Revisiting Graph Neural Networks for Link Prediction

Muhan Zhang[1,2,*] **Pan Li**[3] **Yinglong Xia**[4] **Kai Wang**[4] **Long Jin**[4]
[1]Institute for Artificial Intelligence, Peking University
[2]Beijing Institute for General Artificial Intelligence
[3]Purdue University
[4]Facebook AI

Abstract

In this paper, we theoretically characterize graph neural network's representation power for high-order node set prediction problems (where a prediction is made over a set of more than 1 node). In particular, we focus on one most important second-order task—link prediction. There are two representative classes of GNN methods for link prediction: GAE and SEAL. GAE (Graph Autoencoder) first applies a GNN to the whole graph, and then aggregates the representations of the source and target nodes as their link representation. SEAL extracts a subgraph around the source and target nodes, **labels** the nodes in the subgraph, and then uses a GNN to learn a link representation from the labeled subgraph. At first glance, both GAE and SEAL use a GNN. However, their performance gap can be very large. On the recent Open Graph Benchmark datasets, SEAL achieved 3 first places out of 4 datasets, outperforming the best GAE method by up to 195% in Hits@100. In this paper, by studying this performance gap between GAE and SEAL, we first point out a key limitation of GAE caused by directly aggregating two node representations as a link representation. To address this limitation, we propose the *labeling trick*. Labeling trick unifies several recent successes to improve GNNs' representation power, such as SEAL, Distance Encoding, and Identity-aware GNN, into a single and most basic form. We prove that with labeling trick a sufficiently expressive GNN can learn the most expressive structural representations for node sets. Our work establishes a theoretical foundation for using GNNs for high-order node set prediction.

1 Introduction

Graph neural networks (GNNs) [1–10] have achieved great success in recent years. While graph neural networks have been well studied for single-node tasks (such as node classification) and whole-graph tasks (such as graph classification), using GNNs to predict over a set of several nodes is less studied and less understood. Among such high-order node set prediction problems, link prediction (predicting the link existence between a set of two nodes) is perhaps the most important one. Link prediction is to predict missing or future links between nodes in a network. It has wide applications, such as friend recommendation in social networks [11], movie recommendation in Netflix [12], protein interaction prediction [13], and drug response prediction [14]. In this paper, we use link prediction as a medium to study GNN's high-order node set representation ability. Note that although our examples and experiments are all around link prediction, our main technique and theorem apply to **all** high-order node set representation problems.

Traditional link prediction methods can be categorized into three classes: heuristic methods, embedding methods, and feature-based methods. Heuristic methods compute some heuristic graph

*Corresponding author: Muhan Zhang (muhan@pku.edu.cn)

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

structure features as the likelihood of links [15]. Examples include common neighbors, preferential attachment [16], and Katz index [17]. Embedding methods, including matrix factorization (MF) and Node2vec [18], learn free-parameter node embeddings in a transductive manner, thus do not generalize to unseen nodes and networks. Feature-based methods leverage explicit node features to predict links. By learning from graph structure and node features in a unified way, GNNs have shown superior link prediction performance than traditional methods in recent years [19–23].

There are two main types of GNN-based link prediction methods. One is Graph Autoencoder (**GAE**) [19], where a GNN is first applied to the entire network to compute a representation for each node. Then the representations of the source and target nodes are aggregated to predict the target link. Its variational version is called VGAE. The second type is **SEAL** [20, 23], where a local enclosing subgraph is extracted around each target link. Then the nodes in each enclosing subgraph are *labeled differently* according to their distances to the source and target nodes. Finally a GNN is applied to each enclosing subgraph to learn a link representation for link prediction. At first glance, GAE and SEAL look very similar—they both use a GNN to learn neighborhood structures and features around target links for link prediction. However, as we will see, SEAL has **fundamentally better** link representation learning ability than GAE. The key lies in SEAL's **node labeling** step.

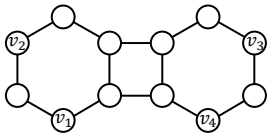

Figure 1: In this graph, nodes v_2 and v_3 are isomorphic; links (v_1, v_2) and (v_4, v_3) are isomorphic; link (v_1, v_2) and link (v_1, v_3) are **not** isomorphic. However, if we aggregate two node representations learned by a GNN as the link representation, we will give (v_1, v_2) and (v_1, v_3) the same prediction.

We first give a simple example to show when GAE fails. In Figure 1, v_2 and v_3 have symmetric positions in the graph—from their perspective points of view, they have exactly the **same h-hop neighborhood**, no matter how large h is. Thus, without node features, a GAE will learn the **same** representation for v_2 and v_3. However, when we want to predict which one of v_2 and v_3 is more likely to form a link with v_1, GAE will aggregate the representations of v_1 and v_2 as the link representation of (v_1, v_2), and aggregate the representations of v_1 and v_3 to represent (v_1, v_3), thus giving (v_1, v_2) and (v_1, v_3) the same representation and prediction. The failure to distinguish links with apparently different structural roles reflects one key limitation of GAE methods—they cannot capture the association between nodes: v_1 is much closer to v_2 than v_3 in the graph, and shares more common neighbors with v_2. By learning v_1 and v_2's representations independently, GAE fails to detect these.

One way to alleviate the above failure is to use one-hot encoding of node indices or random initialization as the input node features [24, 25]. With such node-discriminating features, v_2 and v_3 will have different node representations, thus (v_1, v_2) and (v_1, v_3) will also have different link representations after aggregation. However, using node-discriminating features loses GNN's inductive learning ability to map nodes and links with identical neighborhoods (such as nodes v_2 and v_3, and links (v_1, v_2) and (v_4, v_3)) to the same representation, which results in some loss of the generalization ability. Is there a way to make GNNs discriminate the cases like (v_1, v_2) and (v_1, v_3), while still maintaining their inductive learning ability, so that links (v_1, v_2) and (v_4, v_3) are mapped to the same representation (so are nodes v_2 and v_3)?

In this paper, we aim to address the above problem from a *structural link representation* learning point of view. Based on Theorem 1 of [26], the node set prediction problem on graphs ultimately requires finding a (most expressive) *structural representation* of node sets which gives two node sets the same representation if and only if they are *isomorphic* in the graph. For example, in Figure 1, link (v_1, v_2) and link (v_4, v_3) are isomorphic (i.e., symmetric in positions). A (most expressive) structural link representation should give any two isomorphic links the same representation while discriminating all non-isomorphic links (such as (v_1, v_2) and (v_1, v_3)). According to our discussion above, GAE fails to learn such a structural link representation. In this regard, we propose the *labeling trick* (Section 4.1), which gives a particular label to each node as its additional feature according to its structural relationship with the target node set to predict. We prove that combined with a labeling trick, a sufficiently expressive GNN can **learn structural link representations** by aggregating its pairwise node representations learned from the labeled graph. We further show that the techniques used in SEAL [20], Distance Encoding [23], and ID-GNN [27] are all specific implementations of the labeling trick, which explains their better link prediction performance than plain GAE.

Finally, we notice that isomorphic nodes/links might not be prevalent in real-world networks. Therefore, a more practical definition called *local isomorphism* is proposed. We argue that GNNs for

high-order node set prediction should focus on local isomorphism instead of (global) isomorphism, and extend our theory to local isomorphism. Experiments on four Open Graph Benchmark (OGB) link prediction datasets [28] demonstrate the effectiveness of the proposed labeling trick.

2 Preliminaries

In this section, we introduce some important concepts that will be used in the discussion of the paper, including *permutation*, *set isomorphism* and *structural representation*.

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

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 $\mathbf{A}$ as $\pi(\mathbf{A})$, where $\pi(\mathbf{A})_{\pi(i),\pi(j),:} = \mathbf{A}_{i,j,:}$.

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

Definition 1. ***(Set isomorphism)*** *Given two n-node graphs $\mathcal{G} = (V, E, \mathbf{A})$, $\mathcal{G}' = (V', E', \mathbf{A}')$, and two node sets $S \subseteq V$, $S' \subseteq V'$, we say $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are isomorphic (denoted by $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$) if $\exists \pi \in \Pi_n$ such that $S = \pi(S')$ and $\mathbf{A} = \pi(\mathbf{A}')$.*

When $(V, \mathbf{A}) \simeq (V', \mathbf{A}')$, we say two graphs $\mathcal{G}$ and $\mathcal{G}'$ are *isomorphic* (abbreviated as $\mathbf{A} \simeq \mathbf{A}'$ because $V = \pi(V')$ for any π). Note that set isomorphism is **more strict** than graph isomorphism, because it not only requires graph isomorphism, but also requires the permutation maps a specific node set S to another node set S'. In practice, when $S \neq V$, we are often more concerned with the case of $\mathbf{A} = \mathbf{A}'$, where we are to find isomorphic node sets **in the same graph** (automorphism). For example, when $S = \{i\}, S' = \{j\}$ and $(i, \mathbf{A}) \simeq (j, \mathbf{A})$, we say nodes i and j are isomorphic in graph $\mathbf{A}$ (or they have symmetric positions/same structural role within the graph). An example is v_2 and v_3 in Figure 1.

