Chapter 10
Graph Neural Networks: Link Prediction

Muhan Zhang

Abstract Link prediction is an important application of graph neural networks. By predicting missing or future links between pairs of nodes, link prediction is widely used in social networks, citation networks, biological networks, recommender systems, and security, etc. Traditional link prediction methods rely on heuristic node similarity scores, latent embeddings of nodes, or explicit node features. Graph neural network (GNN), as a powerful tool for jointly learning from graph structure and node/edge features, has gradually shown its advantages over traditional methods for link prediction. In this chapter, we discuss GNNs for link prediction. We first introduce the link prediction problem and review traditional link prediction methods. Then, we introduce two popular GNN-based link prediction paradigms, node-based and subgraph-based approaches, and discuss their differences in link representation power. Finally, we review recent theoretical advancements on GNN-based link prediction and provide several future directions.

10.1 Introduction

Link prediction is the problem of predicting the existence of a link between two nodes in a network (Liben-Nowell and Kleinberg, 2007). Given the ubiquitous existence of networks, it has many applications such as friend recommendation in social networks (Adamic and Adar, 2003), co-authorship prediction in citation networks (Shibata et al, 2012), movie recommendation in Netflix (Bennett et al, 2007), protein interaction prediction in biological networks (Qi et al, 2006), drug response prediction (Stanfield et al, 2017), metabolic network reconstruction (Oyetunde et al, 2017), hidden terrorist group identification (Al Hasan and Zaki, 2011), knowledge graph completion (Nickel et al, 2016a), etc.

Muhan Zhang
Institute for Artificial Intelligence, Peking University, e-mail: muhan@pku.edu.cn

Link prediction has many names in different application domains. The term "link prediction" often refers to predicting links in homogeneous graphs, where nodes and links both only have a single type. This is the simplest setting and most link prediction works focus on this setting. Link prediction in bipartite user-item networks is referred to as matrix completion or recommender systems, where nodes have two types (user and item) and links can have multiple types corresponding to different ratings users can give to items. Link prediction in knowledge graphs is often referred to as knowledge graph completion, where each node is a distinct entity and links have multiple types corresponding to different relations between entities. In most cases, a link prediction algorithm designed for the homogeneous graph setting can be easily generalized to heterogeneous graphs (e.g., bipartite graphs and knowledge graphs) by considering heterogeneous node type and relation type information.

There are mainly three types of traditional link prediction methods: heuristic methods, latent-feature methods, and content-based methods. Heuristic methods compute heuristic node similarity scores as the likelihood of links (Liben-Nowell and Kleinberg, 2007). Popular ones include common neighbors (Liben-Nowell and Kleinberg, 2007), Adamic-Adar (Adamic and Adar, 2003), preferential attachment (Barabási and Albert, 1999), and Katz index (Katz, 1953). Latent-feature methods factorize the matrix representations of a network to learn low-dimensional latent representations/embeddings of nodes. Popular network embedding techniques such as DeepWalk (Perozzi et al, 2014), LINE (Tang et al, 2015b) and node2vec (Grover and Leskovec, 2016), are also latent-feature methods because they implicitly factorize some matrix representations of networks too (Qiu et al, 2018). Both heuristic methods and latent-feature methods infer future/missing links leveraging the existing network topology. Content-based methods, on the contrary, leverage explicit node attributes/features rather than the graph structure (Lops et al, 2011). It is shown that combining the graph topology with explicit node features can improve the link prediction performance (Zhao et al, 2017).

By learning from graph topology and node/edge features in a unified way, graph neural networks (GNNs) recently show superior link prediction performance than traditional methods (Kipf and Welling, 2016; Zhang and Chen, 2018b; You et al, 2019; Chami et al, 2019; Li et al, 2020e). There are two popular GNN-based link prediction paradigms: node-based and subgraph-based. Node-based methods first learn a node representation through a GNN, and then aggregate the pairwise node representations as link representations for link prediction. An example is (Variational) Graph AutoEncoder (Kipf and Welling, 2016). Subgraph-based methods first extract a local subgraph around each target link, and then apply a graph-level GNN (with pooling) to each subgraph to learn a subgraph representation, which is used as the target link representation for link prediction. An example is SEAL (Zhang and Chen, 2018b). We introduce these two types of methods separately in Section 10.3.1 and 10.3.2, and discuss their expressive power differences in Section 10.3.3.

To understand GNNs' power for link prediction, several theoretical efforts have been made. The γ-decaying theory (Zhang and Chen, 2018b) unifies existing link prediction heuristics into a single framework and proves their local approximability, which justifies using GNNs to "learn" heuristics from the graph structure instead of

using predefined ones. The theoretical analysis of labeling trick (Zhang et al, 2020c) proves that subgraph-based approaches have a higher link representation power than node-based approaches by being able to learn most expressive structural representations of links (Srinivasan and Ribeiro, 2020b) where node-based approaches always fail. We introduce these theories in Section 20.3.

Finally, by analyzing limitations of existing methods, we provide several future directions on GNN-based link prediction in Section 20.4.

10.2 Traditional Link Prediction Methods

In this section, we review traditional link prediction methods. They can be categorized into three classes: heuristic methods, latent-feature methods, and content-based methods.

10.2.1 Heuristic Methods

Heuristic methods use simple yet effective node similarity scores as the likelihood of links (Liben-Nowell and Kleinberg, 2007; Lü and Zhou, 2011). We use x and y to denote the source and target node between which to predict a link. We use $\Gamma(x)$ to denote the set of x's neighbors.

10.2.1.1 Local Heuristics

One simplest heuristic is called **common neighbors** (CN), which counts the number of neighbors two nodes share as a measurement of their likelihood of having a link:

$$f_{\text{CN}}(x,y) = |\Gamma(x) \cap \Gamma(y)|. \tag{10.1}$$

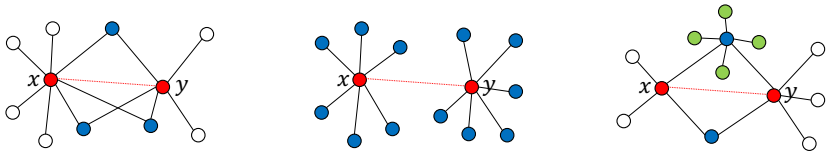
CN is widely used in social network friend recommendation. It assumes that the more common friends two people have, the more likely they themselves are also friends.

Jaccard score measures the proportion of common neighbors instead:

$$f_{\text{Jaccard}}(x,y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}. \tag{10.2}$$

There is also a famous **preferential attachment** (PA) heuristic (Barabási and Albert, 1999), which uses the product of node degrees to measure the link likelihood:

$$f_{\text{PA}}(x,y) = |\Gamma(x)| \cdot |\Gamma(y)|. \tag{10.3}$$



common neighbors (CN): $|\Gamma(x) \cap \Gamma(y)|$ | preferential attachment (PA): $|\Gamma(x)| \cdot |\Gamma(y)|$ | Adamic-Adar (AA): $\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}$

Fig. 10.1: Illustration of three link prediction heuristics: CN, PA and AA.

PA assumes x is more likely to connect to y if y has a high degree. For example, in citation networks, a new paper is more likely to cite those papers which already have a lot of citations. Networks formed by the PA mechanism are called scale-free networks (Barabási and Albert, 1999), which are important subjects in network science.

Existing heuristics can be categorized based on the maximum hop of neighbors needed to calculate the score. CN, Jaccard, and PA are all *first-order heuristics*, because they only involve one-hop neighbors of two target nodes. Next we introduce two *second-order heuristics*.

The **Adamic-Adar** (AA) heuristic (Adamic and Adar, 2003) considers the weight of common neighbors:

$$f_{\text{AA}}(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}, \tag{10.4}$$

where a high-degree common neighbor z is weighted less (down-weighted by the reciprocal of $\log |\Gamma(z)|$). The assumption is that a high degree node connecting to both x and y is less informative than a low-degree node.

Resource allocation (RA) (Zhou et al, 2009) uses a more aggressive down-weighting factor:

$$f_{\text{RA}}(x,y) = \sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{|\Gamma(z)|}, \tag{10.5}$$

thus, it favors low-degree common neighbors more.

Both AA and RA are second-order heuristics, as up to two hops of neighbors of x and y are required to compute the score. Both first-order and second-order heuristics are local heuristics, as they can all be computed from a local subgraph around the target link without the need to know the entire network. We illustrate three local heuristics, CN, PA, and AA, in Figure 10.1.

10.2.1.2 Global Heuristics

There are also **high-order heuristics** which require knowing the entire network. Examples include Katz index (Katz, 1953), rooted PageRank (RPR) (Brin and Page, 2012), and SimRank (SR) (Jeh and Widom, 2002).

Katz index uses a weighted sum of all the walks between x and y where a longer walk is discounted more:

$$f_{\text{Katz}}(x,y) = \sum_{l=1}^{\infty} \beta^l |\text{walks}^{\langle l \rangle}(x,y)|. \tag{10.6}$$

Here β is a decaying factor between 0 and 1, and $|\text{walks}^{\langle l \rangle}(x,y)|$ counts the length-l walks between x and y. When we only consider length-2 walks, Katz index reduces to CN.

