Journal of Machine Learning Research 26 (2025) 1-44 Submitted 5/23; Revised 1/25; Published 1/25

Improving Graph Neural Networks on Multi-node Tasks with the Labeling Trick

Xiyuan Wang WANGXIYUAN@PKU.EDU.CN
Institute for Artificial Intelligence
Peking University
Beijing, China

Pan Li PANLI@GATECH.EDU
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, USA

Muhan Zhang* MUHAN@PKU.EDU.CN
Institute for Artificial Intelligence
Peking University
Beijing, China

Editor: Samy Bengio

Abstract

In this paper, we study using graph neural networks (GNNs) for *multi-node representation learning*, where a representation for a set of more than one node (such as a link) is to be learned. Existing GNNs are mainly designed to learn single-node representations. When used for multi-node representation learning, a common practice is to directly aggregate the single-node representations obtained by a GNN. In this paper, we show a fundamental limitation of such an approach, namely the inability to capture the dependence among multiple nodes in the node set. A straightforward solution is to distinguish target nodes from others. Formalizing this idea, we propose labeling trick, which first labels nodes in the graph according to their relationships with the target node set before applying a GNN and then aggregates node representations obtained in the labeled graph for multi-node representations. Besides node sets in graphs, we also extend labeling tricks to posets, subsets and hypergraphs. Experiments verify that the labeling trick technique can boost GNNs on various tasks, including undirected link prediction, directed link prediction, hyperedge prediction, and subgraph prediction. Our work explains the superior performance of previous node-labeling-based methods and establishes a theoretical foundation for using GNNs for multi-node representation learning.

Keywords: graph neural networks, multi-node representation, subgraph, link prediction

1. Introduction

Graph neural networks (GNNs) (Scarselli et al., 2009; Bruna et al., 2014; Duvenaud et al., 2015; Li et al., 2016; Kipf and Welling, 2017; Defferrard et al., 2016; Dai et al., 2016; Veličković et al., 2018; Zhang et al., 2018b; Ying et al., 2018) have achieved great successes in recent years. While GNNs have been well studied for single-node tasks (such as

* correspondence to Muhan Zhang

node classification) and whole-graph tasks (such as graph classification), using GNNs on tasks that involve multi-nodes is less studied and less understood. Among such *multi-node representation learning* problems, link prediction (predicting the link existence/class/value between a set of two nodes) is perhaps the most important one due to its wide applications in practice, such as friend recommendation in social networks (Adamic and Adar, 2003), movie recommendation in Netflix (Bennett et al., 2007), protein interaction prediction (Qi et al., 2006), drug response prediction (Stanfield et al., 2017), and knowledge graph completion (Nickel et al., 2016). Besides link prediction, other multi-node tasks, like subgraph classification and hyperedge prediction, are relatively new but have found applications in gene set analysis (Wang et al., 2020), user profiling (Alsentzer et al., 2020), drug interaction prediction (Srinivasan et al., 2021), temporal network modeling (Liu et al., 2022), group recommendation Amer-Yahia et al. (2009), etc. In this paper, we study the ability of GNNs to learn multi-node representations. As the link task is the simplest multi-node case, we mainly use link prediction in this paper to visualize and illustrate our method and theory. However, our theory and method apply generally to all multi-node representation learning problems such as subgraph (Alsentzer et al., 2020), hyperedge (Zhang et al., 2018a) and network motif (Liu et al., 2022) prediction tasks.

Starting from the link prediction task, we illustrate the deficiency of existing GNN models for multi-node representation learning which motivates our labeling trick. There are two main classes of GNN-based link prediction methods: Graph AutoEncoder (GAE) (Kipf and Welling, 2016) and SEAL (Zhang and Chen, 2018; Li et al., 2020). **GAE** and its variational version VGAE (Kipf and Welling, 2016) first apply a GNN to the entire graph to compute a representation for each node. The representations of the two end nodes of the link are then aggregated to predict the target link. On the contrary, SEAL assigns node labels according to their distances to the two end nodes before applying the GNN on the graph. SEAL often shows much better practical performance than GAE. The key lies in SEAL's node labeling step.

We first give a simple example to show when GAE fails. In Figure 1a, v_2 and v_3 have symmetric positions in the graph—from their respective views, they have the same h-hop neighborhood for any h. Thus, without node features, GAE will learn the same representation for v_2 and v_3. Therefore, when predicting 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 (v_1, v_2) and (v_1, v_3) that have different structural roles in the graph reflects one key limitation of GAE-type methods: by computing v_1 and v_2's representations independently of each other, GAE cannot capture the dependence between two end nodes of a link. For example, (v_1, v_2) has a much smaller shortest path distance than that of (v_1, v_3); and (v_1, v_2) has both nodes in the same hexagon, while (v_1, v_3) does not. We can also consider this case from another perspective. Common neighbor (CN) (Liben-Nowell and Kleinberg, 2007), one elementary heuristic feature for link prediction, counts the number of common neighbors between two nodes to measure their likelihood of forming a link. It is the foundation of many other successful heuristics such as Adamic-Adar (Adamic and Adar, 2003) and Resource Allocation (Zhou et al., 2009), which are also based on neighborhood overlap. However, GAE cannot capture such neighborhood-overlap-based features. As

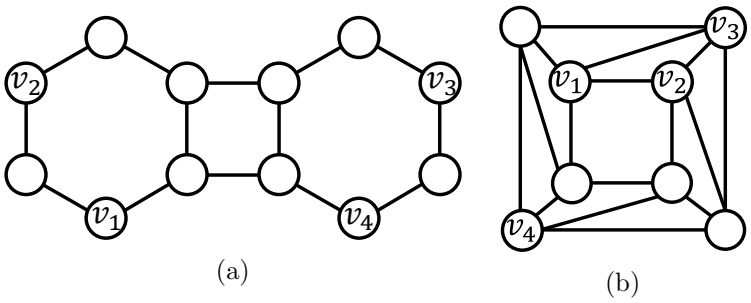

Figure 1: (a) In this graph, nodes v_2 and v_3 are in the same orbit; links (v_1, v_2) and (v_4, v_3) are isomorphic; link (v_1, v_2) and link (v_1, v_3) are **not** isomorphic. However, if we aggregate two node representations learned by a GNN as the link representation, we will give (v_1, v_2) and (v_1, v_3) the same prediction. (b) In this graph, nodes v_3 and v_4 are isomorphic. Aggregating the node embeddings within the subgraph, GNNs will produce equal embeddings for subgraphs (v_1, v_2, v_3) and (v_1, v_2, v_4), while the two subgraphs are not isomorphic. This problem was first observed by You et al. (2019), which was interpret as the failure of GNNs to capture node positions, and later became more formalized in (Srinivasan and Ribeiro, 2020).

shown in Figure 1a, there is 1 common neighbor between (v_1, v_2) and 0 between (v_1, v_3), but GAE always gives (v_1, v_2) and (v_1, v_3) the same representation. The failure to learn common neighbor demonstrates GAE's severe limitation for link prediction. The root cause still lies in that GAE computes node representations independently of each other, and when computing the representation of one end node, it is unaware of the other end node.

In fact, GAE represents a common practice of using GNNs to learn multi-node representations. That is, obtaining individual node representations through a GNN and then aggregating the representations of those target nodes as the multi-node representation. Similar failures caused by independence of node representation learning also happen in general multi-node representation learning problems. In the subgraph representation learning task, which is to learn representations for subgraphs inside a large graph (Alsentzer et al., 2020), representations aggregated from independently computed node representations will fail to differentiate nodes inside and outside the subgraph. Figure 1b (from Wang and Zhang (2022)) shows an example. Directly aggregating node embeddings produced by a GNN will lead to the same representation for subgraphs (v_1, v_2, v_3) and (v_1, v_2, v_4). However, the former subgraph forms a triangle while the latter one does not.

This paper solves the above type of failures from a *structural representation learning* point of view. We adopt and generalize the notion *most expressive structural representation* (Srinivasan and Ribeiro, 2020), which gives multi-node substructure the same representation if and only if they are *isomorphic* (a.k.a. symmetric, on the same orbit) in the graph. For example, link (v_1, v_2) and link (v_4, v_3) in Figure 1a are isomorphic, and a most expressive structural representation should give them the same representation. On the other hand, a most expressive structural representation will discriminate all non-isomorphic links (such as (v_1, v_2) and (v_1, v_3)). According to our discussion above, GAE-type methods that directly aggregate node representations cannot learn a most expressive structural representation. Then, how to learn a most expressive structural representation of node sets?

To answer this question, we revisit the other GNN-based link prediction framework, SEAL, and analyze how node labeling helps a GNN learn better node set representations. We find that two properties of the node labeling are crucial for its effectiveness: 1) target-node distinguishing, which ensures that target nodes receive labels that differentiate them from other nodes in the graph and 2) permutation equivariance. With these two properties, we define *set labeling trick*, which considers each multi-node substructure as a node set and unifies previous node labeling methods into a single and most general form. Theoretically, we prove that with set labeling trick, a sufficiently expressive GNN can learn most expressive structural representations of node sets (Theorem 12), which reassures GNN's node set prediction ability. It also closes the gap between the nature of GNNs to learn node representations and the need of multi-node representation learning in node-set-based inference tasks.

Set labeling trick is for multi-node structure of a node set and can be used on a wide range of tasks including link prediction and subgraph classification. However, to describe and unify even more tasks and methods, we propose three extensions of set labeling trick. One is *poset labeling trick*. In some tasks, target nodes may have intrinsic order relations in real-world problems. For example, in citation graphs, each link is from the citing article to the cited one. In such cases, describing multi-node substructures with node sets leads to loss of order information. This motivates us to add order information to the label and use poset instead to describe substructures. Another extension is *subset labeling trick*. It unifies labeling methods besides SEAL (Zhang and Chen, 2018), like ID-GNN (You et al., 2021) and NBFNet (Zhu et al., 2021). These works label only a subset of nodes each time. We formalize these methods and analyze the expressivity: when using GNNs without strong expressivity, subset labeling trick exhibits higher expressivity than labeling tricks in some cases. Last but not least, by converting hypergraph to bipartite graph, we straightforwardly extend labeling trick to hypergraph.

2. Preliminaries

In this section, we introduce some important concepts that will be used in the analysis of the paper, including *permutation*, *poset-graph isomorphism* and *most expressive structural representation*.

We consider a 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. In this paper, we let all graphs have a node set numbered from 1 to the total number of nodes in the graph. 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). The node/edge types can also be expressed in $\mathbf{A}$ using integers or one-hot encoding vectors for heterogeneous graphs. 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 $(i, j) \in E$, where it is possible $\boldsymbol{A}_{i,j} \neq \boldsymbol{A}_{j,i}$. We let $\boldsymbol{A}$ be the first slice of $\mathbf{A}$, i.e., $\boldsymbol{A} = \mathbf{A}_{:,:,1}$. Since $\mathbf{A}$ contains the complete information of a graph, we also directly denote the graph by $\mathbf{A}$.

2.1 Permutation

The same graph can index nodes in different orders, and these different indices can be connected with permutation.

Definition 1 *A **permutation** π is a bijective mapping from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$. All $n!$ possible π's constitute the permutation group Π_n.*

Depending on the context, permutation π can mean assigning a new index $\pi(i)$ to node $i \in V$, or mapping node i to node $\pi(i)$ of another graph. Slightly extending the notation, we let the permutation of a set/sequence denote permuting each element in the set/sequence. For example, permutation π maps a set of nodes $S \subseteq V$ to $\pi(S) = \{\pi(i)|i \in S\}$ and maps a set of node pairs $S' \subseteq V \times V$ to $\pi(S') = \{\pi((i,j))|(i,j) \in S'\} = \{(\pi(i), \pi(j))|(i,j) \in S'\}$. The permutation of a graph's tensor $\mathbf{A}$, denoted as $\pi(\mathbf{A})$, can also be defined: $\pi(\mathbf{A})_{\pi(i),\pi(j)} = \mathbf{A}_{i,j}$, where original i-th node and j-th node will have new index $\pi(i), \pi(j)$ while keeping the features of the pair $\mathbf{A}_{i,j}$.

Permutation is closely related to *graph isomorphism*, whether two graphs describe the same structure. Intuitively, as nodes in graphs have no order, no matter what permutation is applied to a graph, the transformed graph should be isomorphic to the original graph. Similarly, if one graph can be transformed into another under some permutation, the two graphs should also be isomorphic. Formally speaking,

Definition 2 *Two graphs $\mathbf{A} \in \mathbb{R}^{n \times n \times d}, \mathbf{A'} \in \mathbb{R}^{n' \times n' \times d'}$ are **isomorphic** iff there exists $\pi \in \Pi_n$, $\pi(\mathbf{A}) = \mathbf{A'}$.*

In whole graph classification tasks, models should give isomorphic graphs the same prediction as they describe the same structure, and differentiate non-isomorphic graphs.

2.2 Poset-Graph Isomorphism

To describe a substructure defined by a subset of nodes with internal relation, like a directed edge, we introduce poset. A poset is a set with a partial order. Partial order is a reflexive, antisymmetric, and transitive homogeneous relation on the set (Davey and Priestley, 2002).