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

Now we define (most expressive) structural representation of a node set, following [26, 23]. Basically, it assigns a unique representation to each equivalence class of isomorphic node sets.

Definition 2. *Given an invariant function $\Gamma(\cdot)$, $\Gamma(S, \mathbf{A})$ is a **most expressive structural representation** for $(S, \mathbf{A})$ if $\forall S, \mathbf{A}, S', \mathbf{A}'$, $\Gamma(S, \mathbf{A}) = \Gamma(S', \mathbf{A}') \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}')$.*

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

Definition 2 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 [29] and matrix factorization [30], where two isomorphic nodes can have different node embeddings [31]. GAE using node-discriminating features also learns positional node embeddings.

Why do we need structural representations? Formally speaking, Srinivasan and Ribeiro [26] prove that any joint prediction task over node sets only requires a structural representation of node sets. They show that for any graph prediction task, positional node embeddings carry no more information beyond that of structural representations. Intuitively speaking, it is because two isomorphic nodes in a network are perfectly symmetric and interchangeable with each other, and should be indistinguishable

from any perspective. Learning a structural node representation can guarantee that isomorphic nodes are always classified into the same class. Similarly, learning a structural link representation guarantees isomorphic links, such as (v_1, v_2) and (v_4, v_3) in Figure 1, are always predicted the same, which is not guaranteed by positional node embeddings.

3 GAE cannot learn structural link representations

In this section, we review GAE, and show one key limitation caused by directly aggregating node representations, which prevents GAE from learning structural link representations.

3.1 GAE for link prediction

Given a graph $\mathbf{A}$, GAE methods [19] first use a GNN to compute a node representation $\boldsymbol{z}_i$ for each node i, and then use $f(\boldsymbol{z}_i, \boldsymbol{z}_j)$ to predict link (i, j):

$$\hat{\boldsymbol{A}}_{i,j} = f(\boldsymbol{z}_i, \boldsymbol{z}_j), \text{ where } \boldsymbol{z}_i = \text{GNN}(i, \mathbf{A}), \boldsymbol{z}_j = \text{GNN}(j, \mathbf{A}).$$

Here $\hat{\boldsymbol{A}}_{i,j}$ is the predicted score for link (i, j). The model is trained to maximize the likelihood of reconstructing the true adjacency matrix. The original GAE uses a two-layer GCN [5] as the GNN, and let $f(\boldsymbol{z}_i, \boldsymbol{z}_j) := \sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$. In principle, we can replace GCN with any GNN, and replace $\sigma(\boldsymbol{z}_i^\top \boldsymbol{z}_j)$ with an MLP over any aggregation function over $\{\boldsymbol{z}_i, \boldsymbol{z}_j\}$. Besides inner product, other aggregation choices include bilinear product, concatenation, mean and Hadamard product. In the following, we will use GAE to denote a general class of GNN-based link prediction methods.

GAE uses a GNN to learn node representations and then aggregate pairwise node representations as link representations. Two natural questions to ask are: 1) Is the node representation learned by the GNN a *structural node representation*? 2) Is the link representation aggregated from two node representations a *structural link representation*? We answer them respectively in the following.

3.2 GNN and structural node representation

Practical GNNs [32] usually simulate the 1-dimensional Weisfeiler-Lehman (1-WL) test [33] to iteratively update each node's representation by aggregating its neighbors' representations. We use *1-WL-GNN* to denote a GNN with 1-WL discriminating power, such as GIN [34].

A 1-WL-GNN ensures that isomorphic nodes always have the same representation. But the opposite direction is not guaranteed. For example, a 1-WL-GNN always gives the same representation to all nodes in an r-regular graph. Despite this, 1-WL is known to **discriminate almost all non-isomorphic nodes** [35]. This indicates that a 1-WL-GNN can always give the same representation to isomorphic nodes, and can give different representations to **almost all** non-isomorphic nodes.

We also define a *node-most-expressive GNN*, which gives different representations to **all** non-isomorphic nodes.

Definition 3. *A* GNN *is* ***node-most-expressive*** *if* $\forall i, \mathbf{A}, j, \mathbf{A}'$, $\text{GNN}(i, \mathbf{A}) = \text{GNN}(j, \mathbf{A}') \Leftrightarrow (i, \mathbf{A}) \simeq (j, \mathbf{A}')$.

That is, node-most-expressive GNN learns *structural node representations*[2]. We define such a GNN because we want to answer: whether GAE, even equipped with a node-most-expressive GNN (so that GNN's node representation power is not a bottleneck), can learn structural link representations.

3.3 GAE cannot learn structural link representations

Suppose GAE is equipped with a node-most-expressive GNN which outputs structural node representations. Then the question becomes: does the aggregation of structural node representations of i and j result in a structural *link* representation of (i, j)? The answer is no, as shown in previous works [26, 37]. We have also illustrated it in the introduction: In Figure 1, we have two isomorphic nodes v_2 and v_3, thus v_2 and v_3 will have the same structural node representation. By aggregating

[2] Although a polynomial-time implementation is not known for node-most-expressive GNNs, many practical softwares can discriminate all non-isomorphic nodes quite efficiently [36], which provides a promising direction.

structural node representations, GAE will give (v_1, v_2) and (v_1, v_3) the same link representation and prediction. However, (v_1, v_2) and (v_1, v_3) are not isomorphic in the graph. This indicates:

Proposition 1. *Even with a node-most-expressive GNN, GAE **cannot** learn structural link representations.*

The root cause of this problem is that GAE computes two node representations independently of each other, without considering their relative positions and **associations**. Because of this, GAE even cannot learn to count the **common neighbors** between two nodes (1 for (v_1, v_2) and 0 for (v_1, v_3)), one most important link prediction heuristic [15]. This greatly hurts GAE's link prediction ability.

4 How to learn structural representations

In this section, we first define the general form of the *labeling trick*, and then use a specific example, zero-one labeling trick, to intuitively explain why labeling trick helps a GNN learn better link representations. Next, we present our main theorem which states that labeling trick enables a node-most-expressive GNN to learn structural link representations, thus formally characterizing labeling trick's power for high-order node set representation learning. Finally, we review SEAL and show it exactly uses one labeling trick. We also show two recent successful techniques, Distance Encoding [23] and ID-GNN [27], are also specific labeling tricks.

4.1 Labeling trick

Definition 4. ***(Labeling trick)*** *Given $(S, \mathbf{A})$, we stack a labeling tensor $\mathbf{L}^{(S)} \in \mathbb{R}^{n\times n\times d}$ in the third dimension of $\mathbf{A}$ to get a new $\mathbf{A}^{(S)} \in \mathbb{R}^{n\times n\times(k+d)}$, where $\mathbf{L}$ satisfies: $\forall S, \mathbf{A}, S', \mathbf{A}', \pi \in \Pi_n$,*
(1) $\mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}) \Rightarrow S = \pi(S')$, and
(2) $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') \Rightarrow \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')})$.

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

The labeling tensor $\mathbf{L}^{(S)}$ should satisfy two conditions in Definition 4. Condition (1) 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 (i.e., $\mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')})$) exists between nodes of $\mathbf{A}$ and $\mathbf{A}'$, then S and S' must have distinct labels to guarantee S' is mapped to S by π (i.e., $S = \pi(S')$). Condition (2) requires the labeling function to be *permutation equivariant*, i.e., when (S, A) and (S', A') are isomorphic under π (i.e., $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}')$), the corresponding nodes $i \in S, j \in S', i = \pi(j)$ must always have the same label (i.e., $\mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')})$). In other words, the labeling function should be *consistent* across different S.

Now we introduce a simplest labeling trick satisfying the two conditions in Definition 4, and use it to illustrate how labeling trick helps GNNs learn better link representations.

Definition 5. ***(Zero-one labeling trick)*** *Given a graph $\mathbf{A}$ and a set of nodes S to predict, we give it a diagonal labeling matrix $\mathbf{L}^{(S)} \in \mathbb{R}^{n\times n\times 1}$ such that $\mathbf{L}^{(S)}_{i,i,1} = 1$ if $i \in S$ otherwise 0.*

In other words, the zero-one labeling trick assigns label 1 to nodes in S, and label 0 to all nodes not in S. It is a valid labeling trick, because firstly, nodes in S get *distinct labels*, and secondly, the labeling function is *consistent* by always giving nodes in S a label 1. These node labels will be used as additional node features, which are fed to a GNN together with original node features if available.