Rooted PageRank (RPR) is a generalization of PageRank. It first computes the stationary distribution π_x of a random walker starting from x who randomly moves to one of its current neighbors with probability α, or returns to x with probability $1-\alpha$. Then it uses π_x at node y (denoted by $[\pi_x]_y$) to predict link (x,y). When the network is undirected, a symmetric version of rooted PageRank uses

$$f_{\text{RPR}}(x,y) = [\pi_x]_y + [\pi_y]_x \tag{10.7}$$

to predict the link.

The **SimRank** (SR) score assumes that two nodes are similar if their neighbors are also similar. It is defined in a recursive way: if $x = y$, then $f_{\text{SR}}(x,y) := 1$; otherwise,

$$f_{\text{SR}}(x,y) := \gamma \frac{\sum_{a \in \Gamma(x)} \sum_{b \in \Gamma(y)} f_{\text{SR}}(a,b)}{|\Gamma(x)| \cdot |\Gamma(y)|}, \tag{10.8}$$

where γ is a constant between 0 and 1.

High-order heuristics are global heuristics. By computing node similarity from the entire network, high-order heuristics often have better performance than first-order and second-order heuristics.

10.2.1.3 Summarization

We summarize the eight introduced heuristics in Table 10.1. For more variants of the above heuristics, please refer to (Liben-Nowell and Kleinberg, 2007; Lü and Zhou, 2011). Heuristic methods can be regarded as computing predefined **graph structure features** located in the observed node and edge structures of the network. Although effective in many domains, these handcrafted graph structure features have limited expressivity—they only capture a small subset of all possible structure patterns, and cannot express general graph structure features underlying different networks. Besides, heuristic methods only work well when the network formation mechanism

aligns with the heuristic. There may exist networks with complex formation mechanisms which no existing heuristics can capture well. Most heuristics only work for homogeneous graphs.

Table 10.1: Popular heuristics for link prediction

Name	Formula	Order
common neighbors	$\lvert\Gamma(x)\cap\Gamma(y)\rvert$	first
Jaccard	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$	first
preferential attachment	$\lvert\Gamma(x)\rvert\cdot\lvert\Gamma(y)\rvert$	first
Adamic-Adar	$\sum_{z\in\Gamma(x)\cap\Gamma(y)}\frac{1}{\log\lvert\Gamma(z)\rvert}$	second
resource allocation	$\sum_{z\in\Gamma(x)\cap\Gamma(y)}\frac{1}{\lvert\Gamma(z)\rvert}$	second
Katz	$\sum_{l=1}^{\infty}\beta^l\lvert\text{walks}^{\langle l\rangle}(x,y)\rvert$	high
rooted PageRank	$[\pi_x]_y+[\pi_y]_x$	high
SimRank	$\gamma\frac{\sum_{a\in\Gamma(x)}\sum_{b\in\Gamma(y)}\text{score}(a,b)}{\lvert\Gamma(x)\rvert\cdot\lvert\Gamma(y)\rvert}$	high

Notes: $\Gamma(x)$ denotes the neighbor set of vertex x. $\beta<1$ is a damping factor. $\lvert\text{walks}^{\langle l\rangle}(x,y)\rvert$ counts the number of length-l walks between x and y. $[\pi_x]_y$ is the stationary distribution probability of y under the random walk from x with restart, see (Brin and Page, 2012). SimRank score uses a recursive definition.

10.2.2 Latent-Feature Methods

The second class of traditional link prediction methods is called latent-feature methods. In some literature, they are also called latent-factor models or embedding methods. Latent-feature methods compute latent properties or representations of nodes, often obtained by factorizing a specific matrix derived from the network, such as the adjacency matrix and the Laplacian matrix. These latent features of nodes are not explicitly observable—they must be computed from the network through optimization. Latent features are also not interpretable. That is, unlike explicit node features where each feature dimension represents a specific property of nodes, we do not know what each latent feature dimension describes.

10.2.2.1 Matrix Factorization

One most popular latent feature method is matrix factorization (Koren et al, 2009; Ahmed et al, 2013), which is originated from the recommender systems literature. Matrix factorization factorizes the observed adjacency matrix A of the network into

the product of a low-rank latent-embedding matrix Z and its transpose. That is, it approximately reconstructs the edge between i and j using their k-dimensional latent embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$:

$$\hat{A}_{i,j} = \mathbf{z}_i^\top \mathbf{z}_j, \tag{10.9}$$

It then minimizes the mean-squared error between the reconstructed adjacency matrix and the true adjacency matrix over the observed edges to learn the latent embeddings:

$$\mathscr{L} = \frac{1}{|\mathscr{E}|} \sum_{(i,j)\in\mathscr{E}} (A_{i,j} - \hat{A}_{i,j})^2. \tag{10.10}$$

Finally, we can predict new links by the inner product between nodes' latent embeddings. Variants of matrix factorization include using powers of A (Cangea et al, 2018) and using general node similarity matrices (Ou et al, 2016) to replace the original adjacency matrix A. If we replace A with the Laplacian matrix L and define the loss as follows:

$$\mathscr{L} = \sum_{(i,j)\in\mathscr{E}} \|\mathbf{z}_i - \mathbf{z}_j\|_2^2, \tag{10.11}$$

then the nontrivial solution to the above are constructed by the eigenvectors corresponding to the k smallest nonzero eigenvalues of L, which recovers the Laplacian eigenmap technique (Belkin and Niyogi, 2002) and the solution to spectral clustering (VONLUXBURG, 2007).

10.2.2.2 Network Embedding

Network embedding methods have gained great popularity in recent years since the pioneering work DeepWalk (Perozzi et al, 2014). These methods learn low-dimensional representations (embeddings) for nodes, often based on training a skip-gram model (Mikolov et al, 2013a) over random-walk-generated node sequences, so that nodes which often appear nearby each other in a random walk (i.e., nodes close in the network) will have similar representations. Then, the pairwise node embeddings are aggregated as link representations for link prediction. Although not explicitly factorizing a matrix, it is shown in (Qiu et al, 2018) that many network embedding methods, including LINE (Tang et al, 2015b), DeepWalk, and node2vec (Grover and Leskovec, 2016), implicitly factorize some matrix representations of the network. Thus, they can also be categorized into latent-feature methods. For example, DeepWalk approximately factorizes:

$$\log\Big(\mathrm{vol}(\mathscr{G})\big(\frac{1}{w}\sum_{r=1}^{w}(D^{-1}A)^r\big)D^{-1}\Big) - \log(b), \tag{10.12}$$

where $\text{vol}(\mathcal{G})$ is the sum of node degrees, D is the diagonal degree matrix, w is skip-gram's window size, and b is a constant. As we can see, DeepWalk essentially factorizes the log of some high-order normalized adjacency matrices' sum (up to w). To intuitively understand this, we can think of the random walk as extending a node's neighborhood to w hops away, so that we not only require direct neighbors to have similar embeddings, but also require nodes reachable from each other through w steps of random walk to have similar embeddings.

Similarly, the LINE algorithm (Tang et al, 2015b) in its second order forms implicitly factorizes:

$$\log\left(\text{vol}(\mathcal{G})(D^{-1}AD^{-1})\right) - \log(b). \tag{10.13}$$

Another popular network embedding method, node2vec, which enhances DeepWalk with negative sampling and biased random walk, is also shown to implicitly factorize a matrix. The matrix does not have a closed form due to the use of second-order (biased) random walk (Qiu et al, 2018).

10.2.2.3 Summarization

We can understand latent-feature methods as extracting low-dimensional node embeddings from the graph structure. Traditional matrix factorization methods use the inner product between node embeddings to predict links. However, we are actually not restricted to inner product. Instead, we can apply a neural network over an arbitrary aggregation of pairwise node embeddings to learn link representations. For example, node2vec (Grover and Leskovec, 2016) provides four symmetric aggregation functions (invariant to the order of two nodes): mean, Hadamard product, absolute difference, and squared difference. If we predict directed links, we can also use non-symmetric aggregation functions, such as concatenation.

Latent-feature methods can take global properties and long-range effects into node representations, because all node pairs are used together to optimize a single objective function, and the final embedding learned for a node can be influenced by all nodes in the same connected component during the optimization. However, latent-feature methods cannot capture structural similarities between nodes Ribeiro et al (2017), i.e., two nodes sharing identical neighborhood structures are not mapped to similar embeddings. Latent-feature methods also need an extremely large dimension to express some simple heuristics (Nickel et al, 2014), making them sometimes have worse performance than heuristic methods. Finally, latent-feature methods are transductive learning methods—the learned node embeddings cannot generalize to new nodes or new networks.

There are many latent-feature methods designed for heterogeneous graphs. For example, the RESCAL model (Nickel et al, 2011) generalizes matrix factorization to multi-relation graphs, which essentially performs a kind of tensor factorization. Metapath2vec (Dong et al, 2017) generalizes node2vec to heterogeneous graphs.