Definition 3 *A **poset** S is a tuple $(U, \leq_S)$, where U is a set, and $\leq_S \subseteq U \times U$ is a relation on U. Let $u \leq_S v$ denote $(u, v) \in \leq_S$. $\leq_S$ fulfills the following conditions.*

1. *Reflexivity. $\forall v \in U, v \leq_S v$.*
2. *Antisymmetry. $\forall u, v \in U$, if $u \leq_S v$ and $v \leq_S u$, then $u = v$.*
3. *Transitivity. $\forall u, v, w \in U$, if $u \leq_S v$ and $v \leq_S w$, then $u \leq_S w$.*

The permutation operation on partial order relation and poset is defined as follows.

$$\pi(\leq_S) = \pi(\{(u, v) \mid (u, v) \in \leq_S\}) = \{(\pi(u), \pi(v)) \mid (u, v) \in \leq_S\},$$
$$\pi(S) = \pi((U, \leq_S)) = (\pi(U), \pi(\leq_S)).$$

To describe when two posets derive the same substructure, we define *poset-graph isomorphism*, which generalizes graph isomorphism to arbitrary node posets in a graph.

Definition 4 ***(Poset-graph isomorphism)*** *Given two graphs $\mathcal{G} = (V, E, \mathbf{A})$, $\mathcal{G}' = (V', E', \mathbf{A}')$, and two node posets $S = (U, \leq_S), U \subseteq V$, $S' = (U', \leq_{S'}), U' \subseteq V'$, we say substructures $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are isomorphic (denoted by $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$) iff $\exists \pi \in \Pi_n, S = \pi(S')$ and $\mathbf{A} = \pi(\mathbf{A}')$.*

A set is a particular case of poset, where the partial order only contains reflexive relations $u \leq_S u, u \in U$. It can describe substructures without order, like undirected edges and subgraphs. Abusing the notation of poset, we sometimes also use S to denote a set and omit the trivial partial order relation. Then, *set-graph isomorphism* is defined as follows.

Definition 5 ***(Set-graph isomorphism)*** *Given two 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 substructures $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are isomorphic (denoted by $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$) iff $\exists \pi \in \Pi_n, S = \pi(S')$ and $\mathbf{A} = \pi(\mathbf{A}')$.*

Note that both set- and poset-graph isomorphism are **more strict** than graph isomorphism. They not only need a permutation which maps one graph to the other but also require the permutation to map a specific node poset S to S'.

In practice, when the target node poset does not contain all nodes in the graph, we are often more concerned with the case of $\mathbf{A} = \mathbf{A}'$, where isomorphic node posets are defined **in the same graph**. 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 in graph $\mathbf{A}$). An example is v_2 and v_3 in Figure 1a. Similarly, edge and subgraph isomorphism can also be defined as the isomorphism of their node posets.

2.3 Structural Representations

Graph models should produce the same prediction for isomorphic substructures. We define permutation invariance and equivariance to formalize this property. 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 $\forall \pi \in \Pi_n$, $\pi(f(S, \mathbf{A})) = f(\pi(S), \pi(\mathbf{A}))$, where for example $f(S, \mathbf{A})$ can be a tensor $L \in \mathbb{R}^{n \times n \times d}$, $\pi(f(S, \mathbf{A}))_{\pi(i)\pi(j)} = f(S, \mathbf{A})_{ij}$. Permutation invariance/equivariance ensures that representations learned by a GNN are invariant to node indexing, a fundamental design principle of GNNs.

Now we define the *most expressive structural representation* of a substructure $(S, \mathbf{A})$, following (Srinivasan and Ribeiro, 2020; Li et al., 2020). It assigns a unique representation to each equivalence class of isomorphic substructures.

Definition 6 *Given an invariant function $\Gamma(\cdot)$ mapping node subsets in graphs to a latent space, $\Gamma(\cdot)$ is a* ***most expressive structural representation****, 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 directly 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. For a graph $\mathbf{A}$, we call $\Gamma(\mathbf{A}) = \Gamma(\emptyset, \mathbf{A})$ a *structural graph representation*, $\Gamma(i, \mathbf{A})$ a *structural node representation* for node i, and call $\Gamma(\{i, j\}, \mathbf{A})$ a *structural link representation* for link (i, j). For a general node poset S, we call $\Gamma(S, \mathbf{A})$ a *structural multi-node representation* for S.

Definition 6 requires that the structural representations of two substructures are the same if and only if the two substructures are isomorphic. That is, isomorphic substructures always have the **same** structural representation, while non-isomorphic substructures always have **different** structural representations. Due to the permutation invariance requirement, models should not distinguish isomorphic substructures. This implies that structural representations can discriminate all substructures that any invariant model can differentiate, and structural representations reach the highest expressivity.

3. The Limitation of Directly Aggregating Node Representations

In this section, taking GAE for link prediction as an example, we show the critical limitation of directly aggregating node representations as a multi-node representation.

3.1 GAE for Multi-Node Representation

GAE (Kipf and Welling, 2016) is a kind of link prediction model with GNN. Given a graph $\mathbf{A}$, GAE first uses a GNN to compute a node representation $\boldsymbol{z}_i$ for each node i, and then use the inner product of $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ to predict link $\{i, j\}$:

$$\hat{\boldsymbol{A}}_{i,j} = \text{sigmoid}(\boldsymbol{z}_i^\top \boldsymbol{z}_j), \text{ where } \boldsymbol{z}_i = \text{GCN}(i, \mathbf{A}), \boldsymbol{z}_j = \text{GCN}(j, \mathbf{A}).$$

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 (Kipf and Welling, 2017). In principle, we can replace GCN with any GNN, use any aggregation function over the set of target node embeddings including mean, sum, and max other than inner product, and substitute sigmoid with an MLP. Then, GAE can be used for multi-node tasks. It aggregates target node embeddings produced by the GNN:

$$\boldsymbol{z}_S = \text{MLP}(\text{AGG}(\{\boldsymbol{z}_i \mid i \in S\})) \text{ where } \boldsymbol{z}_i = \text{GNN}(i, \mathbf{A}),$$

where AGG is an aggregation function, which takes a multiset instead of set by default. We will use GAE to denote this general class of GNN-based multi-node representation learning methods in the following. Two natural questions are: 1) Is the node representation learned by the GNN a *structural node representation*? 2) Is the multi-node representation aggregated from a set of node representations a *structural representation for the node set*? We answer them respectively in the following.

3.2 GNN and Structural Node Representation

Practical GNNs (Gilmer et al., 2017) usually simulate the 1-dimensional Weisfeiler-Lehman (1-WL) test (Weisfeiler and Lehman, 1968) 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 (Xu et al., 2019).

A 1-WL-GNN ensures that isomorphic nodes always have the same representation. However, the opposite direction is not guaranteed. For example, a 1-WL-GNN gives the same representation to all nodes in an r-regular graph, in which non-isomorphic nodes exist. Despite this, 1-WL is known to discriminate almost all non-isomorphic nodes as the number of

nodes grows to infinity (Babai and Kucera, 1979), which indicates that a 1-WL-GNN can give different representations to almost all non-isomorphic nodes in large real-world graphs.

To study GNN's maximum expressivity, we define a *node-most-expressive (NME) GNN*, which gives different representations to **all** non-isomorphic nodes.

Definition 7 *A GNN is* ***node-most-expressive (NME)*** *if there exists a parameterization of the GNN that* $\forall i, \mathbf{A}, j, \mathbf{A}'$, $GNN(i, \mathbf{A}) = GNN(j, \mathbf{A}') \Leftrightarrow (i, \mathbf{A}) \simeq (j, \mathbf{A}')$.

NME GNN learns *structural node representations*. We define such a GNN because our primary focus is on multi-node representation. By ignoring the limitations of single-node expressivity, NME GNN simplifies our analysis. Although a polynomial-time implementation is not known for NME GNNs, many practical software tools can discriminate between all non-isomorphic nodes efficiently (McKay and Piperno, 2014), providing a promising direction.

3.3 GAE Cannot Learn Structural Multi-Node Representations

Suppose GAE is equipped with an NME GNN producing structural node representations. Then the question becomes: does the aggregation of structural node representations of the target nodes result in a structural representation of the target node set? The answer is no. We have already illustrated this problem in the introduction: In Figure 1a, we have two isomorphic nodes v_2 and v_3, and thus v_2 and v_3 will have the same structural node representation. By aggregating structural node representations, GAE will give (v_1, v_2) and (v_1, v_3) the same link representation. However, (v_1, v_2) and (v_1, v_3) are not isomorphic in the graph. Figure 1b gives another example on the multi-node case involving more than two nodes. Previous works have similar examples (Srinivasan and Ribeiro, 2020; Zhang and Chen, 2020). All these results indicate that:

Proposition 8 *(Srinivasan and Ribeiro (2020)) GAE* ***cannot*** *learn structural multi-node representations no matter how expressive node representations a GNN can learn.*

The root cause of this problem is that GNN computes node representations independently without being aware of the other nodes in the target node set S. Thus, even though GNN learns the most expressive single-node representations, there is never a guarantee that their aggregation is a structural representation of a node set. In other words, the multi-node representation learning problem is **not breakable** into multiple **independent** single-node representation learning problems. We need to consider the **dependency** between the target nodes when computing their single-node representations.

4. Labeling Trick for Set

Starting from a common case in real-world applications, we first describe the multi-node substructure defined by a node set (instead of a poset) in the graph and define *set labeling trick*. The majority of this part is included in our conference paper (Zhang et al., 2021a).

4.1 Definition of Set Labeling Trick

The set labeling trick is defined as follows.

Definition 9 ***(Set labeling trick)*** *For a graph* $\mathbf{A}$ *and a set* S *of nodes in the graph, we stack a labeling tensor* $\mathbf{L}(S, \mathbf{A}) \in \mathbb{R}^{n\times n\times k}$ *in the third dimension of* $\mathbf{A}$ *to get a new* $\mathbf{A}^{(S)} \in \mathbb{R}^{n\times n\times(k+d)}$. $\mathbf{L}$ *satisfies:* $\forall S, \mathbf{A}, S', \mathbf{A'}, \pi \in \Pi_n$,

1. *(target-nodes-distinguishing)* $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A'})) \Rightarrow S = \pi(S')$.
2. *(permutation equivariance)* $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A'}) \Rightarrow \mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A'}))$.

To explain a bit, labeling trick assigns a label vector to each node/edge in graph $\mathbf{A}$, which constitutes the labeling tensor $\mathbf{L}(S, \mathbf{A})$. By concatenating $\mathbf{A}$ and $\mathbf{L}(S, \mathbf{A})$, 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 **consider node labels only** by default for simplicity, i.e., we let the off-diagonal components $\mathbf{L}(S, \mathbf{A})_{i,j,:}, i \neq j$ be all zero.

The labeling tensor $\mathbf{L}(S, \mathbf{A})$ should satisfy two properties in Definition 9. Property 1 requires that if a permutation π preserving node labels (i.e., $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A'}))$) exists between nodes of $\mathbf{A}$ and $\mathbf{A'}$, then the nodes in S' must be mapped to nodes in S by π (i.e., $S = \pi(S')$). A sufficient condition for property 1 is to make the target nodes S have *distinct labels* from those of the rest nodes so that S is *distinguishable* from others. Property 2 requires that when $(S, \mathbf{A})$ and $(S', \mathbf{A'})$ 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., $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A'}))$). A sufficient condition for property 2 is to make the labeling function *permutation equivariant*, i.e., when the target $(S, \mathbf{A})$ changes to $(\pi(S), \pi(\mathbf{A}))$, the labeling tensor $\mathbf{L}(\pi(S), \pi(\mathbf{A}))$ should equivariantly change to $\pi(\mathbf{L}(S, \mathbf{A}))$.

4.2 How Labeling Trick Works

Obviously, labeling trick puts extra information into the graph, while the details remain unclear. To show some intuition on how labeling trick boosts graph neural networks, we introduce a simplest labeling trick satisfying the two properties in Definition 9.

Definition 10 ***(Zero-one labeling trick)*** *Given a graph* $\mathbf{A}$ *and a set of nodes* S *to predict, we give it a diagonal labeling matrix* $\mathbf{L}_{zo}(S, \mathbf{A}) \in \mathbb{R}^{n\times n\times 1}$ *such that*

$$\mathbf{L}_{zo}(S, \mathbf{A})_{i,i,1} = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}.$$

In other words, the zero-one labeling trick assigns 1 to nodes in S and labels 0 to all other nodes in the graph. It is a valid labeling trick because nodes in S get *distinct labels* from others, and the labeling function is *permutation equivariant* by always giving nodes in the target node set label 1. These node labels serve as additional node features fed to a GNN together with the original node features.