Let's return to the example in Figure 1 to see how the zero-one labeling trick helps GNNs learn better link representations. This time, when we want to predict link (v_1, v_2), we will mark v_1, v_2 with a different label from those of the rest nodes, as shown by the different color in Figure 2 left. With the source and target nodes labeled, when the GNN is computing v_2's representation, it is also "aware" of the source node v_1, instead of the previous agnostic way that treats v_1 the same as other nodes. And when we predict link (v_1, v_3), we will again mark v_1, v_3 with a different label, as shown

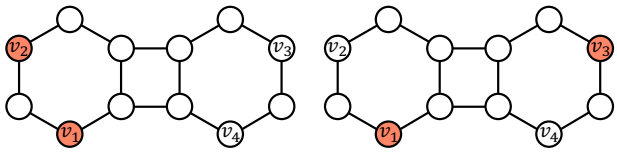

Figure 2: When we predict (v_1, v_2), we will label these two nodes differently from the rest, so that a GNN is aware of the target link when learning v_1 and v_2's representations. Similarly, when predicting (v_1, v_3), nodes v_1 and v_3 will be labeled differently. This way, the representation of v_2 in the left graph will be different from that of v_3 in the right graph, enabling GNNs to distinguish the non-isomorphic links (v_1, v_2) and (v_1, v_3).

in Figure 2 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 predict (v_1, v_2) and (v_1, v_3) differently. The key difference from GAE is that the node representations are no longer learned independently, but are *conditioned* on each other in order to capture two nodes' association.

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 GAE using node-discriminating features.

Now we introduce our main theorem showing that with a valid labeling trick, a node-most-expressive GNN can *learn structural link representations*.

Theorem 1. *Given a node-most-expressive* GNN *and an injective set aggregation function* AGG*, for any* $S, \mathbf{A}, S', \mathbf{A}'$*,* $\text{GNN}(S, \mathbf{A}^{(S)}) = \text{GNN}(S', \mathbf{A}'^{(S')}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}')$*, where* $\text{GNN}(S, \mathbf{A}^{(S)}) := \text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)}) | i \in S\})$.

Theorem 1 implies that $\text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)}) | i \in S\})$ is a structural representation for $(S, \mathbf{A})$. The significance of Theorem 1 is that it closes the gap between GNN's node representation nature and link prediction's link representation requirement. Remember that directly aggregating structural node representations learned from the original graph $\mathbf{A}$ does not lead to structural link representations. Theorem 1 shows that aggregating over the structural node representations learned from the labeled graph $\mathbf{A}^{(S)}$, somewhat surprisingly, results in a structural representation for S.

Theorem 1 also characterizes GNNs' node set representation learning ability, i.e., with labeling trick GNNs can indeed learn (most expressive) structural representations for node sets. This answers the open question raised in [26] questioning GNNs' link prediction ability: *are structural node representations in general–and GNNs in particular–fundamentally incapable of performing link (dyadic) and multi-ary (polyadic) prediction tasks?* With Theorem 1, we argue the answer is no. Although GNNs alone can at most learn structural node representations, GNNs combined with a labeling trick can learn structural link representations too.

Theorem 1 assumes a node-most-expressive GNN. To augment Theorem 1, we give the following theorem, which demonstrates labeling trick's power for 1-WL-GNNs.

Theorem 2. *In any non-attributed graphs with* n *nodes, if the degree of each node in the graph is between* 1 *and* $\mathcal{O}(\log^{\frac{1-\epsilon}{2h}} n)$ *for any constant* $\epsilon > 0$*, then there exists* $\omega(n^{2\epsilon})$ *many pairs of non-isomorphic links* $(u, w), (v, w)$ *such that an* h*-layer* 1-WL-GNN *gives* u, v *the same representation, while with labeling trick the* 1-WL-GNN *gives* u, v *different representations.*

Theorem 2 shows that labeling trick helps a 1-WL-GNN discriminate a large number of non-isomorphic links which are previously not distinguishable without a labeling trick.

4.2 SEAL uses a valid labeling trick

SEAL [20] is a state-of-the-art link prediction method based on GNNs. It first extracts an *enclosing subgraph* (h-hop subgraph) around each target link to predict.

Definition 6. ***(Enclosing subgraph)*** *Given* $(S, \mathbf{A})$*, the* h*-hop enclosing subgraph* $\mathbf{A}_{(S,h)}$ *of* S *is the subgraph induced from* $\mathbf{A}$ *by* $\cup_{j \in S}\{i \mid d(i, j) \leq h\}$*, where* $d(i, j)$ *is the shortest path distance between nodes* i *and* j*.*

Then, SEAL applies Double Radius Node Labeling (DRNL) to give an integer label to each node in the enclosing subgraph. DRNL assigns different labels to nodes with **different distances** to both the

source and target nodes. It works as follows: The source and target nodes are always labeled 1. Nodes farther away from the source and target nodes get larger labels (starting from 2). For example, nodes with distances 1 and 1 to the source and target nodes will get label 2, and nodes with distances 1 and 2 to the source and target nodes will get label 3. So on and so forth. Finally the labeled enclosing subgraph is fed to a GNN to learn the link representation and output the probability of link existence.

Theorem 3. *DRNL is a valid labeling trick.*

Theorem 3 is easily proved by noticing: across different subgraphs, 1) nodes with label 1 are always the source and target nodes, and 2) nodes with the same distances to the source and target nodes always have the same label. Thus, SEAL exactly uses a specific labeling trick to enhance its power.

SEAL only uses a subgraph $\mathbf{A}_{(S,h)}$ within h hops from the source and target nodes instead of using the whole graph. This is not a constraint but rather a practical consideration (just like GAE typically uses less than 3 message passing layers in practice), and its benefits will be discussed in detail in Section 5. When $h \to \infty$, the subgraph becomes the entire graph, and SEAL is able to *learn structural link representations* from the labeled (entire) graph.

Proposition 2. *When $h \to \infty$, SEAL can learn structural link representations with a node-most-expressive GNN.*

Remark 1 (Distance Encoding and DRNL) Recently, SEAL's distance-based node labeling scheme is generalized to *Distance Encoding*, or DE, in [23]. Basically, DRNL is equivalent to DE-2 using shortest path distance. Instead of encoding two distances into one integer label, DE directly concatenate the two distances as a size-2 label vector. For example, nodes with distances 1 and 2 to the source and target nodes will be directly labeled by a vector $[1, 2]$. **DE is also a valid labeling trick**, as it can also distinguish the target set S and is permutation equivariant. Nevertheless, there are some subtle differences in their implementations, which are discussed in Appendix D.

Remark 2 (ID-GNN and zero-one labeling trick) Concurrent to our work, You et al. [27] propose Identity-aware GNN (ID-GNN), which assigns a unique color to the "identity" nodes (the target node set to predict) and then performs message passing for them with a different set of parameters. If the unique colors are 1 and 0, ID-GNN's coloring scheme is equivalent to the zero-one labeling trick.

In summary, labeling trick unifies recent successful practices to improve the power of GNNs based on node labeling, including SEAL, DE, and ID-GNN, into a single framework. Moreover, labeling trick gives the most *basic* and *general* form for such techniques to reach theoretically the highest node set representation learning power. We discuss its computational complexity in Appendix F.

5 Local isomorphism: a more practical view of isomorphism

In the previous sections, we establish our theory based on assigning node sets the same representation if and only if they are *isomorphic* to each other in the graph. However, exact isomorphism is not very common. For example, Babai and Kucera [35] prove that at least $(n - \log n)$ nodes in almost all n-node graphs are *non-isomorphic* to each other. In practice, 1-WL-GNN also takes up to $\mathcal{O}(n)$ message passing layers to reach its maximum power for discriminating non-isomorphic nodes, making it very hard to really target on finding exactly isomorphic nodes/links.

Lemma 1. *Given a graph with n nodes, a 1-WL-GNN takes up to $\mathcal{O}(n)$ message passing layers to discriminate all the nodes that 1-WL can discriminate.*

In this regard, we propose a more practical concept, called *local isomorphism*.