10.2.3 Content-Based Methods

Both heuristic methods and latent-feature methods face the cold-start problem. That is, when a new node joins the network, heuristic methods and latent-feature methods may not be able to predict its links accurately because it has no or only a few existing links with other nodes. In this case, content-based methods might help. Content-based methods leverage explicit content features associated with nodes for link prediction, which have wide applications in recommender systems (Lops et al, 2011). For example, in citation networks, word distributions can be used as content features for papers. In social networks, a user's profile, such as their demographic information and interests, can be used as their content features (however, their friendship information belongs to graph structure features because it is calculated from the graph structure). However, content-based methods usually have worse performance than heuristic and latent-feature methods due to not leveraging the graph structure. Thus, they are usually used together with the other two types of methods (Koren, 2008; Rendle, 2010; Zhao et al, 2017) to enhance the link prediction performance.

10.3 GNN Methods for Link Prediction

In the last section, we have covered three types of traditional link prediction methods. In this section, we will talk about GNN methods for link prediction. GNN methods combine graph structure features and content features by learning them together in a unified way, leveraging the excellent graph representation learning ability of GNNs.

There are mainly two GNN-based link prediction paradigms, node-based and subgraph-based. Node-based methods aggregate the pairwise node representations learned by a GNN as the link representation. Subgraph-based methods extract a local subgraph around each link and use the subgraph representation learned by a GNN as the link representation.

10.3.1 Node-Based Methods

The most straightforward way of using GNNs for link prediction is to treat GNNs as inductive network embedding methods which learn node embeddings from local neighborhood, and then aggregates the pairwise node embeddings of GNNs to construct link representations. We call these methods node-based methods.

10.3.1.1 Graph AutoEncoder

The pioneering work of node-based methods is Graph AutoEncoder (GAE) (Kipf and Welling, 2016). Given the adjacency matrix A and node feature matrix X of a graph, GAE (Kipf and Welling, 2016) first uses a GCN (Kipf and Welling, 2017b) to compute a node representation $\mathbf{z}_i$ for each node i, and then uses $\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$ to predict link (i, j):

$$\hat{A}_{i,j} = \sigma(\mathbf{z}_i^\top \mathbf{z}_j), \text{ where } \mathbf{z}_i = Z_{i,:}, Z = \text{GCN}(X, A) \tag{10.14}$$

where Z is the node representation (embedding) matrix output by the GCN with the i^{th} row of Z being node i's representation $\mathbf{z}_i$, $\hat{A}_{i,j}$ is the predicted probability for link (i, j) and σ is the sigmoid function. If X is not given, GAE can use the one-hot encoding matrix I instead. The model is trained to minimize the cross entropy between the reconstructed adjacency matrix and the true adjacency matrix:

$$\mathscr{L} = \sum_{i \in \mathscr{V}, j \in \mathscr{V}} (-A_{i,j} \log \hat{A}_{i,j} - (1 - A_{i,j}) \log(1 - \hat{A}_{i,j})). \tag{10.15}$$

In practice, the loss of positive edges ($A_{i,j} = 1$) is up-weighted by k, where k is the ratio between negative edges ($A_{i,j} = 0$) and positive edges. The purpose is to balance the positive and negative edges' contribution to the loss. Otherwise, the loss might be dominated by negative edges due to the sparsity of practical networks.

10.3.1.2 Variational Graph AutoEncoder

The variational version of GAE is called VGAE, or Variational Graph AutoEncoder (Kipf and Welling, 2016). Rather than learning deterministic node embeddings $\mathbf{z}_i$, VGAE uses two GCNs to learn the mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\sigma}_i^2$ of $\mathbf{z}_i$, respectively.

VGAE assumes the adjacency matrix A is generated from the latent node embeddings Z through $p(A|Z)$, where Z follows a prior distribution $p(Z)$. Similar to GAE, VGAE uses an inner-product-based link reconstruction model as $p(A|Z)$:

$$p(A|Z) = \prod_{i \in \mathscr{V}} \prod_{j \in \mathscr{V}} p(A_{i,j}|\mathbf{z}_i, \mathbf{z}_j), \text{ where } p(A_{i,j} = 1|\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j). \tag{10.16}$$

And the prior distribution $p(Z)$ takes a standard Normal distribution:

$$p(Z) = \prod_{i \in \mathscr{V}} p(\mathbf{z}_i) = \prod_{i \in \mathscr{V}} \mathscr{N}(\mathbf{z}_i|0, I). \tag{10.17}$$

Given $p(A|Z)$ and $p(Z)$, we may compute the posterior distribution of Z using Bayes' rule. However, this distribution is often intractable. Thus, given the adjacency matrix A and node feature matrix X, VGAE uses graph neural networks to approximate the posterior distribution of the node embedding matrix Z:

$$q(Z|X,A) = \prod_{i \in \mathcal{V}} q(\mathbf{z}_i|X,A), \text{ where } q(\mathbf{z}_i|X,A) = \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_i, \text{diag}(\boldsymbol{\sigma}_i^2)). \quad (10.18)$$

Here, the mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\sigma}_i^2$ of $\mathbf{z}_i$ are given by two GCNs. Then, VGAE maximizes the evidence lower bound to learn the GCN parameters:

$$\mathcal{L} = \mathbb{E}_{q(Z|X,A)}[\log p(A|Z)] - \text{KL}[q(Z|X,A)||p(Z)], \quad (10.19)$$

where $\text{KL}[q(Z|X,A)||p(Z)]$ is the Kullback-Leibler divergence between the approximated posterior and the prior distribution of Z. The evidence lower bound is optimized using the reparameterization trick (Kingma and Welling, 2014). Finally, the embedding mean $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ are used to predict link (i, j) by $\hat{A}_{i,j} = \sigma(\boldsymbol{\mu}_i^\top \boldsymbol{\mu}_j)$.

10.3.1.3 Variants of GAE and VGAE

There are many variants of GAE and VGAE. For example, ARGE (Pan et al, 2018) enhances GAE with an adversarial regularization to regularize the node embeddings to follow a prior distribution. S-VAE (Davidson et al, 2018) replaces the Normal distribution in VGAE with a von Mises-Fisher distribution to model data with a hyperspherical latent structure. MGAE (Wang et al, 2017a) uses a marginalized graph autoencoder to reconstruct node features from corrupted ones through a GCN and applies it to graph clustering.

GAE represents a general class of node-based methods, where a GNN is first used to learn node embeddings and pairwise node embeddings are aggregated to learn link representations. In principle, we can replace the GCN used in GAE/VGAE with any GNN, and replace the inner product $\mathbf{z}_i^\top \mathbf{z}_j$ with any aggregation function over $\{\mathbf{z}_i, \mathbf{z}_j\}$ and feed the aggregated link representation to an MLP to predict the link (i, j). Following this methodology, we can generalize any GNN designed for learning node representations to link prediction. For example, HGCN (Chami et al, 2019) combines hyperbolic graph convolutional neural networks with a Fermi-Dirac decoder for aggregating pairwise node embeddings and outputting link probabilities:

$$p(A_{i,j} = 1|\mathbf{z}_i, \mathbf{z}_j) = [\exp(d(\mathbf{z}_i, \mathbf{z}_j) - r)/t + 1]^{-1}, \quad (10.20)$$

where $d(\cdot,\cdot)$ computes the hyperbolic distance and r, t are hyperparameters.

Position-aware GNN (PGNN) (You et al, 2019) aggregates messages only from some selected anchor nodes during the message passing to capture position information of nodes. Then, the inner product between node embeddings are used to predict links. The PGNN paper also generalizes other GNNs, including GAT (Petar et al, 2018), GIN (**?**) and GraphSAGE (Hamilton et al, 2017b), to the link prediction setting based on the inner-product decoder.

Many graph neural networks use link prediction as an objective for training node embeddings in an unsupervised manner, despite that their final task is still node classification. For example, after computing the node embeddings, GraphSAGE (Hamilton et al, 2017b) minimize the following objective for each $\mathbf{z}_i$ to encourage con-

nected or nearby nodes to have similar representations:

$$L(\mathbf{z}_i) = -\log\left(\sigma(\mathbf{z}_i^\top \mathbf{z}_j)\right) - k_n \cdot \mathbb{E}_{j' \sim p_n} \log\left(1 - \sigma(\mathbf{z}_i^\top \mathbf{z}_{j'})\right), \tag{10.21}$$

where j is a node co-occurs near i on some fixed-length random walk, p_n is the negative sampling distribution, and k_n is the number of negative samples. If we focus on length-2 random walks, the above loss reduces to a link prediction objective. Compared to the GAE loss in Equation (10.15), the above objective does not consider all $\mathcal{O}(n)$ negative links, but uses negative sampling instead to only consider k_n negative pairs (i, j') for each positive pair (i, j), thus is more suitable for large graphs.

In the context of recommender systems, there are also many node-based methods that can be seen as variants of GAE/VGAE. Monti et al (2017) use GNNs to learn user and item embeddings from their respective nearest-neighbor networks, and use the inner product between user and item embeddings to predict links. Berg et al (2017) propose the graph convolutional matrix completion (GC-MC) model which applies a GNN to the user-item bipartite graph to learn user and item embeddings. They use one-hot encoding of node indices as the input node features, and use the bilinear product between user and item embeddings to predict links. SpectralCF (Zheng et al, 2018a) uses a spectral-GNN on the bipartite graph to learn node embeddings. The PinSage model (Ying et al, 2018b) uses node content features as the input node features, and uses the GraphSAGE (Hamilton et al, 2017b) model to map related items to similar embeddings.