Let's return to the example in Figure 1a to see how the zero-one labeling trick helps GNNs learn better multi-node representations. This time, when we want to predict link (v_1, v_2), we will label v_1, v_2 differently from the rest nodes, as shown by the distinct colors in Figure 2 left. When computing v_2's representation, GNN is also "aware" of the source node v_1 with nodes v_1 and v_2 labeled, rather than treating v_1 the same as other nodes. Similarly, when predicting link (v_1, v_3), the model will again label v_1, v_3 differently from other nodes

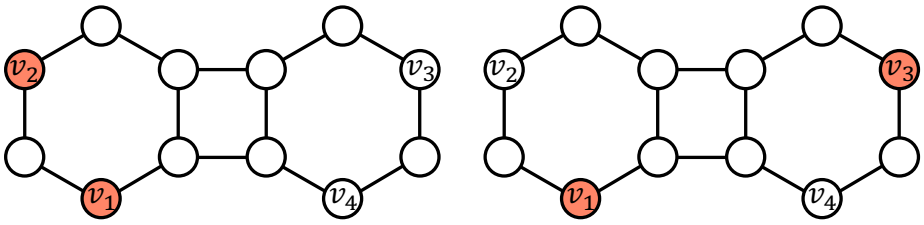

Figure 2: When predicting (v_1, v_2), we will label these two nodes differently from the rest so that a GNN is aware of the target link when learning v_1 and v_2's representations. Similarly, when predicting (v_1, v_3), nodes v_1 and v_3 will be labeled differently. This way, the representation of v_2 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).

as shown 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 the model can predict (v_1, v_2) and (v_1, v_3) differently. The key difference of model with labeling trick from GAE is that the node representations are no longer computed independently, but are *conditioned* on each other in order to capture the dependence between nodes.

4.3 Expressivity of GNN with Labeling Trick

We include all proofs in the appendix.

Labeling trick first bridges the gap between whole-graph representation (the focus of graph level GNNs) and node set representations.

Proposition 11 *(Zhang et al., 2021a) For any node set S in graph $\mathbf{A}$ and S' in graph $\mathbf{A}'$, given a set labeling trick, $(S, \mathbf{A}) \simeq (S', \mathbf{A}') \Leftrightarrow \mathbf{A}^{(S)} \simeq \mathbf{A}'^{(S')}$.*

The problem of graph-level tasks on a labeled graph ($\mathbf{A}^{(S)}$ as defined in Definition 9) is equivalent to that of multi-node tasks. However, the complexity of these graph-level GNNs are usually larger than GNNs encoding nodes. We further want to connect node set representations with node representations. Now we introduce our main theorem showing that with a valid labeling trick, an NME GNN can *learn structural representations of node sets.*

Theorem 12 *(Zhang et al., 2021a) Given an NME GNN and an injective set aggregation function AGG, for any $S, \mathbf{A}, S', \mathbf{A}'$, $GNN(S, \mathbf{A}^{(S)}) = GNN(S', \mathbf{A}'^{(S')}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}')$, where $GNN(S, \mathbf{A}^{(S)}) := AGG(\{GNN(i, \mathbf{A}^{(S)}) \mid i \in S\})$.*

Remember that directly aggregating the structural node representations learned from the original graph $\mathbf{A}$ does not lead to structural representations of node sets (Section 3.3). In contrast, Theorem 12 shows that aggregating the structural node representations learned from the **labeled** graph $\mathbf{A}^{(S)}$, somewhat surprisingly, results in a structural representation for $(S, \mathbf{A})$.

The significance of Theorem 12 is that it closes the gap between the nature of GNNs for single-node representations and the requirement of multi-node representations for node

set prediction problems. Although GNNs alone have severe limitations for multi-node representations, GNNs + labeling trick can learn structural representations of node sets by aggregating structural node representations obtained in the labeled graph.

Theorem 12 assumes an NME GNN. To augment Theorem 12, we give the following theorems, which demonstrate the power of labeling trick for 1-WL-GNNs on link prediction.

Theorem 13 *(Zhang et al., 2021a) Given an h-layer 1-WL-GNN, in any non-attributed graph with n nodes, if the degree of each node in the graph is between* 1 *and* $\big((1-\epsilon)\log n\big)^{1/(2h+2)}$ *for any constant* $\epsilon \in \Big(\frac{\log\log n}{(2h+2)\log n}, 1\Big)$*, there exists* $\omega(n^{2\epsilon})$ *pairs of non-isomorphic links* $(u,w),(v,w)$ *such that 1-WL-GNN gives* u, v *the same representation, while with 1-WL-GNN + zero-one labeling trick gives* u, v *different representations.*

Theorem 13 shows that in any non-attributed graph there exists a large number ($\omega(n^{2\epsilon})$) of link pairs (like the examples (v_1, v_2) and (v_1, v_3) in Figure 1a) which are not distinguishable by 1-WL-GNNs alone but distinguishable by 1-WL-GNNs + labeling trick. This means, labeling trick can boost the expressive power of 1-WL-GNNs on link prediction tasks.

How labeling trick boosts link prediction can also be shown from another perspective: 1-WL-GNN + zero-one labeling trick can **learn various link prediction heuristics** while vanilla 1-WL-GNN cannot.

Proposition 14 *Given a link prediction heuristic of the following form,*

$$f\big(\{\sum_{v\in N(i)} g_2(deg(v,\mathbf{A})), \sum_{v\in N(j)} g_2(deg(v,\mathbf{A}))\}, \sum_{v\in N(i)\bigcap N(j)} g_1(deg(v,\mathbf{A}))\big),$$

where $deg(v,\mathbf{A})$ *is the degree of node* v *in graph* $\mathbf{A}$*,* g_1, g_2 *are positive functions, and* f *is injective w.r.t. the second input with the first input fixed. There exists a 1-WL-GNN + zero-one labeling trick implementing this heuristic. In contrast, 1-WL-GNN cannot implement it.*

The heuristic defined in the above proposition covers many widely-used and time-tested link prediction heuristics, such as common neighbors (CN) (Barabási and Albert, 1999), resource allocation(RA) (Zhou et al., 2009), and Adamic-Adar(AA) (Adamic and Adar, 2003). These important structural features for link prediction are not learnable by vanilla GNNs but can be learned if we augment 1-WL-GNNs with a simple zero-one labeling trick.

Labeling trick can also boost graph neural networks in subgraph tasks with more than two nodes. The following proposition.

Proposition 15 *(Wang and Zhang (2022)) Given an h-layer 1-WL-GNN, in any non-attributed graph with n nodes, if the degree of each node in the graph is between* 1 *and* $\big((1-\epsilon)\log n\big)^{1/(2h+2)}$ *for any constant* $\epsilon > 0$*, there exists* $w(2^n n^{2\epsilon-1})$ *pairs of non-isomorphic subgraphs such that that 1-WL-GNN produces the same representation, while 1-WL-GNN + labeling trick can distinguish them.*

Theorem 15 extends Theorem 13 to more than 2 nodes. It shows that an even larger number of node set pairs need labeling tricks to help 1-WL-GNNs differentiate them.

4.4 Complexity

Despite the expressive power, labeling trick may introduce extra computational complexity. The reason is that for every node set S to predict, we need to relabel the graph $\mathbf{A}$ according to S and compute a new set of node representations within the labeled graph. In contrast, GAE-type methods compute node representations only in the original graph.

Let m denote the number of edges, n denote the number of nodes, and q denote the number of target node sets to predict. As node labels are usually produced by some fast non-parametric method, we neglect the overhead for computing node labels. Then we compare the inference complexity of GAE and GNN with labeling trick. For small graphs, GAE-type methods can compute all node representations first and then predict multiple node sets at the same time, which saves a significant amount of time. In this case, GAE's time complexity is $O(m+n+q)$, while GNN with labeling trick takes up to $O(q(m+n))$ time. However, for large graphs that cannot fit into the GPU memory, extracting a neighborhood subgraph for each node set to predict has to be used for both GAE-type methods and labeling trick, resulting in similar computation cost $O(q(n_s+m_s))$, where n_s, m_s are the average number of nodes and edges in the segregated subgraphs. We also measures time and GPU memory consumption on link prediction task in Appendix D.

5. Labeling Trick for Poset

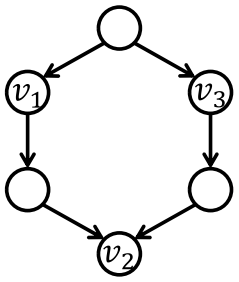

Figure 3: Set labeling with Graph Neural Networks (GNNs) fails to distinguish between non-isomorphic directed links, such as the edge from v_1 to v_2 versus the edge from v_2 to v_3, because it does not account for the order of nodes within the target node pairs.

The previous section describes multi-node substructures $(S, \mathbf{A})$ defined by node set S, which assumes that nodes in S have no order relation. However, the assumption may lose some critical information in real-world tasks. For example, the citing and cited articles should be differentiated in citation graphs. As shown in Figure 3, using set labeling trick cannot discriminate the link direction by giving the two directed links the same representation, yet the two directed links are obviously non-isomorphic. Therefore, introducing order relation into node set is necessary for substructures with internal relation. In this section, we use poset to define multi-node substructures and extend set labeling trick to *poset labeling trick*. Note that node order is only additionally introduced for S because the graph $\mathbf{A}$ already allows directed edges in our definition.

Definition 16 ***(Poset labeling trick)*** *Given a graph $\mathbf{A}$ and a poset S of nodes in it, we stack a labeling tensor $\mathbf{L}(S, \mathbf{A}) \in \mathbb{R}^{n\times n\times d}$ 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: for all poset S of nodes in graph $\mathbf{A}$, poset S' of nodes in graph $\mathbf{A}'$, and $\pi \in \Pi_n$,*

1. *(target-nodes-and-order-distinguishing)* $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A})) \Rightarrow S = \pi(S')$.
2. *(permutation equivariance)* $S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') \Rightarrow \mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A}'))$.

The definition of poset labeling trick is nearly the same as that of set labeling trick, except that we require permutation of poset and poset-graph isomorphism (Definition 3 and 4). Poset labeling trick still assigns a label vector to each node/edge in graph $\mathbf{A}$. The labels distinguish the substructure from other parts of the graph and keep permutation equivariance. As we will show, poset labeling trick enables maximum expressivity for poset learning. Below we first discuss how to design poset labeling tricks that satisfy the two above properties.

5.1 Poset Labeling Trick Design

To describe general partial order relations between nodes in a poset, we introduce *Hasse diagram*, a graph that uniquely determines the partial order relation.

Definition 17 *The Hasse diagram of a poset $S = (U, \leq_S)$, denoted as $\mathcal{H}_S$, is a directed graph (V_H, E_H), $V_H = U$, $E_H = \{(u, v) \mid v \neq u \text{ and } v \text{ covers } u\}$, where v covers u means that $u \leq_S v$ and there exists no $w \in U, w \notin \{u, v\}$, $u \leq_S w$ and $w \leq_S v$.*

Figure 4 shows some examples of Hasse diagram. The reason we use Hasse diagram to encode partial order relation is that we prove any poset labeling trick satisfying Definition 16 must give non-isomorphic nodes in a Hasse diagram different labels.

Proposition 18 *Let $\mathbf{L}$ be the labeling function of a poset labeling trick. If $\exists \pi \in \Pi_n, \mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A}'))$, then for all $v' \in S'$, $\pi(v')$ is in S, and $(\{v'\}, \mathcal{H}_{S'}) \simeq (\{\pi(v')\}, \mathcal{H}_S)$. Furthermore, in the same $\mathcal{H}_S$, non-isomorphic nodes must have different labels.*

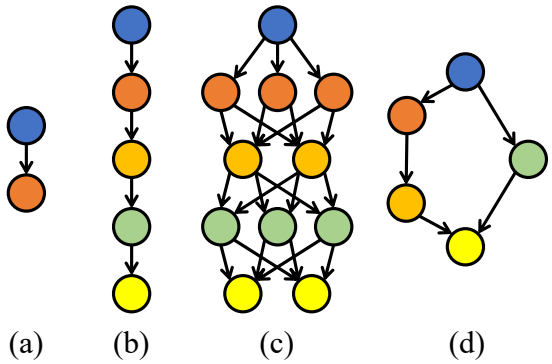

Figure 4: Different Hasse diagrams

Proposition 18 shows that a valid poset labeling trick should differentiate non-isomorphic nodes in a Hasse diagram. Theoretically, we can run an NME GNN on the Hasse diagram so that the node embeddings can serve the purpose. Such a poset labeling trick is defined as follows.

Definition 19 *Given an NME GNN, Hasse embedding labeling trick is*

$$\mathbf{L}(S, \mathbf{A})_{u,u,:} = \begin{cases} iso(u, \mathcal{H}_S) & \text{if } u \in S \\ 0 & \text{otherwise} \end{cases}$$

This labeling trick fulfills the two requirements in Definition 16. $iso(u, \mathcal{H}_S)$ denotes the isomorphism type (non-zero) of node u in Hasse diagram $\mathcal{H}_S$, where $iso(u_1, \mathcal{H}_{S_1}) = iso(u_2, \mathcal{H}_{S_2})$ iff $(u_1, \mathcal{H}_{S_1}) \simeq (u_2, \mathcal{H}_{S_2})$. Hasse embedding labeling trick is similar to the zero-one labeling trick for set in Definition 10. It assigns nodes outside the target poset the same label and distinguishes nodes inside based on their isomorphism class in the Hasse diagram, while the zero-one labeling trick does not differentiate nodes inside the poset.

The above poset labeling trick can work on posets with arbitrary complex partial orders, at the cost of first identifying node isomorphism types in the Hasse diagram. In most real-world tasks, differentiating non-isomorphic nodes in Hasse diagrams is usually quite easy.

For example, in the directed link prediction task, the target posets all have same simple Hasse diagram: only two roles exist in the poset—source node and target node of the link, which is shown in Figure 4(a). Then we can assign a unique color to each equivalent class of isomorphic nodes in the Hasse diagram as the node labels, e.g., giving 1 to the source node, 2 to the target node, and 0 to all other nodes in directed link prediction. We can also design other simple poset labeling tricks. Two cases are discussed in the following.