Definition 7. ***(Local h-isomorphism)*** *$\forall S, \mathbf{A}, S', \mathbf{A}'$, we say $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are locally h-isomorphic to each other if $(S, \mathbf{A}_{(S,h)}) \simeq (S', \mathbf{A}'_{(S',h)})$.*

Local h-isomorphism only requires the h-hop enclosing subgraphs around S and S' are isomorphic, instead of the entire graphs. We argue that this is a more useful definition than isomorphism, because: 1) Exact isomorphism is rare in real-world graphs. 2) Algorithms targeting on exact isomorphism are more likely to overfit. Only assigning the same representations to exactly isomorphic nodes/links may fail to identify a large amount of nodes/links that are not isomorphic but have very similar neighborhoods. Instead, nodes/links *locally isomorphic* to each other may better indicate that they should have the same representation. With local h-isomorphism, *all our previous conclusions based*

on standard isomorphism still apply. For example, GAE (without node-discriminating features) still cannot discriminate locally h-non-isomorphic links. And a node-most-expressive GNN with labeling trick can learn the same representation for two links if and only if they are locally h-isomorphic:

Corollary 1. *Given a node-most-expressive* GNN *and an injective set aggregation function* AGG*, then for any* $S, \mathbf{A}, S', \mathbf{A}', h$, $\text{GNN}(S, \mathbf{A}^{(S)}_{(S,h)}) = \text{GNN}(S', \mathbf{A}'^{(S')}_{(S',h)}) \Leftrightarrow (S, \mathbf{A}_{(S,h)}) \simeq (S', \mathbf{A}'_{(S',h)})$.

Corollary 1 demonstrates labeling trick's power for local isomorphism discriminating, too. To switch to local h-isomorphism, all we need to do is to extract the h-hop enclosing subgraph around a link, and apply the labeling trick and GNN only to the extracted subgraph. This is exactly what SEAL did.

6 Related work

There is emerging interest in studying the representation power of graph neural networks recently. Xu et al. [34] and Morris et al. [38] first show that the discriminating power of GNNs performing neighbor aggregation is bounded by the 1-WL test. Many works have since been proposed to increase the power of GNNs by simulating higher-order WL tests [38–40]. However, most previous works focus on improving GNN's whole-graph representation power. Little work has been done to analyze GNN's node/link representation power. Srinivasan and Ribeiro [26] first formally studied the difference between structural representations of nodes and links. Although showing that structural node representations of GNNs cannot perform link prediction, their way to learn structural link representations is to give up GNNs and instead use Monte Carlo samples of node embeddings learned by network embedding methods. In this paper, we show that GNNs combined with a simple labeling trick can as well learn structural link representations, which reassures using GNNs for link prediction.

Many works have implicitly assumed that if a model can learn node representations well, then combining the pairwise node representations can also lead to good link representations [18, 19, 41]. However, we argue in this paper that simply aggregating node representations fails to discriminate a lot of non-isomorphic links, and with labeling trick the aggregation of structural node representations lead to structural link representations. Li et al. [23] proposed distance encoding (DE), which is shown to be a particular labeling trick. They proved that DE can improve 1-WL-GNNs' discriminating power, enabling them to differentiate almost all $(S, \mathbf{A})$ tuples sampled from r-regular graphs. Our paper's theoretical contribution is orthogonal to that of DE: 1) Our theory focuses on the gap between a GNN's node and node set representation power, instead of improving one particular GNN's discriminating power. We show even a GNN has maximum node representation power, it still fails to learn structural link representations unless combined with a labeling trick. 2) Our theory is not restricted to r-regular graphs, but applies to **any** graphs. 3) The proposed labeling trick is more general than DE. We show that labeling trick is not necessarily distance based—a valid labeling trick need only be permutation equivariant and S-discriminating. More discussion on DE's theory is given in Appendix E.

You et al. [21] also noticed that the structural node representations of GNNs cannot encode the associations and distances between nodes. To learn position-aware node embeddings, they propose P-GNN, which randomly chooses some anchor nodes and aggregates messages only from the anchor nodes. In P-GNN, nodes of similar distances to the anchor nodes, instead of nodes with similar neighborhoods, will have similar embeddings. Thus, P-GNN **cannot** learn structural node/link representations. P-GNN also cannot scale to large datasets. You et al. [27] recently proposed ID-GNN. As discussed in Remark 2, ID-GNN's node coloring is also a specific labeling trick.

7 Experiments

In this section, we empirically validate the effectiveness of labeling trick on Open Graph Benchmark (OGB) [28] datasets. We use all the four link prediction datasets in OGB: `ogbl-ppa`, `ogbl-collab`, `ogbl-ddi`, and `ogbl-citation2`. These datasets are open-sourced, large-scale (up to 2.9M nodes and 30.6M edges), adopt realistic train/validation/test splits, and have standard evaluation procedures, thus providing an ideal place to benchmark an algorithm's realistic link prediction power. The evaluation metrics include Hits@K and MRR. Hits@K counts the ratio of positive edges ranked at the K-th place or above against all the negative edges. MRR (Mean Reciprocal Rank) computes the reciprocal rank of the true target node against 1,000 negative candidates, averaged over all the true source nodes. Both metrics are higher the better. We include more details and statistics of

Table 1: Results for ogbl-ppa, ogbl-collab, ogbl-ddi and ogbl-citation2.

		ogbl-ppa Hits@100 (%)		ogbl-collab Hits@50 (%)		ogbl-ddi Hits@20 (%)		ogbl-citation2 MRR (%)	
Category	**Method**	Validation	**Test**	Validation	**Test**	Validation	**Test**	Validation	**Test**
Non-GNN	**CN**	28.23±0.00	27.6±0.00	60.36±0.00	61.37±0.00	9.47±0.00	17.73±0.00	51.19±0.00	51.47±0.00
	AA	32.68±0.00	32.45±0.00	63.49±0.00	64.17±0.00	9.66±0.00	18.61±0.00	51.67±0.00	51.89±0.00
	MLP	0.46±0.00	0.46±0.00	24.02±1.45	19.27±1.29	–	–	29.03±0.17	29.06±0.16
	Node2vec	22.53±0.88	22.26±0.88	57.03±0.52	48.88±0.54	32.92±1.21	23.26±2.09	61.24±0.11	61.41±0.11
	MF	32.28±4.28	32.29±0.94	48.96±0.29	38.86±0.29	33.70±2.64	13.68±4.75	51.81±4.36	51.86±4.43
Plain GAE	**GraphSAGE**	17.24±2.64	16.55±2.40	56.88±0.77	54.63±1.12	62.62±0.37	53.90±4.74	82.63±0.23	82.60±0.36
	GCN	18.45±1.40	18.67±1.32	52.63±1.15	47.14±1.45	55.50±2.08	37.07±5.07	84.79±0.23	84.74±0.21
	GCN+LRGA	25.75±2.82	26.12±2.35	60.88±0.59	52.21±0.72	**66.75**±0.58	**62.30**±9.12	66.48±1.61	66.49±1.59
Labeling Trick	**GCN+DE**	36.31±3.59	36.48±3.78	64.13±0.16	**64.44**±0.29	29.85±2.25	26.63±6.82	60.17±0.63	60.30±0.61
	GCN+DRNL	46.43±3.03	45.24±3.95	**64.51**±0.42	64.40±0.45	29.47±1.54	22.81±4.93	81.07±0.30	81.27±0.31
	SEAL	**51.25**±2.52	**48.80**±3.16	63.89±0.49	63.64±0.71	28.49±2.69	30.56±3.86	**87.57**±0.31	**87.67**±0.32

these datasets in Appendix G. Note that knowledge graph completion datasets are not used because knowledge graph link prediction is a different problem from the homogeneous graph link prediction problem studied in this paper. Their baselines and datasets are *not* interchangeable with each other.

Baselines. We use the following baselines for comparison. We use 5 non-GNN methods: CN (common neighbor), AA (Adamic-Adar), MLP, MF (matrix factorization) and Node2vec. Among then, CN and AA are two simple link prediction heuristics based on counting common neighbors, which are used for sanity checking. We use 3 plain GAE baselines: GraphSAGE [41], GCN [19], and GCN+LRGA [42]. These methods use the Hadamard product of pairwise node representations output by a GNN as link representations, without using a labeling trick. Finally, we compare 3 GNN methods using labeling tricks: GCN+DE [23], GCN+DRNL, and SEAL [20]. GCN+DE/GCN+DRNL enhance GCN with the DE/DRNL labeling trick. SEAL uses a GCN and the DRNL labeling trick, with an additional subgraph-level readout SortPooling [9]. More details are in Appendix H. Moreover, we test the zero-one labeling trick in our ablation experiments. Results can be found in Appendix I.