In the context of knowledge graph completion, R-GCN (Relational Graph Convolutional Neural Network) (Schlichtkrull et al, 2018) is one representative node-based method, which considers the relation types by giving different weights to different relation types during the message passing. SACN (Structure-Aware Convolutional Network) (Shang et al, 2019) performs message passing for each relation type's induced subgraphs individually and then uses a weighted sum of node embeddings from different relation types.

10.3.2 Subgraph-Based Methods

Subgraph-based methods extract a local subgraph around each target link and learn a subgraph representation through a GNN for link prediction.

10.3.2.1 The SEAL Framework

The pioneering work of subgraph-based methods is SEAL (Zhang and Chen, 2018b). SEAL first extracts an enclosing subgraph for each target link to predict, and then applies a graph-level GNN (with pooling) to classify whether the subgraph corresponds to link existence. The *enclosing subgraph* around a node set is defined as follows.

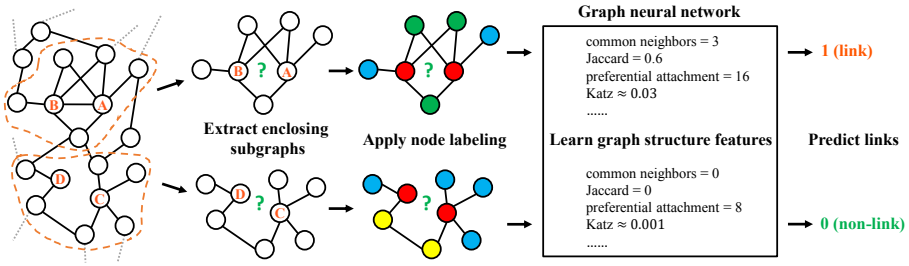


Fig. 10.2: Illustration of the SEAL framework. SEAL first extracts enclosing subgraphs around target links to predict. It then applies a node labeling to the enclosing subgraphs to differentiate nodes of different roles within a subgraph. Finally, the labeled subgraphs are fed into a GNN to learn graph structure features (supervised heuristics) for link prediction.

Definition 10.1. (Enclosing subgraph) For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, given a set of nodes $S \subseteq \mathcal{V}$, the h-hop enclosing subgraph for S is the subgraph $\mathcal{G}_S^h$ induced from $\mathcal{G}$ by the set of nodes $\cup_{j \in S}\{i \mid d(i,j) \leq h\}$, where $d(i,j)$ is the shortest path distance between nodes i and j.

In other words, the h-hop enclosing subgraph around a node set S contains nodes within h hops of any node in S, as well as all the edges between these nodes. In some literature, it is also called h-hop local/rooted subgraph, or h-hop ego network. In link prediction tasks, the node set S denotes the two nodes between which to predict a link. For example, when predicting the link between x and y, $S = \{x, y\}$ and $\mathcal{G}_{x,y}^h$ denotes the h-hop enclosing subgraph for link (x, y).

The motivation for extracting an enclosing subgraph for each link should be that SEAL aims to automatically learn graph structure features from the network. Observing that all first-order heuristics can be computed from the 1-hop enclosing subgraph around the target link and all second-order heuristics can be computed from the 2-hop enclosing subgraph around the target link, SEAL aims to use a GNN to learn general graph structure features (supervised heuristics) from the extracted h-hop enclosing subgraphs instead of using predefined heuristics.

After extracting the enclosing subgraph $G_{x,y}^h$, the next step is **node labeling**. SEAL applies a Double Radius Node Labeling (DRNL) to give an integer label to each node in the subgraph as its additional feature. The purpose is to use different labels to differentiate nodes of different roles in the enclosing subgraph. For instance, the center nodes x and y are the target nodes between which the target link is located, thus they are different from the rest nodes and should be distinguished. Similarly, nodes at different hops w.r.t. x and y may have different structural importance to the link existence, thus can also be assigned different labels. As discussed in Section 10.4.2, a proper node labeling such as DRNL is crucial for the success of subgraph-based link prediction methods, which makes subgraph-based methods have a higher link representation learning ability than node-based methods.

DRNL works as follows: First, assign label 1 to x and y. Then, for any node i with radius $(d(i,x),d(i,y)) = (1,1)$, assign label 2. Nodes with radius $(1,2)$ or $(2,1)$ get label 3. Nodes with radius $(1,3)$ or $(3,1)$ get 4. Nodes with $(2,2)$ get 5. Nodes with $(1,4)$ or $(4,1)$ get 6. Nodes with $(2,3)$ or $(3,2)$ get 7. So on and so forth. In other words, DRNL iteratively assigns larger labels to nodes with a larger radius w.r.t. the two center nodes.

DRNL satisfies the following criteria: 1) The two target nodes x and y always have the distinct label "1" so that they can be distinguished from the context nodes. 2) Nodes i and j have the same label if and only if their "double radius" are the same, i.e., i and j have the same distances to (x,y). This way, nodes of the same relative positions within the subgraph (described by the double radius $(d(i,x),d(i,y))$) always have the same label.

DRNL has a closed-form solution for directly mapping $(d(i,x),d(i,y))$ to labels:

$$l(i) = 1 + \min(d_x,d_y) + (d/2)[(d/2) + (d\%2) - 1], \tag{10.22}$$

where $d_x := d(i,x)$, $d_y := d(i,y)$, $d := d_x + d_y$, $(d/2)$ and $(d\%2)$ are the integer quotient and remainder of d divided by 2, respectively. For nodes with $d(i,x) = \infty$ or $d(i,y) = \infty$, DRNL gives them a null label 0.

After getting the DRNL labels, SEAL transforms them into one-hot encoding vectors, or feed them to an embedding layer to get their embeddings. These new feature vectors are concatenated with the original node content features (if any) to form the new node features. SEAL additionally allows concatenating some pretrained node embeddings such as node2vec embeddings to node features. However, as its experimental results show, adding pretrained node embeddings does not show clear benefits to the final performance (Zhang and Chen, 2018b). Furthermore, adding pretrained node embeddings makes SEAL lose the inductive learning ability.

Finally, SEAL feeds these enclosing subgraphs as well as their new node feature vectors into a graph-level GNN, DGCNN (Zhang et al., 2018g), to learn a graph classification function. The groundtruth of each subgraph is whether the two center nodes really have a link. To train this GNN, SEAL randomly samples N existing links from the network as positive training links, and samples an equal number of unobserved links (random node pairs) as negative training links. After training, SEAL applies the trained GNN to new unobserved node pairs' enclosing subgraphs to predict their links. The entire SEAL framework is illustrated in Figure 10.2. SEAL achieves strong performance for link prediction, demonstrating consistently superior performance than predefined heuristics (Zhang and Chen, 2018b).

10.3.2.2 Variants of SEAL

SEAL inspired many follow-up works. For example, Cai and Ji (2020) propose to use enclosing subgraphs of different scales to learn scale-invariant models. Li et al. (2020e) propose Distance Encoding (DE) which generalizes DRNL to node classification and general node set classification problems and theoretically analyzes the

power it brings to GNNs. The line graph link prediction (LGLP) model (Cai et al, 2020c) transforms each enclosing subgraph into its line graph and uses the center node embedding in the line graph to predict the original link.

SEAL is also generalized to the bipartite graph link prediction problem of recommender systems (Zhang and Chen, 2019). The model is called Inductive Graph-based Matrix Completion (IGMC). IGMC also samples an enclosing subgraph around each target (user, item) pair, but uses a different node labeling scheme. For each enclosing subgraph, it first gives label 0 and label 1 to the target user and the target item, respectively. The remaining nodes' labels are determined based on both their node types and their distances to the target user and item: if a user-type node's shortest path to reach either the target user or the target item has a length k, it will get a label $2k$; if an item-type node's shortest path to reach the target user or the target item has a length k, it will get a label $2k+1$. This way, the target nodes can always be distinguished from the context nodes, and users can be distinguished from items (users always have even labels). Furthermore, nodes of different distances to the center nodes can be differentiated, too. Finally, the enclosing subgraphs are fed into a GNN with R-GCN convolution layers to incorporate the edge type information (each edge type corresponds to a different rating). And the output representations of the target user and the target item are concatenated as the link representation to predict the target rating. IGMC is an inductive matrix completion model without relying on any content features, i.e., the model predicts ratings based only on local graph structures, and the learned model can transfer to unseen users/items or new tasks without retraining.

In the context of knowledge graph completion, SEAL is generalized to GraIL (Graph Inductive Learning) (Teru et al, 2020). It also follows the enclosing subgraph extraction, node labeling, and GNN prediction framework. For enclosing subgraph extraction, it extracts the subgraph induced by all the nodes that occur on at least one path of length at most $h+1$ between the two target nodes. Unlike SEAL, the enclosing subgraph of GraIL does not include those nodes that are only neighbors of one target node but are not neighbors of the other target node. This is because for knowledge graph reasoning, paths connecting two target nodes are of extra importance than dangling nodes. After extracting the enclosing subgraphs, GraIL applies DRNL to label the enclosing subgraphs and uses a variant of R-GCN by enhancing R-GCN with edge attention to output the score for each link to predict.

