (Heuristic Encoding) through Algorithm 2, where we encode various combinations of NCN-dependent heuristics. We can find that the best HE is often achieved by encoding a select number of heuristics rather than all heuristics.

Although SEAL-type methods cannot effectively learn structural information related to the number of common neighbors, we highlight that these methods are powerful for link prediction. Such methods transform the pair-specific link prediction problem into a graph-level classification task. Compared to models like Algorithm 2 that only combine the representations of two target nodes, SEAL-type methods take advantage of the representations of not only two target nodes but also their neighboring nodes in the enclosing subgraph, enabling the model to consider more information of the surrounding environment of the candidate link.

Case study 2. NBFNet lacks the algorithmic ability to leverage node embeddings.

NBFNet[27] is a model specifically developed for link prediction. Differently from GNN-based link prediction methods like Algorithm 2 and SEAL-type methods, NBFNet generalizes traditional link heuristics such as Katz index[21], Personalized PageRank[83] into a general formulation and approximates such formulation using a special network. Unlike aggregation-based GNNs that propagate and aggregate node-wise representations, NBFNet is designed to train edge-wise representations. The model architecture of NBFNet makes it hardly consider node-wise information. This would make NBFNet lack the algorithmic ability to train powerful node embeddings and may lead to non-competitive link prediction performance on dense graphs, considering the strong performance of the GNN only using node embeddings on dense graphs as shown in Fig. 2.

8.3. Further analysis of experimental results

We expand our limitation analysis of existing link prediction methods by examining their experimental performance on four OGB benchmark datasets. The results are presented in Table 4 and Fig. 5. In the interest of brevity, our analysis focuses on several main types of methods.

First, as shown in Fig. 5, the performance of heuristic methods is not stable across the four datasets. For example, RA performs best on ogbl-ppa but second worst on ogbl-ddi. These results are consistent with the research of[54] which indicates that many link heuristics are designed for specific applications and may perform well only on those specific graphs. Moreover, the unstable performance of every single heuristic confirms the need of combining multiple heuristics in link prediction, as shown in our Algorithm 2.

We also report the heuristic encoding results (Fig. 6) obtained through Algorithm 2 when only employing heuristic encoding. We find that it is not always true that the more the heuristics used, the better the performance of HE. In contrast, encoding a certain number of heuristics can yield the best performance, whereas encoding too many heuristics would pose an optimization challenge.

The node embedding methods based on relative distance encoding (DeepWalk[67], NodePiece[69]) perform slightly better than those based on matrix factorization (MF[65], FSSDNMF[66]). Nevertheless, all these methods fall short compared to other methods. This could be attributed to the limitations of such methods, e.g., reliance solely on the adjacency matrix or unsupervised learning without link samples. This also underscores the critical role of link samples in supervising the training of node embeddings for link prediction.

Fig. 5 also presents the results of MLP and general GNNs (GCN[24], GAT[25] and JKNet[58]) that use node embeddings only. MLP(NE) performs much worse than GNNs, demonstrating the significance of neighborhood aggregation of GNNs in training node embeddings, considering that MLP updates each node's representation independently of other nodes. Furthermore, GCN(NE) and GAT(NE) perform comparably, indicating that the expressiveness of GCN is sufficient for learning node embedding. The similar performance of GCN and GAT empirically supports our analyses in Remark 2 and 3, where we point out that the attention mechanism (e.g., GAT) cannot address the inherent issue of GNNs in learning structural information related to the number of each node's neighbors and of common neighbors between two nodes.

In Fig. 5, SEAL-type methods show state-of-the-art performance. Especially, SIEG[75] achieves the best results on two sparse graphs, i.e., ogbl-collab and ogbl-citation2 with graph degrees of 8 and 21, respectively. However, they perform worse than general GNNs with node embeddings (GCN(NE)[24], GAT(NE)[25] and JKNet(NE)[58]) on two dense graphs, i.e., ogbl-ppa and ogbl-ddi. This discrepancy in the performance of SEAL-type methods could be attributed to the algorithmic challenge of training node embeddings using subgraphs. Unlike general GNNs, the algorithm of SEAL-type methods limits each node to perceive other nodes within the subgraph rather than the entire graph, thereby restricting the information flow between nodes and potentially reducing the efficiency of learning node embeddings.