Results and discussion. We present the main results in Table 1. Firstly, we can see that GAE methods without labeling trick do not always outperform non-GNN methods. For example, on ogbl-ppa and ogbl-collab, simple heuristics CN and AA outperform plain GAE methods by large margins. This suggests that GAE methods cannot learn simple heuristics like common neighbors which require capturing the associations between nodes, matching our analysis in Section 3.3. In contrast, when GNNs are enhanced by labeling trick, they are able to beat heuristics. With labeling trick, GNN methods achieve new state-of-the-art performance on 3 out of 4 datasets. In particular, we observe that SEAL outperforms GAE and positional embedding methods, sometimes by surprisingly large margins. For example, in the challenging ogbl-ppa graph, SEAL achieves an Hits@100 of 48.80, which is **87%-195% higher** than GAE methods without using labeling trick. On ogbl-ppa, ogbl-collab and ogbl-citation2, labeling trick methods achieve new state-of-the-art results.

Despite obtaining the best results on three datasets, we observe that labeling trick methods do not perform well on ogbl-ddi. ogbl-ddi is considerably denser than the other graphs. It has 4,267 nodes and 1,334,889 edges, resulting in an average node degree of 500.5. In ogbl-ddi, labeling trick methods fall behind GAE methods using trainable node embeddings. One possible explanation is that ogbl-ddi is so dense that a practical GNN with *limited expressive power* is hard to inductively learn any meaningful structural patterns. In comparison, the transductive way of learning free-parameter node embeddings makes GAEs no longer focus on learning inductive structural patterns, but on optimizing node embeddings. The added parameters also greatly increase GAEs' model capacity. An interesting future topic is to study how to improve labeling tricks' performance on dense graphs.

We present more ablation experiments in Appendix I to study the power of different labeling tricks, the effect of subgraph pooling, and the number of hops/layers, etc.

8 Conclusions

In this paper, we theoretically characterized GNNs' representation learning ability for high-order node sets, such as links. We first pointed out a key limitation of GAE caused by directly aggregating node representations of a GNN as node set representations. To address the problem, we proposed the labeling trick and theoretically proved its power. We also showed that several existing methods,

including SEAL, DE and ID-GNN, all essentially used some labeling trick, explaining their strong performance. Experiments on OGB datasets demonstrated labeling trick's effectiveness.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[3] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[4] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3837–3845, 2016.

[7] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2702–2711, 2016.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[9] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, pages 4438–4445, 2018.

[10] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[11] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3): 211–230, 2003.

[12] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[13] Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins: Structure, Function, and Bioinformatics*, 63(3):490–500, 2006.

[14] Zachary Stanfield, Mustafa Coşkun, and Mehmet Koyutürk. Drug response prediction as a link prediction problem. *Scientific reports*, 7(1):1–13, 2017.

[15] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[16] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[17] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[18] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[19] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[20] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[21] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

[22] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems*, pages 4868–4879, 2019.

[23] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding–design provably more powerful gnns for structural representation learning. *arXiv preprint arXiv:2009.00142*, 2020.

[24] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199*, 2019.

[25] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020.

[26] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJxzFySKwH`.

[27] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. *arXiv preprint arXiv:2101.10320*, 2021.

[28] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[30] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[31] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

[32] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[33] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[35] László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE, 1979.

[36] Brendan D McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[37] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ByxxgCEYDS`.

[38] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[39] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167, 2019.

[40] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems*, pages 15894–15902, 2019.

[41] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[42] Omri Puny, Heli Ben-Hamu, and Yaron Lipman. From graph low-rank global attention to 2-fwl approximation. *arXiv preprint arXiv:2006.07846*, 2020.

[43] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The license can be found in their github.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The packages we used are all open-sourced.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] There are no personal information in the datasets.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 1

We restate Theorem 1: Given a node-most-expressive GNN and an injective set aggregation function AGG, for any $S, \mathbf{A}, S', \mathbf{A}'$, $\text{GNN}(S, \mathbf{A}^{(S)}) = \text{GNN}(S', \mathbf{A}'^{(S')}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}')$, where $\text{GNN}(S, \mathbf{A}^{(S)}) := \text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)}) | i \in S\})$.

Proof. We need to show $\text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)}) | i \in S\}) = \text{AGG}(\{\text{GNN}(i, \mathbf{A}'^{(S')} | i \in S'\})$ iff $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$.

To prove the first direction, we notice that with an injective AGG,

$$\begin{aligned}
&\text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)})) | i \in S\}) = \text{AGG}(\{\text{GNN}(i, \mathbf{A}'^{(S')})) | i \in S'\}) \\
\implies &\exists\, v_1 \in S, v_2 \in S', \text{ such that } \text{GNN}(v_1, \mathbf{A}^{(S)}) = \text{GNN}(v_2, \mathbf{A}'^{(S')}) && (1) \\
\implies &(v_1, \mathbf{A}^{(S)}) \simeq (v_2, \mathbf{A}'^{(S')}) \quad \text{(because GNN is node-most-expressive)} && (2) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } v_1 = \pi(v_2), \mathbf{A}^{(S)} = \pi(\mathbf{A}'^{(S')}). && (3)
\end{aligned}$$

Remember $\mathbf{A}^{(S)}$ is constructed by stacking $\mathbf{A}$ and $\mathbf{L}^{(S)}$ in the third dimension, where $\mathbf{L}^{(S)}$ is a tensor satisfying: $\forall \pi \in \Pi_n$, (1) $\mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}) \Rightarrow S = \pi(S')$, and (2) $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') \Rightarrow \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')})$. With $\mathbf{A}^{(S)} = \pi(\mathbf{A}'^{(S')})$, we have both

$$\mathbf{A} = \pi(\mathbf{A}'),\ \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}).$$

Because $\mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}) \Rightarrow S = \pi(S')$, continuing from Equation (3), we have

$$\begin{aligned}
&\text{AGG}(\{\text{GNN}(i, \mathbf{A}^{(S)}) | i \in S\}) = \text{AGG}(\{\text{GNN}(i, \mathbf{A}'^{(S')}) | i \in S'\}) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } \mathbf{A} = \pi(\mathbf{A}'),\ \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}) && (4) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } \mathbf{A} = \pi(\mathbf{A}'),\ S = \pi(S') && (5) \\
\implies &(S, \mathbf{A}) \simeq (S', \mathbf{A}'). && (6)
\end{aligned}$$

Now we prove the second direction. Because $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') \Rightarrow \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')})$, we have:

$$\begin{aligned}
&(S, \mathbf{A}) \simeq (S', \mathbf{A}') \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') && (7) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}'), \mathbf{L}^{(S)} = \pi(\mathbf{L}^{(S')}) && (8) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A}^{(S)} = \pi(\mathbf{A}'^{(S')}) && (9) \\
\implies &\exists\, \pi \in \Pi_n, \text{ such that } \forall v_1 \in S, v_2 \in S', v_1 = \pi(v_2), \text{ we have } \text{GNN}(v_1, \mathbf{A}^{(S)}) = \text{GNN}(v_2, \mathbf{A}'^{(S')}) && (10) \\
\implies &\text{AGG}(\{\text{GNN}(v_1, \mathbf{A}^{(S)}) | v_1 \in S\}) = \text{AGG}(\{\text{GNN}(v_2, \mathbf{A}'^{(S')}) | v_2 \in S'\}), && (11)
\end{aligned}$$

which concludes the proof.

□

B Proof of Theorem 2

We restate Theorem 2: In any non-attributed graphs with n nodes, if the degree of each node in the graph is between 1 and $\mathcal{O}(\log^{\frac{1-\epsilon}{2h}} n)$ for any constant $\epsilon > 0$, then there exists $\omega(n^{2\epsilon})$ many pairs of non-isomorphic links $(u, w), (v, w)$ such that an h-layer 1-WL-GNN gives u, v the same representation, while with labeling trick the 1-WL-GNN gives u, v different representations.

Proof. Our proof has two steps. First, we would like to show that there are $\omega(n^{\epsilon})$ nodes that are locally h-isomorphic (see Definition 7) to each other. Then, we prove that among these nodes, there are at least $\omega(n^{2\epsilon})$ pairs of nodes such that there exists another node constructing locally h non-isomorphic links with either of the two nodes in each node pair.

Step 1. Consider an arbitrary node v and denote the subgraph induced by the nodes that are at most h-hop away from v as $G_v^{(h)}$ (the h-hop enclosing subgraph of v). As each node is with degree $d = \mathcal{O}(\log^{\frac{1-\epsilon}{2h}} n)$, then the number of nodes in $G_v^{(h)}$, denoted by $|V(G_v^{(h)})|$, satisfies

$$|V(G_v^{(h)})| \leq \sum_{i=0}^{h} d^i = \mathcal{O}(d^h) = \mathcal{O}(\log^{\frac{1-\epsilon}{2}} n).$$

We set the max $K = \max_{v \in V} |V(G_v^{(h)})|$ and thus $K = \mathcal{O}(\log^{\frac{1-\epsilon}{2}} n)$.