10.3.3 Comparing Node-Based Methods and Subgraph-Based Methods

At first glance, both node-based methods and subgraph-based methods learn graph structure features around target links based on a GNN. However, as we will show, subgraph-based methods actually have a higher link representation ability than node-based methods due to modeling the associations between two target nodes.

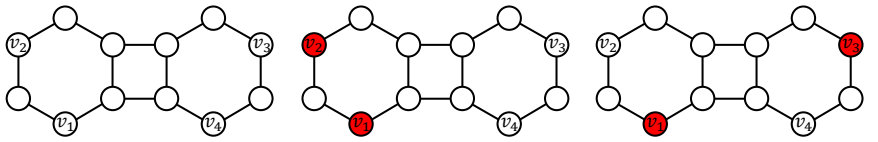


Fig. 10.3: The different link representation ability between node-based methods and subgraph-based methods. In the left graph, nodes v_2 and v_3 are isomorphic; links (v_1, v_2) and (v_4, v_3) are isomorphic; link (v_1, v_2) and link (v_1, v_3) are **not** isomorphic. However, a node-based method cannot differentiate (v_1, v_2) and (v_1, v_3). In the middle graph, when we predict (v_1, v_2), we label these two nodes differently from the rest, so that a GNN is aware of the target link when learning v_1 and v_2's representations. Similarly, when predicting (v_1, v_3), nodes v_1 and v_3 will be labeled differently (shown in the right graph). This way, the representation of v_2 in the left graph will be different from the representation of v_3 in the right graph, enabling GNNs to distinguish (v_1, v_2) and (v_1, v_3).

We first use an example to show node-based methods' limitation for detecting associations between two target nodes. Figure 10.3 left shows a graph we want to perform link prediction on. In this graph, nodes v_2 and v_3 are isomorphic (symmetric to each other), and links (v_1, v_2) and (v_4, v_3) are also isomorphic. However, link (v_1, v_2) and link (v_1, v_3) are **not** isomorphic, as they are not symmetric in the graph. In fact, v_1 is much closer to v_2 than v_3 in the graph, and shares more common neighbors with v_2. Thus, intuitively we do not want to predict (v_1, v_2) and (v_1, v_3) the same. However, because v_2 and v_3 are isomorphic, a node-based method will learn the same node representation for v_2 and v_3 (due to identical neighborhoods). Then, because node-based methods aggregate two node representations as a link representation, they will learn the same link representation for (v_1, v_2) and (v_1, v_3) and subsequently output the same link existence probability for them. This is clearly not what we want.

The root cause of this issue is that node-based methods compute two node representations **independently** of each other, without considering the relative positions and associations between the two nodes. For example, although v_2 and v_3 have different relative positions w.r.t. v_1, a GNN for learning v_2 and v_3's representations is unaware of this difference by treating v_2 and v_3 symmetrically.

With node-based methods, GNNs **cannot even learn to count the common neighbors** between two nodes (which is 1 for (v_1, v_2) and 0 for (v_1, v_3)), one of the most fundamental graph structure features for link prediction. This is still because node-based methods do not consider the other target node when computing one target node's representation. For example, when computing the representation of v_1, node-based methods do not care about which is the other target node—no matter whether the other node has dense connections with it (like v_2) or is far away from it (like v_3), node-based methods will learn the same representation for v_1. The failure to model the associations between two target nodes sometimes results in bad link prediction performance.

Different from node-based methods, subgraph-based methods perform link prediction by extracting an enclosing subgraph around each target link. As we can see, if we extract 1-hop enclosing subgraphs for both (v_1, v_2) and (v_1, v_3), then they are immediately differentiable due to their different enclosing subgraph structures—the enclosing subgraph around (v_1, v_2) is a single connected component, while the enclosing subgraph around (v_1, v_3) is composed of two connected components. Most GNNs can easily assign these two subgraphs different representations.

In addition, the node labeling step in subgraph-based methods also helps model the associations between the two target nodes. For example, let us assume we do not extract enclosing subgraphs, but only apply a node labeling to the original graph. We assume a simplest node labeling which only distinguishes the two target nodes from the rest nodes by assigning label 1 to the two target nodes and label 0 to the rest nodes (we call it *zero-one labeling trick*). Then, when we want to predict link (v_1, v_2), we give v_1, v_2 a different label from those of the rest nodes, as shown by different colors in Figure 10.3 middle. With v_1 and v_2 labeled, when a GNN is computing v_2's representation, it is also "aware" of the source node v_1. And when we want to predict link (v_1, v_3), we will again give v_1, v_3 a different label, as shown in Figure 10.3 right. This way, v_2 and v_3's node representations are no longer the same in the two differently labeled graphs due to the presence of the labeled v_1, and we are able to give different predictions to (v_1, v_2) and (v_1, v_3). This method is called labeling trick (Zhang et al, 2020c). We will discuss it more thoroughly in Section 10.4.2.

10.4 Theory for Link Prediction

In this section, we will introduce some theoretical developments on GNN-based link prediction. For subgraph-based methods, one important motivation is to learn supervised heuristics (graph structure features) from links' neighborhoods. Then, an important question to ask is, how well can GNNs learn existing successful heuristics? The γ-decaying heuristic theory (Zhang and Chen, 2018b) answers this question. In Section 10.3.3, we have seen the limitation of node-based methods for modeling the associations and relationships between two target nodes, and we have also seen that a simple zero-one node labeling can help solve this problem. Why and how can such a simple labeling trick achieve such a better link representation learning ability? What are the general requirements for a node labeling scheme to achieve this ability? The analysis of *labeling trick* answers these questions (Zhang et al, 2020c).

10.4.1 γ-Decaying Heuristic Theory

When using GNNs for link prediction, we want to learn graph structure features useful for predicting links based on message passing. However, it is usually not

possible to use very deep message passing layers to aggregate information from the entire network due to the computation complexity introduced by neighbor explosion and the issue of oversmoothing (Li et al, 2018b). This is why node-based methods (such as GAE) only use 1 to 3 message passing layers in practice, and why subgraph-based methods only extract a small 1-hop or 2-hop local enclosing subgraph around each link.

The γ-decaying heuristic theory (Zhang and Chen, 2018b) mainly answers how much structural information useful for link prediction is preserved in local neighborhood of the link, in order to justify applying a GNN only to a local enclosing subgraph in subgraph-based methods. To answer this question, the γ-decaying heuristic theory studies how well can existing link prediction heuristics be approximated from local enclosing subgraphs. If all these existing successful heuristics can be accurately computed or approximated from local enclosing subgraphs, then we are more confident to use a GNN to learn general graph structure features from these local subgraphs.

10.4.1.1 Definition of γ-Decaying Heuristics

Firstly, a direct conclusion from the definition of h-hop enclosing subgraphs (Definition 10.1) is:

Proposition 10.1. *Any h-order heuristic score for (x,y) can be accurately calculated from the h-hop enclosing subgraph $\mathcal{G}^h_{x,y}$ around (x,y).*

For example, a 1-hop enclosing subgraph contains all the information needed to calculate any first-order heuristics, while a 2-hop enclosing subgraph contains all the information needed to calculate any first and second-order heuristics. This indicates that first and second-order heuristics can be learned from local enclosing subgraphs based on an expressive GNN. However, how about high-order heuristics? High-order heuristics usually have better link prediction performance than local ones. To study high-order heuristics' local approximability, the γ-decaying heuristic theory first defines a general formulation of high-order heuristics, namely the *γ-decaying heuristic*.

Definition 10.2. (γ-decaying heuristic) A γ-decaying heuristic for link (x,y) has the following form:

$$\mathscr{H}(x,y) = \eta \sum_{l=1}^{\infty} \gamma^{l} f(x,y,l), \tag{10.23}$$

where γ is a decaying factor between 0 and 1, η is a positive constant or a positive function of γ which is upper bounded by a constant, f is a nonnegative function of x,y,l under the the given network, and l can be understood as the iteration number.

Next, it proves that under certain conditions, any γ-decaying heuristic can be approximated from an h-hop enclosing subgraph, and the approximation error decreases at least exponentially with h.

Theorem 10.1. *Given a γ-decaying heuristic $\mathscr{H}(x,y) = \eta \sum_{l=1}^{\infty} \gamma^l f(x,y,l)$, if $f(x,y,l)$ satisfies:*

- *(property 1) $f(x,y,l) \leq \lambda^l$ where $\lambda < \frac{1}{\gamma}$; and*
- *(property 2) $f(x,y,l)$ is calculable from $\mathscr{G}^h_{x,y}$ for $l = 1,2,\cdots,g(h)$, where $g(h) = ah+b$ with $a,b \in \mathbb{N}$ and $a > 0$,*

then $\mathscr{H}(x,y)$ can be approximated from $\mathscr{G}^h_{x,y}$ and the approximation error decreases at least exponentially with h.

Proof. We can approximate such a γ-decaying heuristic by summing over its first $g(h)$ terms.

$$\widetilde{\mathscr{H}}(x,y) := \eta \sum_{l=1}^{g(h)} \gamma^l f(x,y,l). \tag{10.24}$$

The approximation error can be bounded as follows.

$$|\mathscr{H}(x,y) - \widetilde{\mathscr{H}}(x,y)| = \eta \sum_{l=g(h)+1}^{\infty} \gamma^l f(x,y,l) \leq \eta \sum_{l=ah+b+1}^{\infty} \gamma^l \lambda^l = \eta (\gamma\lambda)^{ah+b+1} (1-\gamma\lambda)^{-1}$$

The above proof indicates that a smaller $\gamma\lambda$ leads to a faster decaying speed and a smaller approximation error. To approximate a γ-decaying heuristic, one just needs to sum its first few terms calculable from an h-hop enclosing subgraph.