Besides, Fig. 5 shows two link prediction-specific methods, namely NBFNet[27] and Neo-GNN[77]. NBFNet underperforms on four datasets, which aligns with the limitations identified in Section 8.2. Neo-GNN predicts link likelihood by combining the scores obtained by heuristic methods and the result produced by a GNN. It performs on par with the state-of-the-art SEAL-type methods on two sparse graphs (ogbl-collab and ogbl-citation2).

Lastly, our GNN(X)+HE based on Algorithm 2 performs better than SEAL-type methods on ogbl-collab, supporting our limitation analysis of SEAL-type methods in Section 8.2, i.e., such methods could not effectively learn the information equivalent to NCN-dependent heuristics.

It should be noted that our focus does not lie in developing solutions to these limitations of the existing methods, as this goes beyond the scope of our main goal. Nevertheless, these identified issues could pave the way for future research.

9. Implication for Practical Applications

This study carries significant implications for real-world link prediction applications. A particular emphasis is the selection of appropriate solutions tailored to the graph degree.

For link prediction on sparse graphs, the performance of various methods in our experiments highlights the important role of NCN-dependent information. Approaches that can leverage such information, such as SEAL-type methods and Neo-GNN, generally outperform those that cannot. In practical scenarios, both SEAL-type methods and GNNs with heuristic encoding could yield satisfactory performance. Additionally, traditional machine learning models like MLP incorporating multiple heuristic encodings serve as viable alternatives, which could achieve comparable performance to the top-

performing methods, while offering faster processing times. This is particularly advantageous for tasks such as recommender systems that demand rapid model response.

For link prediction on dense graphs, the contribution of node embeddings becomes dominant. Simple GNNs like GCN[24] with the incorporation of trainable node embeddings can outperform most existing methods, rendering such a solution an optimal choice. However, this does not mean that the model using node embeddings will certainly perform better than that only using node features, especially in practical applications where careful feature engineering guided by domain knowledge is conducted. Besides, the use of trainable node embeddings remains limitations in the inductive setting[18], where new nodes are added to the graph, and the model together with all node embeddings may need to be retrained. In such cases, the methods that do not involve the training of node embeddings may offer more practical suitability.

10. Limitations

This paper primarily explores several fundamental issues in link prediction methods, particularly in GNNs. It does not seek to introduce novel model architectures. Some analyses in this paper are provided in the form of examples and may lack rigorous mathematical proofs.

11. Conclusion

Link prediction stands as a pivotal task within the realm of graph applications. Our exploration into this domain reveals noteworthy variations in the performance of various link prediction methods across different graphs, with a significant dependence on graph degrees. Notably, on dense graphs, we observe that straightforward GNNs, like GCN, exhibit superior link prediction performance compared to many models that are developed specifically for link prediction. In contrast, on sparse graphs, the simple common-neighbor method often outshines GNN-based approaches. Understanding and interpreting these performance fluctuations is imperative, serving as a compass for refining existing methodologies and establishing a foundation for the development of more effective link prediction algorithms.

In addition, this work brings suggestions to practitioners in link prediction. Specifically, on sparse graphs, either SEAL-type methods or GNNs plus heuristic encoding can yield satisfactory performance. On dense graphs, GNN with node embeddings is an ideal choice in the transductive setting. For inductive learning, methods that do not involve the training of node embeddings may be more suitable.

Footnotes

[1] For Remark 6, we do not compare the performance of Algorithm 1 and Algorithm 2 due to the difficulty of SimRank in computation. For example, the basic memory requirement of SimRank is 415G and 2.42T on ogbl-collab and ogbl-ppa, respectively. Besides, SimRank produces 0 at Hits@20 on ogbl-ddi.