Now we expand subgraphs $G_v^{(h)}$ to $\bar{G}_v^{(h)}$ by adding $K - |V(G_v^{(h)})|$ independent nodes for each node $v \in V$. Then, all $\bar{G}_v^{(h)}$ have the same number of nodes, which is K, though they may not be connected graphs.

Next, we consider the number of non-isomorphic graphs over K nodes. Actually, the number of non-isomorphic graph structures over K nodes is bounded by $2^{\binom{K}{2}} = \exp(\mathcal{O}(\log^{1-\epsilon} n)) = o(n^{1-\epsilon})$.

Therefore, due to the pigeonhole principle, there exist $n/o(n^{1-\epsilon}) = \omega(n^{\epsilon})$ many nodes v whose $\bar{G}_v^{(h)}$ are isomorphic to each other. Denote the set of these nodes as V_{iso}, which consist of nodes that are all locally h-isomorphic to each other. Next, we focus on looking for other nodes to form locally h-non-isomorphic links with nodes V_{iso}.

Step 2. Let us partition $V_{iso} = \cup_{i=1}^{q} V_i$ so that for all nodes in V_i, they share the same first-hop neighbor sets. Then, consider any pair of nodes u, v such that u, v are from different V_i's. Since u, v share identical h-hop neighborhood structures, an h-layer 1-WL-GNN will give them the same representation. Then, we may pick one u's first-hop neighbor w that is not v's first-hop neighbor. We know such w exists because of the definition of V_i. As w is u's first-hop neighbor and is not v's first-hop neighbor, (u, w) and (v, w) are not isomorphic. With labeling trick, the h-layer 1-WL-GNN will give u, v different representations immediately after the first message passing round due to w's distinct label. Therefore, we know such a $(u, w), (v, w)$ pair is exactly what we want.

Based on the partition V_{iso}, we know the number of such non-isomorphic link pairs (u, w) and (v, w) is at least:

$$Y \geq \prod_{i,j=1, i \neq j}^{q} |V_i||V_j| = \frac{1}{2}\left[(\sum_{i=1}^{q} |V_i|)^2 - \sum_{i=1}^{q} |V_i|^2\right]. \tag{12}$$

Because of the definitions of the partition, $\sum_{i=1}^{q} |V_i| = |V_{iso}| = \omega(n^{\epsilon})$ and the size of each V_i satisfies

$$1 \leq |V_i| \leq d_w = \mathcal{O}(\log^{\frac{1-\epsilon}{2h}} n),$$

where w is one of the common first-hop neighbors shared by all nodes in V_i and d_w is its degree.

By plugging in the range of $|V_i|$, Eq.12 leads to

$$Y \geq \frac{1}{2}(\omega(n^{2\epsilon}) - \omega(n^{\epsilon})\mathcal{O}(\log^{\frac{1-\epsilon}{2h}} n)) = \omega(n^{2\epsilon}),$$

which concludes the proof.

□

C Proof of Lemma 1

We restate Lemma 1: Given a graph with n nodes, a 1-WL-GNN takes up to $\mathcal{O}(n)$ message passing layers to discriminate all the nodes that 1-WL can discriminate.

Proof. We first note that after one message passing layer, 1-WL-GNN gives different embeddings to any two nodes that 1-WL gives different colors to after one iteration. So we only need to show how many iterations 1-WL takes to converge in any graph.

Note that if two nodes are given different colors by 1-WL at some iteration (they are discriminated by 1-WL), their colors are always different in any future iteration. And if at some iteration, all nodes' colors are the same as their colors in the last iteration, then 1-WL will stop (1-WL fails to discriminate any more nodes and has converged). Therefore, before termination, 1-WL will increase its total number of colors by at least 1 after every iteration. Because there are at most n different final colors given an n-node graph, 1-WL takes at most $n - 1 = \mathcal{O}(n)$ iterations before assigning all nodes different colors.

Now it suffices to show that there exists an n-node graph that 1-WL takes $\mathcal{O}(n)$ iterations to converge. Suppose there is a path of n nodes. Then by simple calculation, it takes $\lceil n/2 \rceil$ iterations for 1-WL to converge, which concludes the proof. □

D Comparisons between DRNL and DE

In this section, we discuss the relationships and differences between DRNL [20] and DE [23] (using shortest path distance). Although they are theoretically equivalent in the context of link prediction, there are some subtle differences that might result in significant performance differences.

Suppose x and y are the source and target nodes. **DRNL** (Double Radius Node Labeling) always assigns label 1 to x and y. Then, for any node i with $(d(i,x), d(i,y)) = (1,1)$, it assigns a 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. both the source and target nodes. The DRNL label $f_l(i)$ of a node i can be calculated by the following hashing function:

$$f_l(i) = 1 + \min(d_x, d_y) + (d/2)[(d/2) + (d\%2) - 1], \tag{13}$$

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. This hashing function allows fast closed-form computations of DRNL labels. For nodes with $d(i,x) = \infty$ or $d(i,y) = \infty$, DRNL assigns them a null label 0. Later, the one-hot encoding of these labels are fed to a GNN as the initial node features, or equivalently, we can feed the raw integer labels to an embedding layer first.

Instead of encoding $(d(i,x), d(i,y))$ into a single integer label, **DE** (distance encoding) directly uses the vector $[d(i,x), d(i,y)]$ as a size-2 label for node i. Then, these size-2 labels will be transformed to two-hot encoding vectors to be used as the input node features to GNN. Equivalently, we can also input the size-2 labels to an embedding layer and use the sum-pooled embedding as the initial node features.

These two ways of encoding $(d(i,x), d(i,y))$ theoretically have the same expressive power. However, DRNL and DE have some subtle differences in their implementations. The **first difference** is that DE sets a maximum distance d_{max} (a small integer such as 3) for each $d(i,x)$ or $d(i,y)$, i.e., if $d(i,x) \geq d_{\text{max}}$, DE will let $d(i,x) = d_{\text{max}}$. This potentially can avoid some overfitting by reducing the number of possible DE labels as claimed in the original paper [23].

The **second difference** is that when computing the distance $d(i,x)$, 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. The reason for this "masking trick" is because 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 we will show in Appendix I, this masking trick has a great influence on the performance, which explains DE's inferior performance than DRNL in our experiments.

As we will show in Table 1, DRNL has significantly better performance than DE on some datasets. To study what is the root cause for these in-principle equivalent methods's different practical performance, we propose **DE**$^+$, which adopts DRNL's masking trick in DE. We also try to not set a maximum distance in DE$^+$. This way, there are no more differences in terms of the expressive power between DE$^+$ and DRNL. And we indeed observed that DE$^+$ is able to catch up with DRNL in those datasets where DE does not perform well, as we will show in Appendix I.2.

E More discussion on the differences between DE's theory and ours

Inspired by the empirical success of SEAL [20], Li et al. [23] proposed distance encoding (DE). It generalizes SEAL's distance-based node labeling (DRNL) for link prediction to arbitrary node set prediction, and theoretically studies how the distance information improves 1-WL-GNN's discriminating power. The main theorem in [23] (Theorem 3.3) proves that under mild conditions, a 1-WL-GNN combined with DE can discriminate any $(S, \boldsymbol{A}), (S', \boldsymbol{A}')$ pair sampled uniformly from all r-regular graphs, with high probability. This is a significant result, as 1-WL-GNN's discriminating power is bounded by 1-WL, which fails to discriminate any nodes or node sets from r-regular graphs. DE's theory shows that with DE we can break the limit of 1-WL and 1-WL-GNN on this major class of graphs where without DE they always fail.

Despite the success, DE's theory also has several limitations. Firstly, its analysis leverages one particular random graph model, namely random regular graphs, thus does not show DE's power for arbitrary graphs. Secondly, DE's theory does not answer whether a GNN combined with DE can learn structural representations, which are the core for joint node set prediction tasks such as link prediction according to [26]. Thirdly, the definition of DE relies on distance. It is unknown whether other node labeling tricks (including those do not rely on distance) are also useful.

Our theory partly addresses these limitations and is orthogonal to DE's theory, as: 1) We give a more general definition than DE, namely labeling trick. Labeling trick is not restricted to be distance-based—it only needs to be permutation equivariant and target-node-set-discriminating. In fact, we show even a simple zero-one labeling trick is still very powerful. 2) We show with a sufficiently expressive GNN, labeling trick enables learning structural representations of node sets, answering the open question in [26] which DE's theory fails to address. 3) We show labeling trick's usefulness for arbitrary graphs, instead of only regular graphs.