Then, a natural question to ask is which existing high-order heuristics belong to γ-decaying heuristics that allow local approximations. Surprisingly, the γ-decaying heuristic theory shows that three most popular high-order heuristics: Katz index, rooted PageRank and SimRank (listed in Table 10.1) are all γ-decaying heuristics which satisfy the properties in Theorem 10.1.

To prove these, we need the following lemma first.

Lemma 10.1. *Any walk between x and y with length $l \leq 2h+1$ is included in $\mathscr{G}^h_{x,y}$.*

Proof. Given any walk $w = \langle x, v_1, \cdots, v_{l-1}, y \rangle$ with length l, we will show that every node v_i is included in $\mathscr{G}^h_{x,y}$. Consider any v_i. Assume $d(v_i,x) \geq h+1$ and $d(v_i,y) \geq h+1$. Then, $2h+1 \geq l = |\langle x, v_1, \cdots, v_i \rangle| + |\langle v_i, \cdots, v_{l-1}, y \rangle| \geq d(v_i,x) + d(v_i,y) \geq 2h+2$, a contradiction. Thus, $d(v_i,x) \leq h$ or $d(v_i,y) \leq h$. By the definition of $\mathscr{G}^h_{x,y}$, v_i must be included in $\mathscr{G}^h_{x,y}$.

Next we present the analysis on Katz, rooted PageRank and SimRank.

10.4.1.2 Katz index

The Katz index (Katz, 1953) for (x,y) is defined as

$$\text{Katz}_{x,y} = \sum_{l=1}^{\infty} \beta^l |\text{walks}^{\langle l \rangle}(x,y)| = \sum_{l=1}^{\infty} \beta^l [A^l]_{x,y}, \tag{10.25}$$

where $\text{walks}^{\langle l \rangle}(x,y)$ is the set of length-l walks between x and y, and A^l is the l^{th} power of the adjacency matrix of the network. Katz index sums over the collection of all walks between x and y where a walk of length l is damped by β^l $(0 < \beta < 1)$, giving more weights to shorter walks.

Katz index is directly defined in the form of a γ-decaying heuristic with $\eta = 1, \gamma = \beta$, and $f(x,y,l) = |\text{walks}^{\langle l \rangle}(x,y)|$. According to Lemma 10.1, $|\text{walks}^{\langle l \rangle}(x,y)|$ is calculable from $\mathcal{G}^h_{x,y}$ for $l \leq 2h+1$, thus property 2 in Theorem 10.1 is satisfied. Now we show when property 1 is satisfied.

Proposition 10.2. *For any nodes i, j, $[A^l]_{i,j}$ is bounded by d^l, where d is the maximum node degree of the network.*

Proof. We prove it by induction. When $l = 1$, $A_{i,j} \leq d$ for any (i,j). Thus the base case is correct. Now, assume by induction that $[A^l]_{i,j} \leq d^l$ for any (i,j), we have

$$[A^{l+1}]_{i,j} = \sum_{k=1}^{|V|} [A^l]_{i,k} A_{k,j} \leq d^l \sum_{k=1}^{|V|} A_{k,j} \leq d^l d = d^{l+1}.$$

Taking $\lambda = d$, we can see that whenever $d < 1/\beta$, the Katz index will satisfy property 1 in Theorem 10.1. In practice, the damping factor β is often set to very small values like 5E-4 (Liben-Nowell and Kleinberg, 2007), which implies that Katz can be very well approximated from the h-hop enclosing subgraph.

10.4.1.3 PageRank

The rooted PageRank for node x calculates the stationary distribution of a random walker starting at x, who iteratively moves to a random neighbor of its current position with probability α or returns to x with probability $1-\alpha$. Let π_x denote the stationary distribution vector. Let $[\pi_x]_i$ denote the probability that the random walker is at node i under the stationary distribution.

Let P be the transition matrix with $P_{i,j} = \frac{1}{|\Gamma(v_j)|}$ if $(i,j) \in E$ and $P_{i,j} = 0$ otherwise. Let $\mathbf{e}_x$ be a vector with the x^{th} element being 1 and others being 0. The stationary distribution satisfies

$$\pi_x = \alpha P \pi_x + (1-\alpha)\mathbf{e}_x. \tag{10.26}$$

When used for link prediction, the score for (x,y) is given by $[\pi_x]_y$ (or $[\pi_x]_y + [\pi_y]_x$ for symmetry). To show that rooted PageRank is a γ-decaying heuristic, we introduce the *inverse P-distance* theory (Jeh and Widom, 2003), which states that $[\pi_x]_y$ can be equivalently written as follows:

$$[\pi_x]_y = (1-\alpha) \sum_{w: x \rightsquigarrow y} P[w] \alpha^{\text{len}(w)}, \tag{10.27}$$

where the summation is taken over all walks w starting at x and ending at y (possibly touching x and y multiple times). For a walk $w = \langle v_0, v_1, \cdots, v_k \rangle$, $\text{len}(w) := |\langle v_0, v_1, \cdots, v_k \rangle|$ is the length of the walk. The term $P[w]$ is defined as $\prod_{i=0}^{k-1} \frac{1}{|\Gamma(v_i)|}$, which can be interpreted as the probability of traveling w. Now we have the following theorem.

Theorem 10.2. *The rooted PageRank heuristic is a γ-decaying heuristic which satisfies the properties in Theorem 10.1.*

Proof. We first write $[\pi_x]_y$ in the following form.

$$[\pi_x]_y = (1-\alpha) \sum_{l=1}^{\infty} \sum_{\substack{w: x \rightsquigarrow y \\ \text{len}(w)=l}} P[w] \alpha^l. \tag{10.28}$$

Defining $f(x,y,l) := \sum_{\substack{w: x \rightsquigarrow y \\ \text{len}(w)=l}} P[w]$ leads to the form of a γ-decaying heuristic. Note that $f(x,y,l)$ is the probability that a random walker starting at x stops at y with exactly l steps, which satisfies $\sum_{z \in V} f(x,z,l) = 1$. Thus, $f(x,y,l) \leq 1 < \frac{1}{\alpha}$ (property 1). According to Lemma 10.1, $f(x,y,l)$ is also calculable from $\mathcal{G}^h_{x,y}$ for $l \leq 2h+1$ (property 2).

10.4.1.4 SimRank

The SimRank score (Jeh and Widom, 2002) is motivated by the intuition that two nodes are similar if their neighbors are also similar. It is defined in the following recursive way: if $x = y$, then $s(x,y) := 1$; otherwise,

$$s(x,y) := \gamma \frac{\sum_{a \in \Gamma(x)} \sum_{b \in \Gamma(y)} s(a,b)}{|\Gamma(x)| \cdot |\Gamma(y)|} \tag{10.29}$$

where γ is a constant between 0 and 1. According to (Jeh and Widom, 2002), SimRank has an equivalent definition:

$$s(x,y) = \sum_{w:(x,y)\multimap(z,z)} P[w]\gamma^{\text{len}(w)}, \tag{10.30}$$

where $w:(x,y)\multimap(z,z)$ denotes all simultaneous walks such that one walk starts at x, the other walk starts at y, and they first meet at any vertex z. For a simultaneous walk $w = \langle (v_0,u_0),\cdots,(v_k,u_k)\rangle$, $\text{len}(w) = k$ is the length of the walk. The term $P[w]$ is similarly defined as $\prod_{i=0}^{k-1} \frac{1}{|\Gamma(v_i)||\Gamma(u_i)|}$, describing the probability of this walk. Now we have the following theorem.

Theorem 10.3. *SimRank is a γ-decaying heuristic which satisfies the properties in Theorem 10.1.*

Proof. We write $s(x,y)$ as follows.

$$s(x,y) = \sum_{l=1}^{\infty} \sum_{\substack{w:(x,y)\multimap(z,z)\\ \text{len}(w)=l}} P[w]\gamma^{l}, \tag{10.31}$$

Defining $f(x,y,l) := \sum_{\substack{w:(x,y)\multimap(z,z)\\ \text{len}(w)=l}} P[w]$ reveals that SimRank is a γ-decaying heuristic. Note that $f(x,y,l) \leq 1 < \frac{1}{\gamma}$. It is easy to see that $f(x,y,l)$ is also calculable from $\mathcal{G}^h_{x,y}$ for $l \leq h$.

10.4.1.5 Discussion

There exist several other high-order heuristics based on path counting or random walk (Lü and Zhou, 2011) which can be as well incorporated into the γ-decaying heuristic framework. Another interesting finding is that first and second-order heuristics can be unified into this framework too. For example, common neighbors can be seen as a γ-decaying heuristic with $\eta = \gamma = 1$, and $f(x,y,l) = |\Gamma(x)\cap\Gamma(y)|$ for $l = 1$, $f(x,y,l) = 0$ otherwise.