Linear Order Set. Linear order set means a poset whose each pair of nodes are comparable, so that the Hasse diagram is a chain as shown in Figure 4(b). Therefore, S can be sorted in $u_1 \leq_S u_2 \leq_S u_3 \leq_S ... \leq_S u_k$, where $S = (U, \leq_S), U = \{u_1, u_2, ..., u_k\}$. Then we can assign u_i label i and give nodes outside S 0 label. Such a labeling trick is a valid poset labeling trick and can be used to learn paths with different lengths.

Nearly Linear Order Set. Nearly linear order set means there exists a partition of S, $\{S_1, S_2, ..., S_l\}$, $\leq_S = \bigcup_{i=1}^{l-1} S_i \times S_{i+1}$. As shown in Figure 4(c), the Hasse diagram is nearly a chain whose nodes are replaced with a set of nodes with no relations. We can assign nodes in S_i label i and give nodes outside S 0 label. It is still a valid poset labeling trick. Nearly linear order set can describe a group in an institute, where the top is the leader.

5.2 Poset Labeling Trick Expressivity

We first show that poset labeling trick enables maximum expressivity for poset learning.

Proposition 20 *For any node poset S in graph $\mathbf{A}$ and S' in graph $\mathbf{A'}$, given a set labeling trick, $(S, \mathbf{A}) \simeq (S', \mathbf{A'}) \Leftrightarrow \mathbf{A}^{(S)} \simeq \mathbf{A'}^{(S')}$.*

Proposition 20 shows that structural poset representation is equivalent to the structural whole graph representation of labeled graph. Poset labeling trick can also bridge the gap between node representations and poset representations.

Theorem 21 *Given an NME GNN and an injective aggregation function AGG, for any node posets S, S' in graphs $\mathbf{A}, \mathbf{A'}$, $GNN(S, \mathbf{A}^{(S)}) = GNN(S', \mathbf{A'}^{(S')}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A'})$, where $GNN(S, \mathbf{A}^{(S)}) = AGG(\{GNN(u, \mathbf{A}^{(S)} \mid u \in S\}))$.*

Theorem 21 shows that with an NME GNN, poset labeling trick will produce structural representations of posets. To augment this theorem, we also discuss 1-WL-GNNs with poset labeling trick. 1-WL-GNNs cannot capture any partial order information and cannot differentiate arbitrary different posets with the same set of nodes. Differentiating different posets with different sets is also hard for 1-WL-GNNs as they fail to capture relations between nodes. Poset labeling trick can help in both cases.

Proposition 22 *In any non-attributed graph with n nodes, if the degree of each node in the graph is between 1 and $\big((1-\epsilon)\log n\big)^{1/(2h+2)}$ for any constant $\epsilon > 0$, there exist $w(n^{2\epsilon})$ pairs of links and $w((n!)^2)$ pairs of non-isomorphic node posets such that any h-layer 1-WL-GNN produces the same representation, while with Hasse embedding labeling trick 1-WL-GNN can distinguish them.*

Proposition 22 illustrates that poset labeling trick can help 1-WL-GNNs distinguish significantly more pairs of node posets.

6. Subset Labeling Trick for Multi-Node Representation Learning

Besides set labeling trick, there exist other methods that append extra features to the adjacency to boost GNNs. Among them, ID-GNN (You et al., 2021) and NBFNet (Zhu et al., 2021) assign special features to only one node in the target node set and also achieve outstanding performance. In this section, we propose subset labeling trick. As its name implies, subset labeling trick assigns labels only to a subset of nodes in the target node set. We compare set labeling trick with subset labeling trick in different problem settings. In some cases, subset labeling trick is even more expressive than set labeling trick.

6.1 Subset Labeling Trick

Similar to set labeling trick, subset labeling trick also have two properties.

Definition 23 ***(subset labeling trick)*** *Given set S in graph $\mathbf{A}$ and its subset $P \subseteq S$, we stack a labeling tensor $\mathbf{L}(P, \mathbf{A}) \in \mathbb{R}^{n\times n\times d}$ in the third dimension of $\mathbf{A}$ to get a new $\mathbf{A}^{(P)} \in \mathbb{R}^{n\times n\times(k+d)}$, where $\mathbf{L}$ satisfies: $\forall S, \mathbf{A}, S', \mathbf{A'}, P \subseteq S, P' \subseteq S', \pi \in \Pi_n$,*

1. *(target-subset-distinguishing)* $\mathbf{L}(P, \mathbf{A}) = \pi(\mathbf{L}(P', \mathbf{A'})) \Rightarrow P = \pi(P')$.
2. *(permutation equivariance)* $P = \pi(P'), \mathbf{A} = \pi(\mathbf{A'}) \Rightarrow \mathbf{L}(P, \mathbf{A}) = \pi(\mathbf{L}(P', \mathbf{A'}))$.

Like set labeling trick, subset labeling trick distinguishes the selected subset in the target set and keeps permutation equivariance. However, it does not need to distinguish all target nodes. Subset(k) labeling trick means the subset size is k.

Subset zero-one labeling trick is a simplest subset labeling trick fulfilling the requirements in Definition 23.

Definition 24 ***(Subset zero-one labeling trick)*** *Given a graph $\mathbf{A}$, a set of nodes S to predict, and a subset $P \subseteq S$, we give it a diagonal labeling matrix $\mathbf{L}(P, \mathbf{A}) \in \mathbb{R}^{n\times n\times 1}$ such that $\mathbf{L}(P, \mathbf{A})_{i,i,1} = 1$ if $i \in P$ and $\mathbf{L}(P, \mathbf{A})_{i,i,1} = 0$ otherwise.*

To explain a bit, the subset zero-one labeling trick assigns label 1 to nodes in the selected subset P, and label 0 to all nodes not in P. It only contains the subset identity information.

Then a natural problem arises: how to select subset P from the target node set S? Motivated by previous methods, we propose two different routines: subset-pooling and one-head.

6.2 How to Select Subset

6.2.1 Subset Pooling

ID-GNN (You et al., 2021) proposes an a GNN for node set learning. For each node in the target node set, it labels the node one and all other nodes zero. Then, it uses a 1-WL-GNN to produce the representations of the node. By pooling all node representations, ID-GNN produces the node set representation. As isomorphic node sets can have different embeddings due to different subset selections, choosing only one node randomly can break permutation equivariance. But pooling the representation of all subset selection eliminates the non-determinism caused by selection and solves this problem. Generalizing this method,

we propose the *subset pooling routine*. Subset(k) pooling enumerates all size-k subsets and then pools the embeddings of them.

$$\mathrm{AGG}(\{\mathrm{GNN}(S, \mathbf{A}^{(P)}) \mid P \subseteq S, |P| = k\}),$$

where AGG is an injective set aggregation function.

As for all $\pi \in \Pi_n$ and target node set S in graph $\mathbf{A}$,

$$\mathrm{AGG}(\{\mathrm{GNN}(S, \mathbf{A}^{(P)}) \mid P \subseteq S, |P| = k\}) = \mathrm{AGG}(\{\mathrm{GNN}(\pi(S), \pi(\mathbf{A})^{(P)}) \mid P \subseteq \pi(S), |P| = k\}),$$

the subset pooling routine keeps permutation equivariance.

6.2.2 One Head Routine

Contrary to the subset pooling routine, link prediction model NBFNet (Zhu et al., 2021) labels only one head of the link. This design breaks permutation equivariance but improves the scalability. We propose the *one head routine* to generalize this method to general node set tasks. It selects only one subset to label. Some policies are shown in the following.

- *Random Selection.* For a target set, we can select a subset in it randomly. For example, we can randomly choose one head of each target edge in link prediction task.

- *Graph Structural Selection.* We can select a node with maximum degree in the target node set. Note that it cannot keep permutation equivariance either.

- *Partial Order Relation Selection.* If the least element exists in a poset, we can choose it as the subset. For example, in directed link prediction task, the source node of each link can be the subset. This method can keep permutation equivariance.

6.2.3 Complexity

The efficiency gain of subset labeling trick compared with set labeling trick comes from sharing results across target node sets. GNN with set labeling trick has to compute the representations of each target node set separately. With the target node distinguishing property, no labeling trick can remain unchanged across different target nodes sets. Therefore, the input adjacency will change and node representations have to be reproduced by the GNN.

In contrast, GNN with subset labeling trick can compute the representations of multiple node sets with the same selected subset simultaneously. The subset label is only a function of the selected subset and the graph, so we can maintain the subset label for different target node sets by choosing the same subset. For example, in link prediction task, all links originating from a node share this same source node. By choosing the source node as the subset, these links have the same label and input adjacency to GNN, so the node representations produced by the GNN can be reused. This routine is especially efficient in the knowledge graph completion setting, where a query involves predicting all possible tail entities connected from a head entity with a certain relation.

6.3 Expressivity

When the subset size k equals the target node set size $|S|$, subset labeling trick is equivalent to set labeling trick. What is more interesting is, when $k = |S| - 1$, subset labeling trick with the subset pooling routine can achieve the same power as set labeling trick.

Theorem 25 *Given an NME GNN, for any graph $\mathbf{A}, \mathbf{A}'$, and node sets S, S' in $\mathbf{A}, \mathbf{A}'$ respectively, we have*

$$AGG(\{GNN(S, \mathbf{A}^{(P)}) \mid P \subseteq S, |P| = |S| - 1\}) = AGG(\{GNN(S', \mathbf{A}'^{(P')}) \mid P' \subseteq S', |P'| = |S'| - 1\}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}'). \quad (1)$$

Theorem 25 illustrates that when the selected subset is of $|S| - 1$ size, GNNs can produce structural representation with the subset-pooling routine. This theorem is especially useful when $|S| = 2$, in other words, link prediction task. Labeling only one node each time and pooling the two results can achieve the same high expressivity.

Under the one head routine, we have the following theorem.

Theorem 26 *Given an NME GNN, for any graph $\mathbf{A}, \mathbf{A}'$, and node sets S, S' in $\mathbf{A}, \mathbf{A}'$ respectively, we have*

$$(S, \mathbf{A}) \not\simeq (S', \mathbf{A}') \Rightarrow \forall P \subseteq S, P' \subseteq S', |P| = |S| - 1, |P'| = |S'| - 1, GNN(S, \mathbf{A}^{(P)}) \neq GNN(S, \mathbf{A}'^{(P')}). \quad (2)$$

Though one-head routine may produce different representations for isomorphic sets, the above theorem shows that it maintains the capacity to differentiate non-isomorphic sets.

For larger target node set, subset($|S| - 1$) labeling trick is of little use, as the $|S| - 1$ labeling can hardly be reused by other target sets. In contrast, we focus on the expressivity of subset(1) labeling trick, since it is much more common for target node sets to share node rather than sharing another ($|S| - 1$) node set.

When using NME GNN, according to Theorem 12, set labeling trick leads to the highest expressivity. The problem left is whether subset(1) labeling trick can help NME GNN produce structural representations.

Proposition 27 *Given an NME GNN, there exists pairs of set S in graph $\mathbf{A}$ and set S' in graph $\mathbf{A}'$ such that $AGG(\{GNN(u, \mathbf{A}^{(u)}) \mid u \in S\}) = AGG(\{GNN(u', \mathbf{A}'^{(u')}) \mid u' \in S'\})$ while $(S, \mathbf{A}) \not\simeq (S', \mathbf{A}')$.*

Proposition 27 shows that with NME GNN, subset(1) labeling trick cannot learn structural representation and is less expressive than set labeling trick. However, using 1-WL-GNNs, the expressivity of subset(1) labeling trick is incomparable to that of set labeling trick. In other words, there exists non-isomorphic node sets which are distinguishable by subset(1) labeling trick and indistinguishable by set labeling trick, and vice versa.

Proposition 28 *Given a 1-WL-GNN, there exists $S, \mathbf{A}, S', \mathbf{A}'$ such that $(S, \mathbf{A}) \not\simeq (S', \mathbf{A}')$, $AGG(\{GNN(u, \mathbf{A}^{(u)}) \mid u \in S\}) \neq AGG(\{GNN(u', \mathbf{A}'^{(u')}) \mid u' \in S'\})$ while $GNN(S, \mathbf{A}^S) = GNN(S', \mathbf{A}'^{(S')})$. There also exists $S, \mathbf{A}, S', \mathbf{A}'$ such that $(S, \mathbf{A}) \not\simeq (S', \mathbf{A}')$, $AGG(\{GNN(u, \mathbf{A}^{(u)}) \mid u \in S\}) = AGG(\{GNN(u', \mathbf{A}'^{(u')}) \mid u' \in S'\})$ while $GNN(S, \mathbf{A}^{(S)}) \neq GNN(S', \mathbf{A}'^{(S')})$.*

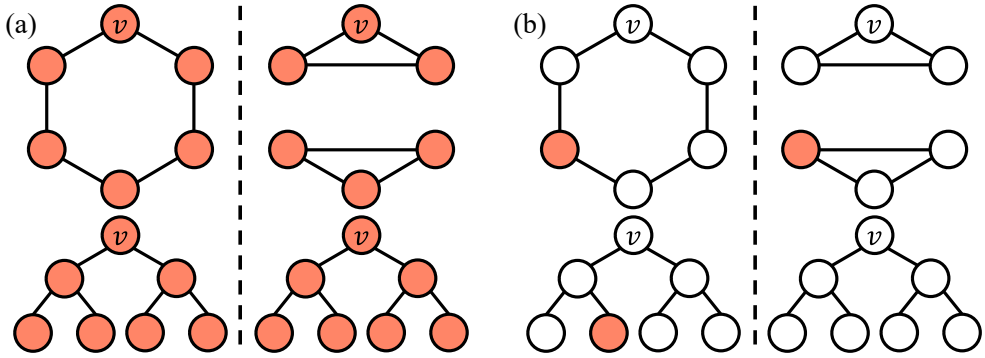

Figure 5: An example of when subset labeling trick differentiates two node sets, while set labeling trick does not. First row: labeled graphs. Second row: rooted subtrees of v.