Nevertheless, we are uncertain whether DE's power for regular graphs can transfer to any valid labeling trick, including those not based on distance. Thus, we leave an open question here for future research: whether an arbitrary labeling trick enables a 1-WL-GNN to discriminate any $(S, \boldsymbol{A}), (S', \boldsymbol{A}')$ pair sampled uniformly from all r-regular graphs, with high probability? Our guess is that the answer is yes for $|S| = 1$ and $|S| = 2$. This is because, with an injective message passing layer, we can propagate the unique labels of S to other nodes, thus "recovering" the distance information through iterative message passing. We leave a rigorous proof or disproof to future work.

F Computational complexity of labeling trick

Despite the power, labeling trick introduces extra computational complexity. The reason is that for every link (v_i, v_j) to predict, we need to relabel the graph according to (v_i, v_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 GAE, where we do not relabel the graph and each node only has a single representation. For a graph with n nodes and m links to predict, GAE only needs to apply a GNN $\mathcal{O}(n)$ times to get a representation for each node, while SEAL (and other GNNs using labeling trick) need to apply a GNN $\mathcal{O}(m)$ times for all links. When $m \gg n$, SEAL has worse time complexity than GAE, which is the price for learning better link representations.

G More details about the datasets

We compare the link prediction performance of different baselines on `ogbl-ppa`, `ogbl-collab`, `ogbl-ddi`, and `ogbl-citation2`. Among them, `ogbl-ppa` is a protein-protein association graph where the task is to predict biologically meaningful associations between proteins. `ogbl-collab` is

an author collaboration graph, where the task is to predict future collaborations. ogbl-ddi is a drug-drug interaction network, where each edge represents an interaction between drugs which indicates the joint effect of taking the two drugs together is considerably different from their independent effects. ogbl-citation2 is a paper citation network, where the task is to predict missing citations. We present the statistics of these OGB datasets in Table 2. More information about these datasets can be found in [28].

Table 2: Statistics and evaluation metrics of OGB link prediction datasets.

Dataset	#Nodes	#Edges	Avg. node deg.	Density	Split ratio	Metric
ogbl-ppa	576,289	30,326,273	73.7	0.018%	70/20/10	Hits@100
ogbl-collab	235,868	1,285,465	8.2	0.0046%	92/4/4	Hits@50
ogbl-ddi	4,267	1,334,889	500.5	14.67%	80/10/10	Hits@20
ogbl-citation2	2,927,963	30,561,187	20.7	0.00036%	98/1/1	MRR

We choose OGB datasets for benchmarking our methods because these datasets adopt realistic train/validation/test splitting methods, such as by resource cost in laboratory (ogbl-ppa), by time (ogbl-collab and ogbl-citation2), and by drug target in the body (ogbl-ddi). They are also large-scale (up to 2.9M nodes and 30.6M edges), open-sourced, and have standard evaluation metrics. OGB has an official leaderboard[3], too, providing a place to fairly compare different methods' link prediction performance.

H More details about the baselines

We include baselines achieving top places on the OGB leaderboard. All the baselines have their open-sourced code and paper available from the leaderboard. We adopt the numbers published on the leaderboard if available, otherwise we run the method ourselves using the open-sourced code. Note that there are other potentially strong baselines that we have to omit here, because they cannot easily scale to OGB datasets. For example, we have contacted the authors of P-GNN [21], and confirmed that P-GNN is not likely to scale to OGB datasets due to the computation of all-pairs shortest path distances.

All the compared methods are in the following. We briefly describe how each method obtains its final node representations.

- **MLP**: Node features are directly used as the node representations without considering graph structure.
- **Node2vec** [29, 18]: The node representations are the concatenation of node features and Node2vec embeddings.
- **MF** (Matrix Factorization): Use free-parameter node embeddings trained end-to-end as the node representations.
- **GraphSAGE** [41]: A GAE method with GraphSAGE as the GNN.
- **GCN** [19]: A GAE method with GCN as the GNN.
- **LRGA** [42]: A GAE method with LRGA-module-enhanced GCN.
- **GCN+DE**: Apply GCN to the DE [23] labeled graphs.
- **GCN+DRNL**: Apply GCN to the DRNL [20] labeled graphs.
- **SEAL** [20]: The same as GCN+DRNL with an additional subgraph-level readout. Note that we reimplemented SEAL in this paper with a greatly improved efficiency and flexibility than the original implementation[4]. The code will be released in the future.

Except SEAL, all models use the Hadamard product between pairwise node representations as the link representations. The link representations are fed to an MLP for final prediction. All the GAE methods' GNNs have 3 message passing layers with 256 hidden dimensions, with a tuned dropout ratio in $\{0, 0.5\}$. All the labeling trick methods (GCN+DE, GCN+DRNL and SEAL) extract 1-hop enclosing subgraphs. The GCNs in GCN+DRNL and GCN+DE also use 3 message passing layers with 256 hidden dimensions for consistency. The GNN in SEAL follows the DGCNN in the original paper,

[3] https://ogb.stanford.edu/docs/leader_linkprop/

[4] https://github.com/muhanzhang/SEAL

which has 3 GCN layers with 32 hidden dimensions each, plus a SortPooling layer [9] and several 1D convolution layers after the GCN layers to readout the subgraph. The use of a subgraph-level readout instead of only reading out two nodes is not an issue for SEAL, because 1) the two center nodes' information is still included in the output of the subgraph-level readout, and 2) the inclusion of additional neighborhood node representations may help learn better neighborhood features than only reading out two center nodes. As we will show in Appendix I.3, a subgraph-level readout sometimes improves the performance.

The `ogbl-ddi` graph contains no node features, so MLP is omitted, and the GAE methods here use free-parameter node embeddings as the GNN input node features and train them together with the GNN parameters. For labeling trick methods, the node labels are input to an embedding layer and then concatenated with the node features (if any) as the GNN input. Note that the original SEAL can also include pretrained node embeddings as additional features. But according to [26], node embeddings bring no additional value given structural representations. This is also consistent with our observation and the experimental results of [20], where including node embeddings gives no better results. Thus, we give up node embeddings in SEAL.

For the baseline GCN+LRGA, its default hyperparameters result in out of GPU memory on `ogbl-citation2`, even we use an NVIDIA V100 GPU with 32GB memory. Thus, we have to reduce its hidden dimension to 16 and matrix rank to 10. It is possible that it can achieve better performance with a larger hidden dimension and larger matrix rank using a GPU with a larger memory.

We implemented the labeling trick methods (GCN+DE, GCN+DRNL and SEAL) using the PyTorch Geometric [43] package. For all datasets, labeling trick methods only used a fixed 1% to 10% of all the available training edges as the positive training links, and sampled an equal number of negative training links randomly. Labeling trick methods showed excellent performance even without using the full training data, which indicates its strong inductive learning ability. Due to using different labeled subgraphs for different links, labeling trick methods generally take longer running time than GAE methods. On the largest `ogbl-citation2` graph, SEAL takes about 7 hours to finishing its training of 10 epochs, and takes another 28 hours to evaluate the validation and test MRR each. For `ogbl-ppa`, SEAL takes about 20 hours to train for 20 epochs and takes about 4 hours for evaluation. The other two datasets are finished within hours. We will release all our code for reproducing the experimental results.

I Ablation study

In this section, we conduct several ablation experiments to more thoroughly study the effect of different components around labeling trick on the final link prediction performance.

I.1 How powerful is the zero-one labeling trick?

Firstly, we aim to understand how powerful the proposed zero-one labeling (Definition 5) is. Although zero-one labeling is a also valid labeling trick that theoretically enables a node-most-expressive GNN to learn structural representations, in practice our GNNs may not be expressive enough. Then how will the zero-one labeling trick perform compared to those more sophisticated ones such as DE and DRNL? We conduct experiments on `ogbl-collab` and `ogbl-citation2` to answer this question. In Table 3, we compare GCN (1-hop) + all-zero labeling (not a valid labeling trick), GCN (1-hop) + zero-one labeling trick, and GCN (1-hop) + DRNL. All methods use the same 3 GCN layers with 256 hidden dimensions, 1-hop enclosing subgraphs, and Hadamard product of the source and target nodes as the link representations. All the remaining settings follow those of GCN+DRNL.

From Table 3, we can see that GCN+zero-one labeling trick indeed has better performance than GCN without labeling trick, which aligns with our theoretical results that even a simple zero-one labeling is also a valid labeling trick that enables learning structural representations. Nevertheless, the zero-one labeling trick is indeed less powerful than DRNL, as shown by the performance gaps especially on the `ogbl-citation2` dataset. We are then interested in figuring out what could cause such large performance differences between two (both valid) labeling tricks, because as Theorem 1 shows, any valid labeling trick can enable a node-most-expressive GNN to learn structural link representations.