References

1. ^Zhou J, Liu L, Wei W, Fan J (2022). "Network representation learning: from preprocessing, feature extraction to node embedding". *ACM Computing Surveys (CSUR)*. **55** (2): 1–35.

2. a, b, c, d, e, f, g, h Khemani B, Patil S, Kotecha K, Tanwar S (2024). "A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions". *Journal of Big Data*. **11** (1): 18.

3. a, b, c, d, e Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? In: *International Conference on Learning Representations*; 2018.

4. ^Chen Z, Chen L, Villar S, Bruna J (2020). "Can graph neural networks count substructures?" *Advances in neural information processing systems*. **33**: 10383–10395.

5. ^Geerts F, Reutter JL (2021). "Expressiveness and Approximation Properties of Graph Neural Networks". In: *International Conference on Learning Representations*.

6. ^Bouritsas G, Frasca F, Zafeiriou S, Bronstein MM (2022). "Improving graph neural network expressivity via subgraph isomorphism counting". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **45**(1): 657–668.

7. ^Liu Z, Yang D, Wang Y, Lu M, Li R (2023). "EGNN: Graph structure learning based on evolutionary computation helps more in graph neural networks". *Applied Soft Computing*. **135**: 110040.

8. ^Zhou Z, Zhou S, Mao B, Zhou X, Chen J, Tan Q, Zha D, Feng Y, Chen C, Wang C. "Opengsl: A comprehensive benchmark for graph structure learning". *Advances in Neural Information Processing Systems* **36**, 2024.

9. a, b, c, d, e, f, g, h, i, j, k, l Zhang M, Chen Y (2018). "Link prediction based on graph neural networks". *Advances in Neural Information Processing Systems*. **31**: 5165–5175.

10. a, b Singh A, Huang Q, Huang SL, Bhalerao O, He H, Lim SN, Benson AR (2021). "Edge proposal sets for link prediction". *arXiv preprint arXiv:2106.15810*. Available from: https://arxiv.org/abs/2106.15810.

11. a, b Zhao T, Liu G, Wang D, Yu W, Jiang M. "Learning from counterfactual links for link prediction." In: *International Conference on Machine Learning*. PMLR; 2022. p. 26911-26926.

12. a, b Guo Z, Shiao W, Zhang S, Liu Y, Chawla N, Shah N, Zhao T (2022). "Linkless Link Prediction via Relational Distillation". *arXiv preprint arXiv:2210.05801*.

13. ^Wang H, Cui Z, Liu R, Fang L, Sha Y (2023). "A multi-type transferable method for missing link prediction in heterogeneous social networks". *IEEE Transactions on Knowledge and Data Engineering*. **35** (11): 10981–10991.

14. ^Liu X, Li X, Fiumara G, De Meo P (2023). "Link prediction approach combined graph neural network with capsule network". *Expert Systems with Applications*. **212**: 118737.

15. a, b, c, d, e, f, g Li J, Shomer H, Mao H, Zeng S, Ma Y, Shah N, Tang J, Yin D (2024). "Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking". *Advances in Neural Information Processing Systems* **36**.

16. ^Huang X, Romero M, Ceylan I, Barcel\'o P (2024). "A theory of link prediction via relational weisfeiler-leman on knowledge graphs". *Advances in Neural Information Processing Systems* **36**.

17. [a, b, c, d, e, f]Li P, Wang Y, Wang H, Leskovec J (2020). "Distance encoding: Design provably more powerful neural networks for graph representation learning". Advances in Neural Information Processing Systems **33**: 4465–4478.

18. [a, b, c, d, e, f, g]Teru K, Denis E, Hamilton W. "Inductive relation prediction by subgraph reasoning." In:International Conference on Machine Learning. PMLR; 2020. p. 9448-9457.

19. [a, b, c, d, e]Yin H, Zhang M, Wang Y, Wang J, Li P (2022). "Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning". arXiv preprint arXiv:2202.13538.