And 1-WL-GNN with subset(1) labeling trick can also differentiate many pairs of node sets that 1-WL-GNN cannot differentiate, as shown in the following theorem.

Proposition 29 *In any non-attributed graph with n nodes, if the degree of each node in the graph is between 1 and $\left((1-\epsilon)\log n\right)^{1/(2h+2)}$ for any constant $\epsilon \in \left(\frac{\log\log n}{(2h+2)\log n}, 1\right)$, there exist $w(n^{2\epsilon})$ pairs of links and $w(2^n n^{3\epsilon-1})$ pairs of non-isomorphic node sets such that any h-layer 1-WL-GNN produces the same representation, while with subset(1) labeling trick 1-WL-GNN can distinguish them.*

6.3.1 Why Subset Labeling Trick Outperforms Labeling Trick in Some Cases?

In this section, we take a closer look at some special cases and then give some intuitions on subset labeling trick and set labeling trick. NME GNN is too expressive to show some weakness of set labeling trick, so we focus on 1-WL-GNN.

Subset labeling trick helps differentiate nodes with the same label. Taking the two graphs in Figure 5 as an example, the target set is the whole graph. With zero-one labeling trick, 1-WL-GNN cannot differentiate them as all nodes in the two graphs have the same rooted subtree (see Figure 5a). However, subset zero-one labeling trick can solve this problem. The rooted subtree in the first graph always contains a nodes with label 1, whereas in the second graph, the rooted subtree may sometimes contain no labeled nodes, leading to different 1-WL-GNN embeddings.

The drawback of subset labeling trick is that it captures pair-wise relation only and loses high-order relations. As shown in Figure 6, the two target node sets (each containing three nodes) are non-isomorphic, but every node pair from the first set is isomorphic to a node pair from the second set. This difference is also reflected in the rooted subtree of target nodes (see the bottom of Figure 6), where set labeling trick (Figure 6a) can differentiate v while subset(1) labeling trick (Figure 6b) cannot.

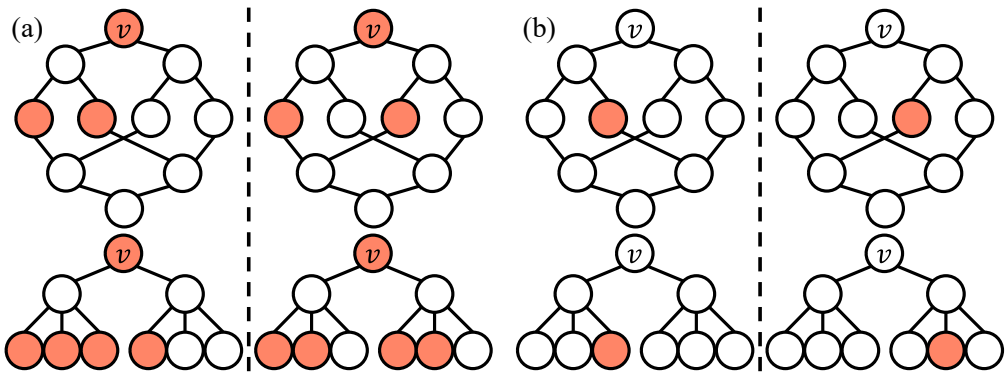

Figure 6: An example of when subset labeling trick fails to differentiate two node sets while set labeling trick does. First row: labeled graphs. Second row: rooted subtrees of v.

7. Comparison between Labeling Trick and High-Order Graph Neural Network

Unlike ordinary GNNs which produce single-node representations, High-Order Graph Neural Networks (HOGNNs) generate representations for node tuples. HOGNNs encompass various approaches, including k-dimensional Graph Neural Networks (k-GNNs) (Morris et al., 2019) inspired by the k-dimensional Weisfeiler-Lehman test (k-WL) (Cai et al., 1992), Provably Powerful Graph Neural Networks (Maron et al., 2019a) based on the 2-dimensional folklore Weisfeiler-Lehman test (2-FWL), k-Invariant Graph Networks (k-IGN) (Maron et al., 2019b), Local Relational Pooling methods (Chen et al., 2020) that create permutation-invariant functions with adjacency matrices as input, and subgraph GNNs (Bevilacqua et al., 2022; Zhao et al., 2022; Zhang and Li, 2021; Qian et al., 2022; Zhang et al., 2023) which apply ordinary 1-WL-GNNs to subgraphs extracted from the original graph.

These methods all target whole-graph tasks by pooling the generated node tuple representations to graph representations, whereas our labeling trick are designed for multi-node tasks. Nevertheless, HOGNNs also yield representations for node tuples and can be employed for multi-node tasks. Moreover, their ability to handle multi-node tasks is closely linked to their effectiveness in whole-graph tasks as follows.

Proposition 30 *Let $c(S, \mathbf{A})$ denote the color produced by a HOGNN for a graph $G = (V, E, \mathbf{A})$ and a node tuple $S \in V^k$ in the graph. Let AGG denote an injective pooling function. Given two graphs $G_1 = (V_1, E_1, \mathbf{A}_1), G_2 = (V_2, E_2, \mathbf{A}_2)$, $AGG(\{\{c(S, \mathbf{A}_1) \mid S \in V_1^k\}\}) \neq AGG(\{\{c(S, \mathbf{A}_2) \mid S \in V_2^k\}\})$ (HOGNNs can differentiate the two graphs) is equivalent to the node tuple embedding function c being able to differentiate two multisets of node tuples in two graphs.*

A direct corollary is that if there exist two non-isomorphic graphs that a HOGNN cannot differentiate, then there exist two node tuples that the tuple representations output by HOGNN cannot differentiate. Moreover, if the node tuple embedding function c_1 is

more expressive than c_2, such that $c_1(S_1, \mathbf{A}_1) = c_1(S_2, \mathbf{A}_2) \Rightarrow c_2(S_1, \mathbf{A}_1) = c_2(S_2, \mathbf{A}_2)$, the HOGNN corresponding to c_1 is also more expressive than that corresponding to c_2. Therefore, we can establish expressivity comparisons between labeling tricks for multi-node representations with HOGNNs by comparing their node tuple embedding functions. Following Zhou et al. (2023), we first define the comparison between two HOGNNs for whole-graph representations.

Definition 31 *For any algorithm A and B, we denote the final color of graph G computed by them as $c_A(G)$ and $c_B(G)$. We say:*

- *A is **more expressive** than B ($B \preceq A$) if for any pair of graphs G and H, $c_A(G) = c_A(H) \Rightarrow c_B(G) = c_B(H)$. Otherwise, there exists a pair of graphs that B can differentiate while A cannot, denoted as $B \npreceq A$.*
- *A is **as expressive as** B ($A \cong B$) if $B \preceq A \wedge A \preceq B$.*
- *A is **strictly more expressive** than B ($B \prec A$) if $B \preceq A \ \wedge \ A \ncong B$, i.e., for any pair of graphs G and H, $c_A(G) = c_A(H) \Rightarrow c_B(G) = c_B(H)$, and there exists at least one pair of graphs G, H s.t. $c_B(G) = c_B(H), c_A(G) \neq c_A(H)$.*
- *A and B are **incomparable** ($A \nsim B$) if $A \npreceq B \ \wedge \ B \npreceq A$. In this case, A can distinguish a pair of non-isomorphic graphs that cannot be distinguished by B and vice versa.*

k-dimensional Weisfeiler-Lehman (k-WL) test has strong expressivity and forms the basis of HOGNNs' expressivity hierarchy. It assigns colors to all k-tuples and iteratively updates them. The initial colors $c_k^0(S, G)$ of tuples $S \in V(G)^k$ are determined by their isomorphism types (Maron et al., 2019a). Two tuples $S \in [n]^k$ in graph G, and $S' \in [n]^k$ in graph G' receive the same isomorphism type if and only if (1) there exists a permutation function π such that $\pi(S_i) = S'_i$ for all $i = 1, 2, ..., k$; and (2) the subgraphs $G[S]$ and $G'[S']$ induced by tuples S and S' (with nodes S_i in $G[S]$ and nodes S'_i in $G'[S']$ assigned extra label i correspondingly for $i = 1, 2, ..., k$) are isomorphic. At the t-th iteration, the color updating scheme is

$$c_k^t(S, G) = \text{Hash}(c_k^{t-1}(S, G), (\{\{c_k^{t-1}(\psi_i(S, u), G) \mid u \in V(G)\}\} \mid i \in [k])),$$

where $\psi_i(S, u)$ means replacing the i-th element in S with u. The color of S is updated by its original color and the color of its high-order neighbors $\psi_i(S, u)$. The iterative update continues until the color converges, e.g. $\forall S, S' \in V(G)^k, c_k^{t+1}(S, G) = c_k^{t+1}(S', G) \Leftrightarrow c_k^t(S, G) = c_k^t(S', G)$. Let $c_k(S, G)$ denote the k-WL color of tuple S in graph G. The color of the whole graph is the multiset of all tuple colors,

$$c_k(G) = \text{Hash}(\{\{c_k(S, G) \mid S \in V(G)^k\}\}).$$

k-WL can also be used to produce l-tuple representations ($l \leq k$), Given $S \in V(G)^l$, its color is

$$c_k(S) = \text{Hash}(\{\{c_k(S \| S', G) \mid S' \in V(G)^{k-l}\}\}),$$

where $\|$ means concatenation. The HOGNN corresponding to this tuple representation is $\text{Hash}(\{\{c_k(S) \mid S \in V(G)^l\}\}$. This algorithm, namely k-WL with l-pooling in this work, shares the same expressivity as k-WL.

Proposition 32 *Given $l < k$, k-WL is as expressive as k-WL with l pooling.*

Therefore, we slightly abuse the notation of k-WL and use k-WL instead of k-WL with l pooling when analyzing k-WL for l-tuple representation.

7.0.1 k, l-WL and Poset Labeling Trick for Linear Order Set

As most HOGNNs learn representations for node tuples, we first compare HOGNNs with the labeling trick for node tuples, where node tuple is essentially poset with linear order. We use k, l-WL (Zhou et al., 2023) to represent a general framework for HOGNNs, which includes k-WL-based methods (Morris et al., 2019), subgraph GNNs (Zhang and Li, 2021; Zhao et al., 2022; Qian et al., 2022; Bevilacqua et al., 2022; Zhang et al., 2023) and relational pooling (Chen et al., 2020).

Definition 33 *(k, l-WL) For a graph $G = (V, E, \mathbf{A})$, the graph color produced by k, l-WL is as follows.*

1. *For each l-tuple of node S, the labeled graph G^S is G with the i-th node S_i in tuple S augmented with an additional feature i.*
2. *Runs k-WL on each labeled graph G^S, leading to graph color $c_{k,l}(S, \mathbf{A})$.*
3. *The final color of graph G is* $\text{HASH}(\{\{c_{k,l}(S, \mathbf{A}) \mid S \in V^l\}\})$.

k, l-WL establishes a fine expressivity hierarchy for GNNs.

Proposition 34 *(Zhou et al., 2023) For all $k \geq 2, l \geq 0$*

- *$k+1, l$-WL is strictly more expressive than $k, l+1$-WL.*
- *$k, l+1$-WL is strictly more expressive than k, l-WL.*
- *$k+1, l$-WL is strictly more expressive than k, l-WL.*
- *There exist two graphs that $2, l$-WL can differentiate while $l+1$-WL cannot.*

Note that similar to k, l-WL, the poset labeling trick for linear order sets also assigns node indices in the tuple as additional node features and runs GNN on the augmented graph. Therefore, we have

Corollary 35 *Given two graphs $G_1 = (V_1, E_1, \mathbf{A}_1), G_2 = (V_2, E_2, \mathbf{A}_2)$ and two node tuples $S_1 \in V_1^l, S_2 \in V_2^l$, the poset labeling trick with k-WL equivalent GNN can differentiate S_1, S_2 if and only if k, l-WL produces different tuple colors $c_{k,l}(S_1, \mathbf{A}_1) \neq c_{k,l}(S_2, \mathbf{A}_2)$.*

In other words, poset labeling trick for linear order sets (i.e., node tuples) combined with k-WL is equivalent in expressivity to k, l-WL where l is the size of the set. Thus we can readily inherit the k, l-WL hierarchy to analyze labeling trick. For example, in real-world applications, the labeling trick is typically used with 1-WL-GNNs, which have the same expressivity as 2-WL. Therefore,

Corollary 36 *Using 1-WL-GNN together with the poset labeling trick for linear order sets, for node tuples of size l, there exist two node tuples that $l+1$-WL cannot differentiate, while 1-WL-GNN with the labeling trick can differentiate. Moreover, for any two node tuples of size l, if $l+2$-WL cannot differentiate, 1-WL-GNN with the labeling trick cannot either.*

Despite having the same expressivity, as the labeling trick only needs to compute representation of the query node tuple, it can be much more scalable than HOGNNs for multi-node tasks (saving n^l times time and space, where l is the size of node tuple).