Table 3: Ablation study on the power of the zero-one labeling trick.

	ogbl-collab Hits@50 (%)		ogbl-citation2 MRR (%)	
Method	Validation	**Test**	Validation	**Test**
GCN (1-hop) + all-zero labeling	24.35±1.28	25.92±1.47	36.97±0.56	36.98±0.57
GCN (1-hop) + zero-one labeling trick	44.45±1.39	44.79±1.26	38.73±0.86	38.78±0.88
GCN (1-hop) + DRNL	64.51±0.42	64.40±0.45	81.07±0.30	81.27±0.31
GIN (1-hop) + zero-one labeling trick	60.31±0.81	59.48±1.17	78.32±1.07	78.50±1.08

We suspect that the insufficient expressive power of GCN is the cause. Therefore, we change GCN to Graph Isomorphism Network (GIN) [34]. By replacing the linear feature transformations in GCN with MLPs, GIN is one of the most expressive GNNs based on message passing. The results are shown in the last column of Table 3. As we can see, GIN (1-hop) + zero-one labeling trick has much better performance than GCN (1-hop) + zero-one labeling trick, and is almost catching up with GCN (1-hop) + DRNL. The results very well align with our theory—as long as we have a sufficiently expressive GNN, even a simple zero-one labeling trick can be very powerful in terms of enabling learning structural representations. Nevertheless, in practice when we only have less powerful GNNs, we should better choose those more sophisticated labeling tricks such as DE and DRNL for better link prediction performance.

I.2 DE vs. DE$^+$ vs. DRNL

In Appendix D, we have discussed the differences of the implementations of DE and DRNL. That is, although DE and DRNL are equivalent in theory, there are two differences in their implementations: 1) DE sets a maximum distance d_{max} (by default 3) while DRNL does not, and 2) DRNL masks the source node when computing the distances to the target node and vice versa, while DE does not. To study whether it is these implementation differences between DE and DRNL that result in the large performance differences in Table 1, we propose **DE$^+$** which no longer sets a maximum distance in DE and additionally does the masking trick like DRNL. We compare DE, DE$^+$, and DRNL on `ogbl-ppa` and `ogbl-citation2` (where DE shows significantly lower performance than DRNL in Table 1). All of them use GCN as the GNN with the same hyperparameters. The results are shown in Table 4.

Table 4: Comparison of DE, DE$^+$ and DRNL.

	ogbl-ppa Hits@100 (%)		ogbl-citation2 MRR (%)	
Method	Validation	**Test**	Validation	**Test**
GCN+DE ($d_{\text{max}} = 3$)	36.31±3.59	36.48±3.78	60.17±0.63	60.30±0.61
CCN+DE$^+$ ($d_{\text{max}} = 3$)	47.17±1.84	45.70±3.46	74.75±1.18	75.00±1.20
CCN+DE$^+$ ($d_{\text{max}} = \infty$)	45.81±3.53	43.88±5.18	79.37±4.50	78.85±0.17
GCN+DRNL	46.43±3.03	45.24±3.95	81.07±0.30	81.27±0.31

We can observe that DE$^+$ outperforms DE by large margins. This indicates that the masking trick used in DRNL is very important. Intuitively, temporarily masking the target node y when computing distances to the source node x can give more diverse node labels. Without the masking, $d(i, x)$ will be upper bounded by $d(i, y) + d(x, y)$. Because the distance between x and y can be small in positive links, without the masking $d(i, x)$ will be restricted to small numbers, which hurts their ability to detect subtle differences between nodes' relative positions within the subgraph. Nevertheless, the benefit of the masking trick is not observed in smaller datasets such as `ogbl-collab` (Table 1).

We can also find that DE$^+$ without setting a maximum distance has very close performance to DRNL, which aligns with our discussion in Appendix D. By removing the maximum distance restriction, DE$^+$ essentially becomes DRNL. However, there are still small performance differences, possibly because DRNL has a larger embedding table than DE$^+$ (DRNL's maximum label is larger) which results in a slightly larger model capacity. Nevertheless, this can be alleviated by doubling the embedding dimension of DE$^+$. In summary, we can conclude that the masking trick used in DRNL is crucial to the performance on some datasets. Compared to DE, DE$^+$ and DRNL show better practical performance. Studying more powerful labeling tricks is also an important future direction.

I.3 Is a subgraph-level readout useful?

In Table 1, we observe that SEAL is generally better than GCN+DRNL. SEAL also uses GCN and the DRNL labeling trick, so the main difference is the subgraph-level readout in SEAL. That is, instead of only reading out the two center nodes' representations as the link representation, SEAL performs a readout over all the nodes in the enclosing subgraph. Here we study this effect further by testing whether a subgraph-level sum-pooling readout is also useful. We replace the Hadamard product of two center node representations in GCN+DRNL with the sum over all node representations within the enclosing subgraph. The results are shown in Table 5.

Table 5: Ablation study on subgraph-level readout.

	`ogbl-collab` Hits@50 (%)		`ogbl-citation2` MRR (%)	
Method	Validation	**Test**	Validation	**Test**
GCN+DRNL	64.51±0.42	64.40±0.45	81.07±0.30	81.27±0.31
GCN+DRNL (sum-pooling)	64.64±0.24	63.26±0.35	84.98±0.23	85.20±0.26
SEAL	63.89±0.49	63.64±0.71	87.57±0.31	87.67±0.32

As we can see, using sum-pooling has a similar effect to the SortPooling in SEAL, i.e., it greatly improves the performance on `ogbl-citation2`, while slightly reduces the performance on `ogbl-collab`. This means, using a subgraph-level readout can sometimes be very helpful. Although according to Theorem 1 we only need to aggregate the representations of the center two nodes (source and target nodes) as the link representation, in practice, because our GNNs only have limited expressive power, reading out all nodes within the enclosing subgraph could help GNNs learn better subgraph-level features thus better detecting the target link's local h-isomorphism class. Such subgraph representations can be more expressive than only the two center nodes' representations, especially when the number of message passing layers is small so that the center nodes have not gained enough information from the whole subgraph.

I.4 Is it helpful to make number of layers larger than number of hops?

In all labeling trick methods, we have used a fixed enclosing subgraph hop number $h = 1$, and a fixed number of message passing layers $l = 3$. Using a number of message passing layers larger than the number of hops is different from the practice of previous work. For example, in GAE, we always select $h = l$ hops of neighbors if we decide to use l message passing layers. So is it really helpful to use $l > h$? Intuitively, using $l > h$ layers can make GNNs more sufficiently absorb the entire enclosing subgraph information and learn better link representations. Theoretically, as we have shown in Lemma 1, to reach the maximum representation power of 1-WL-GNN, we need to use $\mathcal{O}(n)$ number of message passing layers, where n is the number of nodes in the enclosing subgraph. Thus, using $l > h$ can enhance GNN's representation power and learn more expressive link representations.

Table 6: Ablation study on subgraph-level readout.

	`ogbl-ppa` Hits@100 (%)		`ogbl-collab` Hits@50 (%)		`ogbl-citation2` MRR (%)	
Method	Validation	**Test**	Validation	**Test**	Validation	**Test**
GCN+DRNL ($l = 3$)	46.43±3.03	45.24±3.95	64.51±0.42	64.40±0.45	81.07±0.30	81.27±0.31
GCN+DRNL ($l = 1$)	31.59±2.79	33.57±3.06	64.38±0.13	63.95±0.42	77.77±0.42	78.02±0.44

To validate the above , we conduct experiments on GCN+DRNL by using $l = 1$ message passing layers (and still $h = 1$). The results are shown in Table 6. As we can observe, using $l = 1$ results in lower performance than using $l = 3$ in all three datasets. On `ogbl-collab`, this effect is very small. However, on `ogbl-ppa` and `ogbl-citation2`, the performance gaps are significant. These results demonstrate the usefulness of using more message passing layers than hops.

Nevertheless, we are unsure whether it is still helpful to make $l > h$ when we use a large h, such as $h = 2$ or $h = 3$. We cannot generally verify this because increasing h will exponentially increase our subgraph sizes. And considering the huge computation cost on two relatively large datasets `ogbl-ppa` and `ogbl-citation2`, using $h = 1$ is currently the maximum h we can afford. We

thus only conduct experiments using different h's on the smallest `ogbl-collab` dataset. We have tried different combinations of (l, h) from $(1, 1)$ all the way up to $(4, 3)$, and the testing scores are consistently around 63 to 64. This seems to indicate increasing h or l is not helpful in this dataset. Nevertheless, `ogbl-collab` may not be representative enough to derive a general conclusion. For example, in the original SEAL paper [20], the authors found using $h = 2$ is helpful for many datasets. Thus, fully answering this question might need further investigations. But when $h = 1$, we can conclude that using $l > h$ is better.