20. [a, b, c]Tan Q, Zhang X, Liu N, Zha D, Li L, Chen R, Choi SH, Hu X. Bring your own view: Graph neural networks for link prediction with personalized subgraph selection. Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining. 2023:625-633.

21. [a, b, c, d]Katz L. "A new status index derived from sociometric analysis".Psychometrika. **18**(1):39–43, 1953.

22. [a, b, c]Corso G, Cavalleri L, Beaini D, Li{\`o} P, Veli{\v{c}}kovi{\'c} P (2020). "Principal neighbourhood aggregation for graph nets". Advances in Neural Information Processing Systems **33**: 13260--13271.

23. [a, b, c, d, e]Zhang M, Li P, Xia Y, Wang K, Jin L (2021). "Labeling trick: A theory of using graph neural networks for multi-node representation learning". Advances in Neural Information Processing Systems **34**: 9061--9073.

24. [a, b, c, d, e, f, g]Kipf TN, Welling M. "Semi-Supervised Classification with Graph Convolutional Networks". In:International Conference on Learning Representations (ICLR); 2017.

25. [a, b, c, d, e, f, g, h, i, j]Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y (2018). "Graph Attention Networks". In: International Conference on Learning Representations.

26. [a, b, c, d]Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, Catasta M, Leskovec J (2020). "Open graph benchmark: Datasets for machine learning on graphs". Advances in neural information processing systems **33**: 22118–22133.

27. [a, b, c, d, e]Zhu Z, Zhang Z, Xhonneux L-P, Tang J (2021). "Neural bellman-ford networks: A general graph neural network framework for link prediction". Advances in Neural Information Processing Systems **34**.

28. [^]Liben-Nowell D, Kleinberg J (2003). "The link prediction problem for social networks". In:Proceedings of the twelfth international conference on Information and knowledge management. pp. 556–559.

29. [^]Srilatha P, Manjula R (2016). "Similarity index based link prediction algorithms in social networks: A survey"Journal of Telecommunications and Information Technology. (2): 87–94.

30. [^]Kumar A, Singh SS, Singh K, Biswas B (2020). "Link prediction techniques, applications, and performance: A survey". Physica A: Statistical Mechanics and its Applications **553**: 124289.

31. [a, b]Jaccard P (1901). "\u00c9tude comparative de la distribution florale dans une portion des Alpes et des Jura".Bull Soc Vaudoise Sci Nat. **37**: 547--579.

32. [a, b]Adamic LA, Adar E (2003). "Friends and neighbors on the web".Social networks. **25** (3): 211–230.

33. [a, b]Zhou T, L\u00fc L, Zhang YC (2009). "Predicting missing links via local information".The European Physical Journal B. **71** (4): 623--630.

34. [a, b]Sørensen TJ. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. I kommission hos E. Munksgaard; 1948.

35. [a, b]*Salton G. "Introduction to modern information retrieval".McGraw-Hill. 1983.*

36. [a, b, c, d]*Ravasz E, Somera AL, Mongru DA, Oltvai ZN, Barabási AL (2002). "Hierarchical organization of modularity in metabolic networks". Science. 297(5586): 1551–1555.*

37. [^]*Cannistraci CV, Alanis-Lobato G, Ravasi T (2013). "From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks". Scientific reports. 3 (1): 1613.*

38. [^]*Wu Z, Lin Y, Wang J, Gregory S (2016). "Link prediction with node clustering coefficient"Physica A: Statistical Mechanics and its Applications. 452: 1–8.*

39. [^]*Wu Z, Lin Y, Wan H, Jamil W (2016). "Predicting top-L missing links with node and link clustering information in large-scale networks". Journal of Statistical Mechanics: Theory and Experiment 2016 (8): 083202.*

40. [^]*Leicht EA, Holme P, Newman MEJ (2006). "Vertex similarity in networks".Physical Review E. 73 (2): 026120.*