8. Labeling trick for hypergraph

Graph is appropriate to describe bilateral relations between entities. However, high-order relations among several entities are also worth studying (Agarwal et al., 2006). Hypergraph, composed of nodes and hyperedges, can model such high-order relations naturally. In this section, we study multi-node representation learning in hypergraphs.

We consider a hypergraph $H := (V, E, \boldsymbol{H}, \mathbf{X}^V, \mathbf{X}^E)$, where V is the node set $\{1, 2, ..., n\}$, E is the hyperedge set $\{1, 2, ..., m\}$, and $\boldsymbol{H} \in \{0, 1\}^{n\times m}$ is the incidence matrix with $\boldsymbol{H}_{i,j} = 1$ if node i is in hyperedge j and 0 otherwise. Each hyperedge contains at least one node. $\mathbf{X}^V \in \mathbb{R}^{n\times d}$ and $\mathbf{X}^E \in \mathbb{R}^{m\times d}$ are node and hyperedge features respectively, where $\mathbf{X}^V_{i,:}$ is of node i, and $\mathbf{X}^E_{j,:}$ is of hyperedge j.

We define a hypergraph permutation $\pi = (\pi_1, \pi_2) \in \Pi_n \times \Pi_m$. Its action on a hypergraph $H = (V, E, \boldsymbol{H}, \mathbf{X}^V, \mathbf{X}^E)$ is $\pi(H) = (\pi_1(V), \pi_2(E), \pi(\boldsymbol{H}), \pi_1(\mathbf{X}^V), \pi_2(\mathbf{X}^E))$, where incidence matrix permutation is $\pi(\boldsymbol{H})_{\pi_1(i),\pi_2(j)} = \boldsymbol{H}_{i,j}$.

The graph isomorphism and poset-graph isomorphism of hypergraph are defined as follows.

Definition 37 *Hypergraphs H, H' are isomorphic iff there exists $\pi \in \Pi_n \times \Pi_m$, $\pi(H) = H'$. Given node posets S in H and S' in H', $(S, H), (S', H')$ are isomorphic iff there exists $\pi = (\pi_1, \pi_2) \in \Pi_n \times \Pi_m$, $(\pi_1(S), \pi(H)) = (S', H')$.*

We can define labeling trick for hypergraph similar to that of graph from scratch. However, converting the hypergraph problem to a graph problem is more convenient. We formalize the known convertion (Bretto, 2013) as follows.

Definition 38 ***(Incidence graph)*** *Given a hypergraph $H = (V, E, \boldsymbol{H}, \mathbf{X}^V, \mathbf{X}^E)$, $V = \{1, 2, ..., n\}$, $E = \{1, 2, ..., m\}$, $\boldsymbol{H} \in \{0, 1\}^{n\times m}$, $\mathbf{X}^V \in \mathbb{R}^{n\times d}$,$\mathbf{X}^E \in \mathbb{R}^{m\times d}$, its incidence graph is $IG_H = (V_H, E_H, \mathbf{A})$, where the node set $V_H = \{1, 2, ..., n, n+1, ..., n+m\}$, edge set $E_H = \{(i, j) \mid i \in V, j \in E, \boldsymbol{H}_{i,j} = 1\}$, adjacency tensor $\mathbf{A} \in \mathbb{R}^{(n+m)\times(n+m)\times(d+1)}$. For all $i \in V, j \in E$, $\mathbf{A}_{i,i,:d} = \mathbf{X}^V_{i,:}$, $\mathbf{A}_{i,i,d+1} = \mathbf{X}^V_{i,:}$, $\mathbf{A}_{n+j,n+j,:d} = \mathbf{X}^E_{j,:}$, $\mathbf{A}_{i,n+j,d+1} = \boldsymbol{H}_{i,j}$. All other elements in $\mathbf{A}$ are 0.*

The incidence graph IG_H considers H's nodes and hyperedges both as its nodes. Two nodes in IG_H are connected iff one is a node and the other is a hyperedge containing it in H.

The incidence graph contains all information in the hypergraph. Hypergraph isomorphism and poset-hypergraph isomorphism are equivalent to the graph isomorphism and poset-graph isomorphism in the corresponding incidence graphs.

Theorem 39 *Given node posets S in hypergraph H, S' in hypergraph H', $(S, H) \simeq (S', H')$ iff $(S, IG_H) \simeq (S', IG_{H'})$.*

Therefore, a hypergraph task can be converted to a graph task. Labeling tricks can be extended to hypergraph by using them on the corresponding incidence graph.

Corollary 40 *Given an NME GNN, and an injective aggregation function AGG, for any S, H, S', H', let $\mathbf{A}, \mathbf{A'}$ denote the adjacency tensors of graphs $IG_H, IG_{H'}$ respectively. Then $GNN(S, \mathbf{A}^{(S)}) = GNN(S', \mathbf{A'}^{(S')}) \Leftrightarrow (S, H) \simeq (S', H')$.*

With NME GNN, set labeling trick can still produce structural representations on hypergraph. This enables us to boost the representation power of hyperedge prediction tasks.

9. Related work

There is emerging interest in recent study of graph neural networks' expressivity. Xu et al. (2019) and Morris et al. (2019) first show that the 1-WL test bounds the discriminating power of GNNs performing neighbor aggregation. Many works have since been proposed to increase the power of GNNs by simulating higher-order WL tests (Morris et al., 2019; Maron et al., 2019a; Chen et al., 2019; Azizian and Lelarge, 2021), approximating permutation equivariant functions (Maron et al., 2019b; Geerts, 2020; Maron et al., 2019a; Puny et al., 2022; Chen et al., 2020), , encoding subgraphs (Frasca et al., 2022; Zhang and Li, 2021; Feng et al., 2022), utilizing graph spectral features (Kreuzer et al., 2021; Lim et al., 2023), etc. However, most previous works focus on improving GNN's whole-graph representation power. Little work has been done to analyze GNN's substructure representation power. Srinivasan and Ribeiro (2020) 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 labeling tricks can also 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 node set (for example link) representations (Grover and Leskovec, 2016; Kipf and Welling, 2016; Hamilton et al., 2017). However, we argue in this paper that simply aggregating node representations fails to discriminate a large number of non-isomorphic node sets (links), and with labeling trick the aggregation of structural node representations leads to structural representations.

Li et al. (2020) proposed distance encoding (DE), whose implementations based on S-discriminating distances can be shown to be specific labeling tricks. You et al. (2019) also noticed that structural node representations of GNNs cannot capture the dependence (in particular distance) 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 with similar distances to the anchor nodes, instead of nodes with similar neighborhoods, have similar embeddings. Thus, P-GNN cannot learn structural node/link representations. P-GNN also cannot scale to large datasets.

Finally, although labeling trick is formally defined in our conference paper (Zhang et al., 2021a), various forms of specific labeling tricks have already been used in previous works. To

our best knowledge, SEAL (Zhang and Chen, 2018) proposes to add shortest path distance to target node to each node's feature, which is designed to improve GNN's link prediction power. To our best knowledge, it is the first labeling trick. It is later adopted in the completion of inductive knowledge graphs (Teru et al., 2020) and matrix completion (Zhang and Chen, 2020), and is generalized to DE (Li et al., 2020) and GLASS (Wang and Zhang, 2022), which works for the cases $|S| > 2$. Wan et al. (2021) use labeling trick for hyperedge prediction. Besides these set labeling tricks, some labeling methods similar to the subset labeling trick also exist in existing works. ID-GNN (You et al., 2021) and NBFNet (Zhu et al., 2021) both use a mechanism equivalent to the one head routine of subset labeling trick. RWL (Huang et al., 2023) further generalize these methods to a general framework similar to our subset labeling trick with subset size $= 1$ and connects its expressivity with logical boolean classifier.

10. Experiments

Our experiments include various multi-node representation learning tasks: undirected link prediction, directed link prediction, hyperlink prediction, and subgraph prediction. Labeling trick boosts GNNs on all these tasks. All metrics in this section are the higher the better. Datasets are detailed in Appendix C. Our code is available at `https://github.com/GraphPKU/LabelingTrick`. In all experiments, we use GNNs without labeling trick (NO) for ablation.

10.1 Undirected link prediction

In this section, we use a two-node task, link prediction, to empirically validate the effectiveness of set and subset labeling trick.

Following the setting in SEAL (Zhang and Chen, 2018), we use eight datasets: USAir, NS, PB, Yeast, C.ele, Power, Router, and E.coli. These datasets are relatively small. So we additionally use four large datasets in Open Graph Benchmark (OGB) (Hu et al., 2020): `ogbl-ppa`, `ogbl-collab`, `ogbl-ddi`, `ogbl-citation2`. To facilitate the comparison, we use the same metrics, including auroc, Hits@K, and MRR, as in previous works.

We use the following baselines for comparison. We use 4 non-GNN methods: CN (Common-Neighbor), AA (Adamic-Adar), MF (matrix factorization) and Node2vec (Grover and Leskovec, 2016). CN and AA are two simple link prediction heuristics based on counting common neighbors. MF uses free-parameter node embeddings trained end-to-end as the node representations. Two set labeling trick methods are used: ZO and SEAL. ZO uses the zero-one labeling trick, and SEAL uses the DRNL labeling trick (Zhang and Chen, 2018). Three subset labeling trick methods are compared: subset zero-one labeling trick with subset pooling (ZO-S), subset distance encoding labeling trick with subset pooling (DE-S), subset zero-one labeling trick with one-head routine (ZO-OS).

Results and discussion. We present the main results in Table 1. Compared with all non-GNN methods, vanilla 1-WL-GNN with no labeling trick (NO) gets lower auroc on almost all datasets. However, with labeling trick or subset labeling trick, 1-WL-GNN can outperform the baselines on almost all datasets. ZO, SEAL use set labeling trick and outperform non-GNN methods by 4% and 9% respectively on average. The performance difference between ZO and SEAL illustrates that labeling trick implementation can still affect the

	USAir	NS	PB	Yeast	Cele	Power	Router	Ecoli
CN	$93.80_{\pm 1.22}$	$94.42_{\pm 0.95}$	$92.04_{\pm 0.35}$	$89.37_{\pm 0.61}$	$85.13_{\pm 1.61}$	$58.80_{\pm 0.88}$	$56.43_{\pm 0.52}$	$93.71_{\pm 0.39}$
AA	$95.06_{\pm 1.03}$	$94.45_{\pm 0.93}$	$92.36_{\pm 0.34}$	$89.43_{\pm 0.62}$	$\underline{86.95_{\pm 1.40}}$	$58.79_{\pm 0.88}$	$56.43_{\pm 0.51}$	$95.36_{\pm 0.34}$
NV	$91.44_{\pm 1.78}$	$91.52_{\pm 1.28}$	$85.79_{\pm 0.78}$	$93.67_{\pm 0.46}$	$84.11_{\pm 1.27}$	$76.22_{\pm 0.92}$	$65.46_{\pm 0.86}$	$90.82_{\pm 1.49}$
MF	$94.08_{\pm 0.80}$	$74.55_{\pm 4.34}$	$94.30_{\pm 0.53}$	$90.28_{\pm 0.69}$	$85.90_{\pm 1.74}$	$50.63_{\pm 1.10}$	$78.03_{\pm 1.63}$	$93.76_{\pm 0.56}$
NO	$89.04_{\pm 2.14}$	$74.10_{\pm 2.62}$	$90.87_{\pm 0.56}$	$83.04_{\pm 0.93}$	$73.25_{\pm 1.67}$	$65.89_{\pm 1.65}$	$92.47_{\pm 0.76}$	$93.27_{\pm 0.49}$
ZO	$94.08_{\pm 1.43}$	$95.60_{\pm 0.93}$	$91.82_{\pm 1.26}$	$94.69_{\pm 0.45}$	$74.94_{\pm 2.01}$	$73.85_{\pm 1.37}$	$93.21_{\pm 0.66}$	$92.09_{\pm 0.67}$
SEAL	$\mathbf{97.09_{\pm 0.70}}$	$97.71_{\pm 0.93}$	$\mathbf{95.01_{\pm 0.34}}$	$97.20_{\pm 0.64}$	$86.54_{\pm 2.04}$	$84.18_{\pm 1.82}$	$\underline{95.68_{\pm 1.22}}$	$97.22_{\pm 0.28}$
ZO-S	$\underline{96.15_{\pm 1.06}}$	$\underline{98.10_{\pm 0.67}}$	$94.15_{\pm 0.50}$	$97.41_{\pm 0.37}$	$86.31_{\pm 1.80}$	$78.31_{\pm 0.91}$	$94.52_{\pm 0.72}$	$\underline{97.48_{\pm 0.23}}$
DE-S	$94.97_{\pm 0.61}$	$\mathbf{99.29_{\pm 0.14}}$	$\underline{94.44_{\pm 0.52}}$	$\mathbf{98.17_{\pm 0.41}}$	$85.95_{\pm 0.36}$	$\mathbf{94.16_{\pm 0.14}}$	$\mathbf{99.33_{\pm 0.09}}$	$\mathbf{98.91_{\pm 0.08}}$
ZO-OS	$94.62_{\pm 0.63}$	$97.42_{\pm 0.49}$	$94.36_{\pm 0.26}$	$\underline{97.46_{\pm 0.06}}$	$\mathbf{88.04_{\pm 0.52}}$	$\underline{84.95_{\pm 0.30}}$	$93.77_{\pm 0.20}$	$95.53_{\pm 0.62}$

Table 1: Results on undirected link prediction task: auroc (%) ± standard deviation.