The above results reveal that most existing link prediction heuristics inherently share the same γ-decaying heuristic form, and thus can be effectively approximated from an h-hop enclosing subgraph with exponentially smaller approximation error. The ubiquity of γ-decaying heuristics is not by accident—it implies that a successful link prediction heuristic is better to put exponentially smaller weight on structures far away from the target, as remote parts of the network intuitively make little contribution to link existence. The γ-decaying heuristic theory builds the foundation for learning supervised heuristics from local enclosing subgraphs, as they imply that local enclosing subgraphs already contain enough information to learn good graph structure features for link prediction which is much desired considering

learning from the entire network is often infeasible. This motivates the proposition of subgraph-based methods.

To summarize, from small enclosing subgraphs extracted around links, we are able to accurately calculate first and second-order heuristics, and approximate a wide range of high-order heuristics with small errors. Therefore, given a sufficiently expressive GNN, learning from such enclosing subgraphs is expected to achieve performance at least as good as a wide range of heuristics.

10.4.2 Labeling Trick

In Section 10.3.3, we have briefly discussed the difference between node-based methods' and subgraph-based methods' link representation learning abilities. This is formalized into the analysis of *labeling trick* (Zhang et al, 2020c).

10.4.2.1 Structural Representation

We first introduce some preliminary knowledge on *structural representation*, which is a core concept in the analysis of labeling trick.

We define a graph to be $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{A})$, where $\mathcal{V} = \{1, 2, \ldots, n\}$ is the set of n vertices, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathsf{A} \in \mathbb{R}^{n \times n \times k}$ is a 3-dimensional tensor (we call it adjacency tensor) containing node and edge features. The diagonal components $\mathsf{A}_{i,i,:}$ denote features of node i, and the off-diagonal components $\mathsf{A}_{i,j,:}$ denote features of edge (i, j). We further use $A \in \{0,1\}^{n \times n}$ to denote the adjacency matrix of $\mathcal{G}$ with $A_{i,j} = 1$ iff $(i, j) \in E$. If there are no node/edge features, we let $\mathsf{A} = A$. Otherwise, A can be regarded as the first slice of A, i.e., $A = \mathsf{A}_{:,:,1}$.

A *permutation* π is a bijective mapping from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$. Depending on the context, $\pi(i)$ can mean assigning a new index to node $i \in V$, or mapping node i to node $\pi(i)$ of another graph. All $n!$ possible π's constitute the permutation group Π_n. For joint prediction tasks over a set of nodes, we use S to denote the **target node set**. For example, $S = \{i, j\}$ if we want to predict the link between i, j. We define $\pi(S) = \{\pi(i) | i \in S\}$. We further define the permutation of A as $\pi(\mathsf{A})$, where $\pi(\mathsf{A})_{\pi(i),\pi(j),:} = \mathsf{A}_{i,j,:}$.

Next, we define *set isomorphism*, which generalizes graph isomorphism to arbitrary node sets.

Definition 10.3. (Set isomorphism) Given two n-node graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{A})$, $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathsf{A}')$, and two node sets $S \subseteq \mathcal{V}$, $S' \subseteq \mathcal{V}'$, we say (S, A) and (S', A') are isomorphic (denoted by $(S, \mathsf{A}) \simeq (S', \mathsf{A}')$) if $\exists \pi \in \Pi_n$ such that $S = \pi(S')$ and $\mathsf{A} = \pi(\mathsf{A}')$.

When $(\mathcal{V}, \mathsf{A}) \simeq (\mathcal{V}', \mathsf{A}')$, we say two graphs $\mathcal{G}$ and $\mathcal{G}'$ are *isomorphic* (abbreviated as $\mathsf{A} \simeq \mathsf{A}'$ because $\mathcal{V} = \pi(\mathcal{V}')$ for any π). Note that set isomorphism is **more strict** than graph isomorphism, because it not only requires graph isomorphism, but also requires that the permutation maps a specific node set S to another node set S'.

In practice, when $S \neq \mathcal{V}$, we are often more concerned with the case of $\mathsf{A} = \mathsf{A}'$, where we are to find isomorphic node sets **in the same graph** (automorphism). For example, when $S = \{i\}, S' = \{j\}$ and $(i, \mathsf{A}) \simeq (j, \mathsf{A})$, we say nodes i and j are isomorphic in graph A (or they have symmetric positions/same structural role within the graph). An example is v_2 and v_3 in Figure 10.3 left.

We say a function f defined over the space of (S, A) is *permutation invariant* (or *invariant* for abbreviation) if $\forall \pi \in \Pi_n$, $f(S, \mathsf{A}) = f(\pi(S), \pi(\mathsf{A}))$. Similarly, f is *permutation equivariant* if $\forall \pi \in \Pi_n$, $\pi(f(S, \mathsf{A})) = f(\pi(S), \pi(\mathsf{A}))$.

Now we define structural representation of a node set, following (Srinivasan and Ribeiro, 2020b; Li et al, 2020e). It assigns a unique representation to each equivalence class of isomorphic node sets.

Definition 10.4. (Most expressive structural representation) Given an invariant function $\Gamma(\cdot)$, $\Gamma(S, \mathsf{A})$ is a most expressive structural representation for (S, A) if $\forall S, \mathsf{A}, S', \mathsf{A}'$, $\Gamma(S, \mathsf{A}) = \Gamma(S', \mathsf{A}') \Leftrightarrow (S, \mathsf{A}) \simeq (S', \mathsf{A}')$.

For simplicity, we will briefly use *structural representation* to denote most expressive structural representation in the rest of this section. We will omit A if it is clear from context. We call $\Gamma(i, \mathsf{A})$ a *structural node representation* for i, and call $\Gamma(\{i, j\}, \mathsf{A})$ a *structural link representation* for (i, j).

Definition 10.4 requires the structural representations of two node sets to be the same if and only if they are isomorphic. That is, isomorphic node sets always have the **same** structural representation, while non-isomorphic node sets always have **different** structural representations. This is in contrast to *positional node embeddings* such as DeepWalk (Perozzi et al, 2014) and matrix factorization (Mnih and Salakhutdinov, 2008), where two isomorphic nodes can have different node embeddings (Ribeiro et al, 2017).

So why do we need structural representations? Formally speaking, Srinivasan and Ribeiro (2020b) prove that any joint prediction task over node sets only requires *most-expressive structural representations* of node sets, which are the same for two node sets if and only if these two node sets are isomorphic. This means, for link prediction tasks, we need to learn the same representation for isomorphic links while discriminating non-isomorphic links by giving them different representations. Intuitively speaking, two links being isomorphic means they should be indistinguishable from any perspective—if one link exists, the other should exist too, and vice versa. Therefore, link prediction ultimately requires finding such a *structural link representation* for node pairs which can uniquely identify link isomorphism classes.

According to Figure 10.3 left, node-based methods that directly aggregate two node representations **cannot** learn such a valid structural link representation because they cannot differentiate non-isomorphic links such as (v_1, v_2) and (v_1, v_3). One may wonder whether using one-hot encoding of node indices as the input node features help node-based methods learn such a structural link representation. Indeed, using node-discriminating features enables node-based methods to learn different representations for (v_1, v_2) and (v_1, v_3) in Figure 10.3 left. However, it also loses GNN's ability to map isomorphic nodes (such as v_2 and v_3) and isomorphic links (such as (v_1, v_2) and (v_4, v_3)) to the same representations, since any two nodes already

have different representations from the beginning. This might result in poor generalization ability—two nodes/links may have different final representations even they share identical neighborhoods.

To ease our analysis, we also define a *node-most-expressive GNN*, which gives different representations to all non-isomorphic nodes and gives the same representation to all isomorphic nodes. In other words, a node-most-expressive GNN learns structural node representations.

Definition 10.5. (Node-most-expressive GNN) A GNN is node-most-expressive if it satisfies: $\forall i, \mathsf{A}, j, \mathsf{A}', \ \mathrm{GNN}(i, \mathsf{A}) = \mathrm{GNN}(j, \mathsf{A}') \Leftrightarrow (i, \mathsf{A}) \simeq (j, \mathsf{A}')$.

Although a polynomial-time implementation of a node-most-expressive GNN is not known, practical GNNs based on message passing can still discriminate almost all non-isomorphic nodes (Babai and Kucera, 1979), thus well approximating its power.

10.4.2.2 Labeling Trick Enables Learning Structural Representations

Now, we are ready to introduce the labeling trick and see how it enables learning structural representations of node sets. As we have seen in Section 10.4.2, a simple zero-one labeling trick can help a GNN distinguish non-isomorphic links such as (v_1, v_2) and (v_1, v_3) in Figure 10.3 left. At the same time, isomorphic links, such as (v_1, v_2) and (v_4, v_3), will still have the same representation, since the zero-one labeled graph for (v_1, v_2) is still symmetric to the zero-one labeled graph for (v_4, v_3). This brings an exclusive advantage over using one-hot encoding of node indices.

Below we give the formal definition of labeling trick, which incorporates the zero-one labeling trick as one specific form.