41. [^]*Barabási AL, Jeong H, Néda Z, Ravasz E, Schubert A, Vicsek T (2002). "Evolution of the social network of scientific collaborations". Physica A: Statistical mechanics and its applications 311 (3-4): 590–614.*

42. [a, b]*Jeh G, Widom J. "Simrank: a measure of structural-context similarity." In:Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 2002. p. 538-543.*

43. [a, b, c, d, e]*Hamilton W, Ying Z, Leskovec J (2017). "Inductive representation learning on large graphs". In:Advances in neural information processing systems. pp. 1024–1034.*

44. [a, b, c, d]*Wang Z, Zhou Y, Hong L, Zou Y, Su H (2021). "Pairwise Learning for Neural Link Prediction"arXiv preprint arXiv:2112.02936.*

45. [a, b, c, d]*Zhang M, Cui Z, Neumann M, Chen Y (2018). "An end-to-end deep learning architecture for graph classification". Proceedings of the AAAI conference on artificial intelligence 32 (1).*

46. [a, b]*Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013). "Distributed representations of words and phrases and their compositionality". Advances in neural information processing systems 26.*

47. [^]*Fey M, Lenssen JE (2019). "Fast Graph Representation Learning with PyTorch Geometric". In:ICLR Workshop on Representation Learning on Graphs and Manifolds.*

48. [^]*Glorot X, Bengio Y (2010). "Understanding the difficulty of training deep feedforward neural networks". In: Proceedings of the thirteenth international conference on artificial intelligence and statistics JMLR Workshop and Conference Proceedings. pp. 249--256.*

49. [^]*He K, Zhang X, Ren S, Sun J (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". Proceedings of the IEEE international conference on computer vision pages 1026–1034.*

50. [^]*Kingma DP, Ba J (2014). "Adam: A method for stochastic optimization".arXiv preprint arXiv:1412.6980.*

51. [^]*Li Z, Arora S. An exponential learning rate schedule for deep learning. In:International Conference on Learning Representations; 2020.*

52. [^]*Shang KK, Li TC, Small M, Burton D, Wang Y (2019). "Link prediction for tree-like networks".Chaos: An Interdisciplinary Journal of Nonlinear Science. 29 (6).*

53. [a, b]*Martínez V, Berzal F, Cubero JC (2016). "A survey of link prediction in complex networks"ACM computing surveys*

(CSUR). **49** (4): 1–33.

54. a, b, c Kov{\'a}cs IA, 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.

55. ^Liang S, Li Z, Liang B, Ding Y, Wang Y, Chen F. Failure prediction for large-scale water pipe networks using GNN and temporal failure series. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. 2021. p. 3955–3964.

56. ^Huang K, Xiao C, Glass LM, Zitnik M, Sun J (2020). "SkipGNN: predicting molecular interactions with skip-graph networks". Scientific reports. **10** (1): 1–16.

57. ^Ying C, Cai T, Luo S, Zheng S, Ke G, He D, Shen Y, Liu T (2021). "Do Transformers Really Perform Badly for Graph Representation?" Advances in Neural Information Processing Systems **34**.

58. a, b, c, d Xu K, Li C, Tian Y, Sonobe T, Kawarabayashi K, Jegelka S. "Representation learning on graphs with jumping knowledge networks." In: International conference on machine learning. PMLR; 2018. p. 5453-5462.

59. ^Zou C, Han A, Lin L, Li M, Gao J (2023). "A simple yet effective framelet-based graph neural network for directed graphs". IEEE Transactions on Artificial Intelligence. 2023.

60. ^Chiang WL, Liu X, Si S, Li Y, Bengio S, Hsieh CJ (2019). "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks". In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019. p. 257–266.

61. ^Puny O, Ben-Hamu H, Lipman Y (2020). "Global attention improves graph networks generalization". arXiv preprint arXiv:2006.07846. Available from: https://arxiv.org/abs/2006.07846.