Dataset	collab	ddi	citation2	ppa
metrics	Hits@50	Hits@20	MRR	Hits@100
NO	$44.75_{\pm 1.07}$	$\underline{37.07_{\pm 5.07}}$	$\underline{84.74_{\pm 0.21}}$	$18.67_{\pm 1.32}$
ZO	$53.29_{\pm 0.23}$	$23.90_{\pm 0.75}$	$78.50_{\pm 1.08}$	$37.75_{\pm 3.42}$
SEAL	$\mathbf{54.71_{\pm 0.49}}$	$30.56_{\pm 3.86}$	$\mathbf{87.67_{\pm 0.32}}$	$\mathbf{48.80_{\pm 3.16}}$
ZO-OS	$49.17_{\pm 3.29}$	$\mathbf{41.24_{\pm 1.49}}$	$82.85_{\pm 0.43}$	$\underline{43.27_{\pm 1.19}}$
ZO-S	$\underline{54.69_{\pm 0.51}}$	$29.27_{\pm 0.53}$	$82.45_{\pm 0.62}$	$36.04_{\pm 4.50}$

Table 2: Results on undirected link prediction task.

expressivity of 1-WL-GNN. However, even the simplest labeling trick can still boost 1-WL-GNNs by 6%. Subset(1) labeling trick ZO-S and DE-S also achieve 9% and 11% score increase on average. Compared with ZO, though ZO-S also uses only the target set identity information, it distinguishes nodes in the target node set and achieves up to 5% performance increase on average, which verifies the usefulness of subset labeling trick. Last but not least, though subset labeling trick with one-head routine (ZO-OS) loses permutation invariance compared with subset pooling routine (ZO-S), it still achieves outstanding performance and even outperforms ZO-S on 4/8 datasets.

We also conduct experiments on some larger datasets as shown in Table 2. GNN augmented by labeling tricks achieves the best performance on all datasets.

10.2 Directed link prediction tasks

To illustrate the necessity of introducing partial order to labeling trick, we compare set labeling trick and poset labeling trick on the directed link prediction task. Following previous work (He et al., 2022), we use six directed graph datasets, namely Cornell, Texas, Wisconsin, CoraML, Citeseer, and Telegram. Our baselines includes previous state-of-the-art GNNs for directed graph, including DGCN (Tong et al., 2020b), DiGCN and DiGCNIB (Tong et al.,

	Cornell	Texas	Wisconsin	CoraML	CiteSeer	Telegram
DGCN	$70.4_{\pm 9}$	$69.7_{\pm 6}$	$69.8_{\pm 6}$	$77.2_{\pm 1}$	$71.2_{\pm 2}$	$86.4_{\pm 1}$
DiGCN	$69.3_{\pm 7}$	$69.7_{\pm 7}$	$66.2_{\pm 7}$	$75.1_{\pm 1}$	$73.0_{\pm 1}$	$78.2_{\pm 1}$
DiGCNIB	$65.7_{\pm 3}$	$63.7_{\pm 6}$	$67.6_{\pm 6}$	$75.9_{\pm 4}$	$73.7_{\pm 2}$	$79.8_{\pm 2}$
MagNet	$70.4_{\pm 8}$	$73.1_{\pm 7}$	$70.4_{\pm 7}$	$77.3_{\pm 1}$	$71.8_{\pm 1}$	$\underline{86.7_{\pm 1}}$
NO	$67.5_{\pm 7}$	$72.0_{\pm 4}$	$71.6_{\pm 5}$	$78.7_{\pm 1}$	$74.3_{\pm 1}$	$84.1_{\pm 1}$
PL	$\mathbf{71.5_{\pm 4}}$	$\mathbf{78.0_{\pm 4}}$	$\mathbf{79.0_{\pm 3}}$	$\mathbf{82.2_{\pm 1}}$	$\underline{79.8_{\pm 1}}$	$\mathbf{91.7_{\pm 1}}$
ZO	$\underline{71.1_{\pm 5}}$	$\underline{77.6_{\pm 5}}$	$\underline{74.0_{\pm 2}}$	$\underline{79.4_{\pm 1}}$	$\mathbf{79.9_{\pm 1}}$	$85.6_{\pm 1}$

Table 3: Results on directed link prediction tasks: accuracy (%) ± standard deviation.

	NDC-c	NDC-s	tags-m	tags-a	email-En	email-EU	congress
ceGCN	$61.4_{\pm 0.5}$	$42.1_{\pm 1.4}$	$59.9_{\pm 0.9}$	$54.5_{\pm 0.5}$	$61.8_{\pm 3.2}$	$66.4_{\pm 0.3}$	$41.2_{\pm 0.3}$
ceSAGE	$65.7_{\pm 2.0}$	$47.9_{\pm 0.7}$	$63.5_{\pm 0.3}$	$59.7_{\pm 0.7}$	$59.4_{\pm 4.6}$	$65.1_{\pm 1.9}$	$53.0_{\pm 5.5}$
seRGCN	$67.6_{\pm 4.9}$	$52.5_{\pm 0.6}$	$57.2_{\pm 0.3}$	$54.5_{\pm 0.6}$	$59.9_{\pm 4.0}$	$66.1_{\pm 0.6}$	$54.4_{\pm 0.4}$
FS	$\underline{76.8_{\pm 0.4}}$	$51.2_{\pm 3.2}$	$64.2_{\pm 0.6}$	$60.5_{\pm 0.2}$	$68.5_{\pm 1.6}$	$68.7_{\pm 0.2}$	$56.6_{\pm 1.1}$
NO	$60.2_{\pm 2.3}$	$45.6_{\pm 0.8}$	$56.6_{\pm 1.4}$	$56.5_{\pm 1.8}$	$56.9_{\pm 1.7}$	$57.2_{\pm 0.9}$	$54.1_{\pm 0.5}$
ZO	$\mathbf{82.5_{\pm 1.3}}$	$\mathbf{63.6_{\pm 1.5}}$	$\mathbf{71.4_{\pm 0.5}}$	$\mathbf{70.4_{\pm 0.8}}$	$\underline{66.1_{\pm 1.2}}$	$\underline{72.1_{\pm 1.1}}$	$\mathbf{65.1_{\pm 0.2}}$
ZO-S	$75.8_{\pm 0.7}$	$\underline{62.2_{\pm 1.2}}$	$\underline{71.0_{\pm 0.4}}$	$\underline{69.6_{\pm 0.7}}$	$\mathbf{67.7_{\pm 1.8}}$	$\mathbf{73.3_{\pm 0.5}}$	$\underline{64.2_{\pm 0.3}}$

Table 4: Results on hyperedge prediction tasks: f1-score (%) ± standard deviation.

2020a), and MagNet (Zhang et al., 2021c). Our models include NO (vanilla 1-WL-GNN), PL (poset labeling trick which labels the source node as 1, target node as 2, other nodes as 0), ZO (zero-one labeling trick).

The results are shown in Table 3. The existing state-of-the-art method MAGNet (Zhang et al., 2021b) outperforms 1-WL-GNN by 0.25% on average. However, 1-WL-GNN with labeling trick outperforms all baselines. Moreover, poset labeling trick (PL) achieves 2% performance gain compared with the set labeling trick (ZO). These results validate the power of poset labeling trick and show that modeling partial order relation is critical for some tasks.

10.3 Hyperedge prediction task

We use the datasets and baselines in (Srinivasan et al., 2021). Our datasets includes two drug networks (NDC-c, NDC-s), two forum networks (tags-m, tags-a), two email networks (email-En, email-Eu), and a network of congress members (congress). We use four GNNs designed for hypergraph as baselines, including ceGCN, ceSAGE, seRGCN, and FS (family set) (Srinivasan et al., 2021). Our models include ZO (zero-one labeling trick), ZO-S (subset(1) labeling trick with subset pooling), NO (vanilla 1-WL-GNN).ZO and ZO-S outperform all other methods significantly.

Method	density	coreness	cutratio	ppi-bp	hpo-metab	hpo-neuro	em-user
SubGNN	$91.9_{\pm 0.6}$	$65.9_{\pm 3.1}$	$62.9_{\pm 1.3}$	$59.9_{\pm 0.8}$	$53.7_{\pm 0.8}$	$64.4_{\pm 0.6}$	$81.6_{\pm 1.3}$
Sub2Vec	$45.9_{\pm 1.2}$	$36.0_{\pm 1.9}$	$35.4_{\pm 1.4}$	$38.8_{\pm 0.1}$	$47.2_{\pm 1.0}$	$61.8_{\pm 0.3}$	$77.9_{\pm 1.3}$
NO	$47.8_{\pm 2.9}$	$47.8_{\pm 5.3}$	$81.4_{\pm 1.5}$	$61.3_{\pm 0.9}$	$59.7_{\pm 1.2}$	$66.8_{\pm 0.7}$	$84.7_{\pm 2.1}$
ZO	$\mathbf{98.4_{\pm 1.2}}$	$\mathbf{87.3_{\pm 15.0}}$	$\mathbf{93.0_{\pm 1.3}}$	$\mathbf{61.9_{\pm 0.7}}$	$\mathbf{61.4_{\pm 0.5}}$	$\mathbf{68.5_{\pm 0.5}}$	$\mathbf{88.8_{\pm 0.6}}$
ZO-S	$\underline{94.3_{\pm 6.9}}$	$\underline{75.8_{\pm 7.0}}$	$\underline{85.6_{\pm 2.5}}$	$\underline{61.7_{\pm 0.4}}$	$\underline{60.4_{\pm 1.1}}$	$\underline{67.4_{\pm 1.3}}$	$\underline{86.3_{\pm 2.5}}$

Table 5: Results on subgraph tasks: f1-score (%) ± standard deviation.

10.4 Subgraph prediction task

We use the datasets and baselines in (Alsentzer et al., 2020). We use three synthetic datasets, namely density, coreness, and cutratio, and four real-world datasets, namely ppi-bp, hpo-metab, hpo-neuro, em-user. SubGNN (Alsentzer et al., 2020) and Sub2Vec (Adhikari et al., 2018) are models designed for subgraph. Our models include ZO (zero-one labeling trick, results on ppi-bp, hpo-metab, hpo-neuro, em-user are from Wang and Zhang (2022)), ZO-S (subset labeling trick), and NO (vanilla 1-WL-GNN without labeling trick, results on ppi-bp, hpo-metab, hpo-neuro, em-user are from Wang and Zhang (2022)). Compared with NO, Labeling tricks boost vanilla 1-WL-GNN significantly. Moreover, vanilla GNN augmented by labeling trick also outperforms GNN designed for subgraph on all datasets. Moreover, ZO outperforms ZO-S, which illustates that subset labeling tricks, while ZO can capture high-order relations better as shown in Section 6.3.1.

11. Conclusions

In this paper, we proposed a theory of using GNNs for multi-node representation learning. We first pointed out the key limitation of a common practice in previous works that directly aggregates node representations as a node-set representation. To address the problem, we proposed set labeling trick which gives target nodes distinct labels in a permutation equivariant way and characterized its expressive power. We further extended set labeling trick to poset and subset labeling trick, as well as extending graph to hypergraph. Our theory thoroughly discusses different variants and scenarios of using labeling trick to boost vanilla GNNs, and provides a solid foundation for future researchers to develop novel labeling tricks.

Acknowledgments

This work is supported by the National Key R&D Program of China (2022ZD0160300) and National Natural Science Foundation of China (62276003).

Appendix A. Proofs

A.1 Proof of Proposition 11 and Proposition 20

For Proposition 11,

$$\begin{aligned}(S, \mathbf{A}) \simeq (S', \mathbf{A}') &\Leftrightarrow \exists \pi \in \Pi_n, \pi(S) = S', \pi(\mathbf{A}) = \mathbf{A}' \\ &\Leftrightarrow \exists \pi \in \Pi_n, \pi(\mathbf{L}(S, \mathbf{A})) = \mathbf{L}(S', \mathbf{A}'), \pi(\mathbf{A}) = \mathbf{A}' \\ &\Leftrightarrow \exists \pi \in \Pi_n, \pi(\mathbf{A}^{(S)}) = \mathbf{A}'^{(S')}\end{aligned}$$

For Proposition 20, we can simply replace set S above with poset.

A.2 Proof of Theorem 12

Following Zhang et al. (2021a), we restate Theorem 12: Given an NME GNN and an injective set aggregation function AGG, for any $S, \mathbf{A}, S', \mathbf{A}'$, $\mathrm{GNN}(S, \mathbf{A}^{(S)}) = \mathrm{GNN}(S', \mathbf{A}'^{(S')}) \Leftrightarrow (S, \mathbf{A}) \simeq (S', \mathbf{A}')$, where $\mathrm{GNN}(S, \mathbf{A}^{(S)}) := \mathrm{AGG}(\{\mathrm{GNN}(i, \mathbf{A}^{(S)}) | i \in S\})$.