Definition 10.6. (Labeling trick) Given (S, A), we stack a labeling tensor $\mathsf{L}^{(S)} \in \mathbb{R}^{n \times n \times d}$ in the third dimension of A to get a new $\mathsf{A}^{(S)} \in \mathbb{R}^{n \times n \times (k+d)}$, where L satisfies: $\forall S, \mathsf{A}, S', \mathsf{A}', \pi \in \Pi_n$, (1) $\mathsf{L}^{(S)} = \pi(\mathsf{L}^{(S')}) \Rightarrow S = \pi(S')$, and (2) $S = \pi(S'), \mathsf{A} = \pi(\mathsf{A}') \Rightarrow \mathsf{L}^{(S)} = \pi(\mathsf{L}^{(S')})$.

To explain a bit, labeling trick assigns a label vector to each node/edge in graph A, which constitutes the labeling tensor $\mathsf{L}^{(S)}$. By concatenating A and $\mathsf{L}^{(S)}$, we get the adjacency tensor $\mathsf{A}^{(S)}$ of the new labeled graph. By definition we can assign labels to both nodes and edges. For simplicity, here we only consider node labels, i.e., we let off-diagonal components $\mathsf{L}^{(S)}_{i,j,:}$ be all zero.

The labeling tensor $\mathsf{L}^{(S)}$ should satisfy two conditions in Definition 10.6. The first condition requires the target nodes S to have *distinct labels* from those of the rest nodes, so that S is distinguishable from others. This is because if a permutation π preserving node labels exists between nodes of A and A', then S and S' must have distinct labels to guarantee S' is mapped to S by π. The second condition requires the labeling function to be *permutation equivariant*, i.e., when (S, A) and (S', A') are isomorphic under π, the corresponding nodes $i \in S, j \in S', i = \pi(j)$ must always have the same label. In other words, the labeling should be consistent across different S.

For example, the zero-one labeling is a valid labeling trick by always giving label 1 to nodes in S and 0 otherwise, which is both consistent and S-discriminating. However, an all-one labeling is not a valid labeling trick, because it cannot distinguish the target set S.

Now we introduce the main theorem of labeling trick showing that with a valid labeling trick, a node-most-expressive GNN can learn structural link representations by aggregating its node representations learned from the **labeled** graph.

Theorem 10.4. *Given a node-most-expressive* GNN *and an injective set aggregation function* AGG*, for any* $S, \mathsf{A}, S', \mathsf{A}'$*,* $\text{GNN}(S, \mathsf{A}^{(S)}) = \text{GNN}(S', \mathsf{A}'^{(S')}) \Leftrightarrow (S, \mathsf{A}) \simeq (S', \mathsf{A}')$*, where* $\text{GNN}(S, \mathsf{A}^{(S)}) := \text{AGG}(\{\text{GNN}(i, \mathsf{A}^{(S)}) | i \in S\})$.

The proof of the above theorem can be found in Appendix A of (Zhang et al, 2020c). Theorem 10.4 implies that $\text{AGG}(\{\text{GNN}(i, \mathsf{A}^{(S)}) | i \in S\})$ is a structural representation for (S, A). Remember that directly aggregating structural node representations learned from the original graph A does not lead to structural link representations. Theorem 10.4 shows that aggregating over the structural node representations learned from the adjacency tensor $\mathsf{A}^{(S)}$ of the **labeled graph**, somewhat surprisingly, results in a structural representation for S.

The significance of Theorem 10.4 is that it closes the gap between GNN's node representation nature and link prediction's link representation requirement, which solves the open question raised in (Srinivasan and Ribeiro, 2020b) questioning node-based GNN methods' ability of performing link prediction. Although directly aggregating pairwise node representations learned by GNNs does not lead to structural link representations, combining GNNs with a labeling trick enables learning structural link representations.

It can be easily proved that the zero-one labeling, DRNL and Distance Encoding (DE) (Li et al, 2020e) are all valid labeling tricks. This explains subgraph-based methods' superior empirical performance than node-based methods (Zhang and Chen, 2018b; Zhang et al, 2020c).

10.5 Future Directions

In this section, we introduce several important future directions for link prediction: accelerating subgraph-based methods, designing more powerful labeling tricks, and understanding when to use one-hot features.

10.5.1 Accelerating Subgraph-Based Methods

One important future direction is to accelerate subgraph-based methods. Although subgraph-based methods show superior performance than node-based methods both empirically and theoretically, they also suffer from a huge computation complexity,

which prevent them from being deployed in modern recommendation systems. How to accelerate subgraph-based methods is thus an important problem to study.

The extra computation complexity of subgraph-based methods comes from their node labeling step. The reason is that for every link (i, j) to predict, we need to relabel the graph according to (i, j). The same node v will be labeled differently depending on which one is the target link, and will be given a different node representation by the GNN when it appears in different links' labeled graphs. This is different from node-based methods, where we do not relabel the graph and each node only has a single representation.

In other words, for node-based methods, we only need to apply the GNN to the whole graph once to compute a representation for each node, while subgraph-based methods need to repeatedly apply the GNN to differently labeled subgraphs each corresponding to a different link. Thus, when computing link representations, subgraph-based methods require re-applying the GNN for each target link. For a graph with n nodes and m links to predict, node-based methods only need to apply a GNN $\mathcal{O}(n)$ times to get a representation for each node (and then use some simple aggregation function to get link representations), while subgraph-based methods need to apply a GNN $\mathcal{O}(m)$ times for all links. When $m \gg n$, subgraph-based methods have much worse time complexity than node-based methods, which is the price for learning more expressive link representations.

Is it possible to accelerate subgraph-based methods? One possible way is to simplify the enclosing subgraph extraction process and simplify the GNN architecture. For example, we may adopt sampling or random walk when extracting the enclosing subgraphs which might largely reduce the subgraph sizes and avoid hub nodes. It is interesting to study such simplifications' influence on performance. Another possible way is to use distributed and parallel computing techniques. The enclosing subgraph extraction process and the GNN computation on a subgraph are completely independent of each other and are naturally parallelizable. Finally, using multi-stage ranking techniques could also help. Multi-stage ranking will first use some simple methods (such as traditional heuristics) to filter out most unlikely links, and use more powerful methods (such as SEAL) in the later stage to only rank the most promising links and output the final recommendations/predictions.

Either way, solving the scalability issue of subgraph-based methods can be a great contribution to the field. That means we can enjoy the superior link prediction performance of subgraph-based GNN methods without using much more computation resources, which is expected to extend GNNs to more application domains.

10.5.2 Designing More Powerful Labeling Tricks

Another direction is to design more powerful labeling tricks. Definition 10.6 gives a general definition of labeling trick. Although any labeling trick satisfying Definition 10.6 can enable a node-most-expressive GNN to learn structural link representations, the real-world performance of different labeling tricks can vary a lot due

to the limited expressive power and depths of practical GNNs. Also, some subtle differences in implementing a labeling trick can also result in large performance differences. For example, given the two target nodes x and y, when computing the distance $d(i,x)$ from a node i to x, DRNL will temporarily mask node y and all its edges, and when computing the distance $d(i,y)$, DRNL will temporarily mask node x and all its edges (Zhang and Chen, 2018b). The reason for this "masking trick" is that DRNL aims to use the pure distance between i and x without the influence of y. If we do not mask y, $d(i,x)$ will be upper bounded by $d(i,y)+d(x,y)$, which obscures the "true distance" between i and x and might hurt the node labels' ability to discriminate structurally-different nodes. As shown in Appendix H of (Zhang et al, 2020c), this masking trick can greatly improve the performance. It is thus interesting to study how to design a more powerful labeling trick (not necessarily based on shortest path distance like DRNL and DE). It should not only distinguish the target nodes, but also assign diverse but generalizable labels to nodes with different roles in the subgraph. A further theoretical analysis on the power of different labeling tricks is also needed.

10.5.3 Understanding When to Use One-Hot Features

Finally, one last important question remaining to be answered is when we should use the original node features and when we should use one-hot encoding features of node indices. Although using one-hot features makes it infeasible to learn structural link representations as discussed in Section 10.4.2, node-based methods using one-hot features show strong performance on dense networks (Zhang et al, 2020c), outperforming subgraph-based methods without using one-hot features by large margins. On the other hand, Kipf and Welling (2017b) show that GAE/VGAE with one-hot features give worse performance than using original features. Thus, it is interesting to study when to use one-hot features and when to use original features and theoretically understand their representation power differences on networks of different properties. Srinivasan and Ribeiro (2020b) provide a good analysis connecting positional node embeddings (such as DeepWalk) with structural node representations, showing that positional node embeddings can be seen as a sample while the structural node representation can be seen as a distribution. This can serve as a starting point to study the power of GNNs using one-hot encoding features, as GNNs using one-hot encoding features can be seen as combining positional node embeddings with message passing.

Editor's Notes: Link prediction is the problem of predicting the existence of a link between two nodes in a network. Hence the techniques are relevant to graph structure learning (chapter 19), which aims to discover useful graph structure, i.e. links, from data. Scalability property (chapter 6) and expressiveness power theory (chapter 8) play an important role in applying and designing link prediction methods. Link prediction also motivates several downstream tasks in various domains, such as predicting protein-protein and protein-drug interactions (chapter 25), drug development (chapter 24), recommender systems (chapter 19). Besides, predicting links in the complex network, including dynamic graphs (chapter 19), knowledge graphs (chapter 24) and heterogeneous graphs (chapter 26), are also the extension of link prediction tasks.