62. ^Abu-El-Haija S, Perozzi B, Kapoor A, Alipourfard N, Lerman K, Harutyunyan H, Ver Steeg G, Galstyan A (2019). "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing". arXiv preprint arXiv:1905.00067.

63. ^You J, Gomes-Selman JM, Ying R, Leskovec J (2021). "Identity-aware graph neural networks". Proceedings of the AAAI Conference on Artificial Intelligence. **35** (12): 10737–10745.

64. ^Koren Y, Bell R, Volinsky C (2009). "Matrix factorization techniques for recommender systems". Computer. **42** (8): 30–37.

65. a, b, c Menon AK, Elkan C. Link prediction via matrix factorization. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer; 2011. p. 437-452.

66. a, b, c Chen G, Wang H, Fang Y, Jiang L (2022). "Link prediction by deep non-negative matrix factorization" Expert Systems with Applications. **188**: 115991.

67. a, b, c, d Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. 2014. p. 701–710.

68. a, b Grover A, Leskovec J (2016). "node2vec: Scalable feature learning for networks". In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864.

69. a, b, c, d Galkin M, Wu J, Denis E, Hamilton WL (2021). "Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs". arXiv preprint arXiv:2106.12144. Available from:

https://arxiv.org/abs/2106.12144.

70. ^Yao X, Shao Y, Cui B, Chen L. "Uninet: Scalable network representation learning with metropolis-hastings sampling." In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE; 2021. p. 516-527.

71. ^Chib S, Greenberg E (1995). "Understanding the metropolis-hastings algorithm". The American Statistician. **49** (4): 327–335.

72. ^Sennrich R, Haddow B, Birch A. Neural machine translation of rare words with subword units. InProceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2016. p. 1715–1725.

73. ^Cai L, Li J, Wang J, Ji S (2021). "Line graph neural networks for link prediction" IEEE Transactions on Pattern Analysis and Machine Intelligence. **2021**. Published by IEEE.

74. ^Pan L, Shi C, Dokmanić I (2021). "Neural Link Prediction with Walk Pooling" arXiv preprint arXiv:2110.04375. Available from: *https://arxiv.org/abs/2110.04375*.

75. a, b, c Ai B, Qin Z, Shen W, Li Y (2022). "Structure Enhanced Graph Neural Networks for Link Prediction" arXiv preprint arXiv:2201.05293. 2022.

76. ^Zhang S, Zhang J, Song X, Adeshina S, Zheng D, Faloutsos C, Sun Y (2023). "PaGE-Link: Path-based graph neural network explanation for heterogeneous link prediction". Proceedings of the ACM Web Conference 2023. 2023: 3784--3793.

77. a, b, c Yun S, Kim S, Lee J, Kang J, Kim HJ (2021). "Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction". Advances in Neural Information Processing Systems **34**: 13683–13694.

78. ^Roy I, De A, Chakrabarti S (2021). "Adversarial permutation guided node representations for link prediction". Proceedings of the AAAI conference on artificial intelligence **35** (11): 9445–9453.

79. ^Wu E, Cui H, Chen Z (2022). "RelpNet: Relation-based Link Prediction Neural Network". In:Proceedings of the 31st ACM International Conference on Information & Knowledge Management. pp. 2138–2147.

80. ^Shang Y, Hao Z, Yao C, Li G (2022). "Improving Graph Neural Network Models in Link Prediction Task via A Policy-Based Training Method". Applied Sciences. **13** (1): 297.

81. ^Li J, Sun X, Li Y, Li Z, Cheng H, Yu JX. "Graph intelligence with large language models and prompt learning." In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2024. p. 6545-6554.

82. ^Schlichtkrull M, Kipf TN, Bloem P, van den Berg R, Titov I, Welling M. Modeling relational data with graph convolutional networks. In: European semantic web conference. Springer; 2018. p. 593–607.

83. ^Langville AN, Meyer CD. "Google's PageRank and beyond." In:Google's PageRank and Beyond. Princeton university press; 2011.