Proof

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

To prove $\Rightarrow$, we notice that with an injective AGG,

$$\begin{aligned}&\mathrm{AGG}(\{\mathrm{GNN}(i, \mathbf{A}^{(S)})) | i \in S\}) = \mathrm{AGG}(\{\mathrm{GNN}(i, \mathbf{A}'^{(S')})) | i \in S'\}) \\ \Longrightarrow\ &\exists\ v_1 \in S, v_2 \in S', \text{ such that } \mathrm{GNN}(v_1, \mathbf{A}^{(S)}) = \mathrm{GNN}(v_2, \mathbf{A}'^{(S')}) \quad (3) \\ \Longrightarrow\ &(v_1, \mathbf{A}^{(S)}) \simeq (v_2, \mathbf{A}'^{(S')}) \quad \text{(because GNN is node-most-expressive)} \quad (4) \\ \Longrightarrow\ &\exists\ \pi \in \Pi_n, \text{ such that } v_1 = \pi(v_2), \mathbf{A}^{(S)} = \pi(\mathbf{A}'^{(S')}). \quad (5)\end{aligned}$$

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

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

Because $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A}')) \Rightarrow S = \pi(S')$, continuing from Equation (5), we have

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

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

$$\begin{aligned}&(S, \mathbf{A}) \simeq (S', \mathbf{A}') \\ \Longrightarrow\ &\exists\ \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}') \\ \Longrightarrow\ &\exists\ \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A} = \pi(\mathbf{A}'), \mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A}')) \\ \Longrightarrow\ &\exists\ \pi \in \Pi_n, \text{ such that } S = \pi(S'), \mathbf{A}^{(S)} = \pi(\mathbf{A}'^{(S')}) \\ \Longrightarrow\ &\exists\ \pi \in \Pi_n, \text{ such that } \forall v_2 \in S', v_1 = \pi(v_2) \in S, \mathrm{GNN}(v_1, \mathbf{A}^{(S)}) = \mathrm{GNN}(v_2, \mathbf{A}'^{(S')}) \\ \Longrightarrow\ &\mathrm{AGG}(\{\mathrm{GNN}(v_1, \mathbf{A}^{(S)}) | v_1 \in S\}) = \mathrm{AGG}(\{\mathrm{GNN}(v_2, \mathbf{A}'^{(S')}) | v_2 \in S'\}),\end{aligned}$$

which concludes the proof. ■

A.3 Proof of Theorem 13 and Theorem 29

Following Zhang et al. (2021a), as an h-layer 1-WL-GNN only encodes an h-hop neighbors for each node, we define locally h-isomorphism.

Definition 41 *For all $S, \mathbf{A}, S', \mathbf{A}'$, $(S, \mathbf{A})$ and $(S', \mathbf{A}')$ are locally h-isomorphic iff $(S, \mathbf{A}_{S,h}) \simeq (S', \mathbf{A}_{S',h})$, where $\mathbf{A}_{S,h}$ means the subgraph of $\mathbf{A}$ induced by the node set $\{v \in V | \exists u \in S, d_{sp}(u, v, \mathbf{A}) \leq h\}$, and $d_{sp}(u, v, \mathbf{A})$ means the shortest path distance between node u, v in graph $\mathbf{A}$.*

We restate Theorem 13(Theorem 29): In any non-attributed graph with n nodes, if the degree of each node in the graph is between 1 and $((1-\epsilon)\log n)^{1/(2h+2)}$ 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 zero-one labeling trick (subset zero-one labeling trick) the 1-WL-GNN gives u, v different representations. These two theorems can be proved together because the special cases we build can be solved by both of them.

Proof

Our proof has two steps. First, we would like to show that there are $\omega(n^{\epsilon})$ nodes that are locally h-isomorphic 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 node set 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 $d \leq \big((1-\epsilon)\log n\big)^{1/(2h+2)}$, then the number of nodes in $G_v^{(h)}$, denoted by $|V(G_v^{(h)})|$, satisfies

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

We set $K = \max_{v \in V} |V(G_v^{(h)})| \leq \big((1-\epsilon)\log n\big)^{1/2}$.

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

$$2^{\binom{K}{2}} \leq 2^{(1-\epsilon)\log n} = n^{1-\epsilon}.$$

Therefore, due to the pigeonhole principle, there exist $\omega(n/n^{1-\epsilon}) = \omega(n^{\epsilon})$ 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.

Step 2. Let us partition $V_{iso} = \cup_{i=1}^{q} V_i$ so that for all $i \in \{1, 2, ..., q\}$, nodes in V_i share the same first-hop neighbor sets. Note that all nodes in each V_i share the same neighbors, so $|V_i|$

is no more than maximum degree $((1-\epsilon)\log n)^{1/(2h+2)} < n^{\epsilon}$ when $\epsilon > \frac{1}{(2h+2)\log n}(\log\log n)$. 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 $w \in N(u) - N(v)$ (If w does not exists, then $N(u) - N(v) =$, so $N(v) - N(u) \neq \emptyset$ because of the definition of V_i. We can simply exchange u and v). 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:

$$Y \geq \sum_{i,j=1,i<j}^{q} |V_i||V_j| = \frac{1}{2}\left[(\sum_{i=1}^{q}|V_i|)^2 - \sum_{i=1}^{q}|V_i|^2\right]. \tag{6}$$

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

$$1 \leq |V_i| \leq d_w \leq \big((1-\epsilon)\log n\big)^{1/(2h+2)},$$

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.6 leads to

$$\begin{aligned} Y &\geq \frac{1}{2}[(\sum_{i=1}^{q}|V_i|)^2 - \sum_{i=1}^{q}|V_i|(\max_{j\in\{1,2,\ldots,q\}}|V_j|)] \\ &= \frac{1}{2}(\omega(n^{2\epsilon}) - \omega(n^{\epsilon})\mathcal{O}\Big(\big((1-\epsilon)\log n\big)^{1/(2h+2)}\Big) \\ &= \omega(n^{2\epsilon}), \end{aligned}$$

which concludes the proof. ■

A.4 Proof of Theorem 15

Proof This proof shares the same first step as Appendix A.3.

Step 2. Let us partition $V_{iso} = \bigcup_{i=1}^{q} V_i$, nodes in each V_i share the same one-hop neighbor. Consider two nodes $u \in V_i, v \in V_j, i \neq j$. There exists a node $w \in N(u), w \notin N(v)$ (If w does not exists, then $N(u) - N(v) =$, so $N(v) - N(u) \neq \emptyset$ because of the definition of V_i. We can simply exchange u and v). Let $\tilde{V}_{u,v,w}$ denote $V - \{u, v, w\} - N(v)$. $\tilde{V}_{u,v,w} \geq n - 3 - \big((1-\epsilon)\log n\big)^{1/(2h+2)}$. Consider arbitrary subset V' of $\tilde{V}_{u,v,w}$. Let $\mathcal{S}_1$ denote the subgraph induced by $V' \bigcup \{u, w\}$, $\mathcal{S}_2$ denote the subgraph induced by $V' \bigcup \{v, w\}$. Compared with $\mathcal{S}_2$, $\mathcal{S}_1$ has the same number of nodes. Moreover, $\mathcal{S}_2$ has edge between nodes in V' and edges between V' and w, while $\mathcal{S}_1$ further has more edge (u, w) and edges between V' and u, so the density of $\mathcal{S}_1$ is higher than $\mathcal{S}_2$. And 1-WL-GNN with zero-one labeling trick can fit density perfectly (Theorem 1 in (Wang and Zhang, 2022)), so 1-WL-GNN with labeling trick can distinguish $\mathcal{S}_1$ and $\mathcal{S}_2$, while 1-WL-GNNs cannot.

The number of pair (u, v, w) is $w(n^{2\epsilon})$. Therefore, the number of these pairs of subgraphs is bounded by

$$w(n^{2\epsilon})2^{n-3-((1-\epsilon)\log n)^{1/(2h+2)}} = w(2^n n^{3\epsilon-1}).$$

■

A.5 Proof of Theorem 22

This proof shares the same first step as Appendix A.3.

Number of link: the same as the step 2 in Appendix A.3.

Number of subgraph: similar to the step 2 in Appendix A.4. Let us partition $V_{iso} = \bigcup_{i=1}^{q} V_i$, nodes in each V_i share the same one-hop neighbor. Consider two nodes $u \in V_i, v \in V_j, i \neq j$. There exists a node $w \in N(u), w \notin N(v)$. Let $\tilde{V}_{u,v,w}$ denote $V - \{u, v, w\} - N(u)$. $|V_v| \geq n - 3 - \big((1-\epsilon)\log n\big)^{1/(2h+2)}$. Consider arbitrary subset V' of $\tilde{V}_{u,v,w}$ and a partial order $\leq_{V'}$. Let $\mathcal{S}_1$ denote the subgraph induced by poset $\big((V' \bigcup \{u, w\}), \leq_{V'} \cup \{(u, a)|a \in sV'\} \cup \{(w, a)|a \in sV' \cup \{u\}\}\big)$, $\mathcal{S}_2$ denote the subgraph induced by poset $\big(V' \bigcup \{v, w\}, \leq_{V'} \cup \{(v, a)|a \in sV'\} \cup \{(w, a)|a \in V' \cup \{v\}\}\big)$. 1-WL-GNN with labeling trick can distinguish $\mathcal{S}_1$ and $\mathcal{S}_2$ as the edges between (u, w) and (v, w) are distinct, while 1-WL-GNNs cannot.

The number of pair (u, v, w) is $w(n^{2\epsilon})$. Therefore, the number of these pairs of subgraphs is bounded by

$$w(n^{2\epsilon})w(n - 3 - \big((1-\epsilon)\log n\big)^{1/(2h+2)})! = w\Big(\big((1-\epsilon)n\big)!\Big).$$

A.6 Proof of Proposition 14

As shown in Figure 1a, 1-WL-GNN cannot count common neighbor and thus fail to implement h. Now we prove that with zero-one labeling trick, 1-WL-GNN can implement h.

Given a graph $\mathbf{A}$ and a node pair (i, j), let $z_k^{(k)}$ denote the embedding of node i at k^{th} message passing layer.

$$z_k^{(0)} = \begin{bmatrix} 1 \\ \delta_{ki} + \delta_{kj} \end{bmatrix}.$$

The first dimension is all 1 (vanilla node feature), and the second dimension is zero-one label.

The first layer is,

$$z_k^{(1)} = \begin{bmatrix} g_1(a_k^{(1)}[1]) \\ g_2(a_k^{(1)}[1]) \\ a_k^{(1)}[2] > 2 \end{bmatrix}$$

where $a_k^{(1)} = \sum_{l \in N(k)} z_l^{(0)}$, [1] means the first element of vector, and [2] means the second element.

The second layer is

$$z_k^{(2)} = \begin{bmatrix} \sum_{l \in N(k)} z_k^{(1)}[3] z_k^{(1)}[2] \\ \sum_{l \in N(k)} (1 - z_k^{(1)}[3]) z_k^{(1)}[1] \end{bmatrix}$$

The pooling layer is

$$z_{ij} = f(\{z_i[2], z_j[2]\}, \frac{z_i[1] + z_j[1]}{2})$$

A.7 Proof of Theorem 21

Proof $\Leftarrow$: When $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$, there exists a permutation π, $\pi(S) = S', \pi(\mathbf{A}) = \mathbf{A}'$.

$$\begin{aligned}
\mathrm{GNN}(S, \mathbf{A}^{(S)}) &= \mathrm{AGG}(\{\mathrm{GNN}(v, \mathbf{A}^{(S)} | v \in S\}) && (7)\\
&= \mathrm{AGG}(\{\mathrm{GNN}(\pi(v), \pi(\mathbf{A}^{(S)})) | v \in S\}) && (8)\\
&= \mathrm{AGG}(\{\mathrm{GNN}(\pi(v), \mathbf{A}'^{(S')} | v \in S\}) && (9)\\
&= \mathrm{AGG}(\{\mathrm{GNN}(v', \mathbf{A}'^{(S')}) | v' \in S'\}) && (10)\\
&= \mathrm{GNN}(S', \mathbf{A}'^{(S')}) && (11)
\end{aligned}$$

$\Rightarrow$:

$$\mathrm{GNN}(S, \mathbf{A}^{(S)}) = \mathrm{GNN}(S', \mathbf{A}'^{(S')})$$

$$\mathrm{AGG}(\{\mathrm{GNN}(v, \mathbf{A}^{(S)}) | v \in S\}) = \mathrm{AGG}(\{\mathrm{GNN}(v', \mathbf{A}'^{(S')}) | v' \in S'\})$$

As AGG is injective, There exist $v_0 \in S, v_0' \in S'$,

$$\mathrm{GNN}(v_0, \mathbf{A}^{(S)}) = \mathrm{GNN}(v_0', \mathbf{A}^{(S')})$$

As GNN is node most expressive,

$$\exists \pi, \pi(v_0) = v_0', \pi(\mathbf{A}) = \mathbf{A}', \pi(\mathbf{L}(S, \mathbf{A})) = \mathbf{L}(S', \mathbf{A}').$$

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

A.8 Proof of Theorem 25

Proof $\Leftarrow$: When $(S, \mathbf{A}) \simeq (S', \mathbf{A}')$, there exists a permutation π, $\pi(S) = S', \pi(\mathbf{A}) = \mathbf{A}'$.

$$\begin{aligned}
&\mathrm{AGG}\left(\left\{\mathrm{AGG}(\{\mathrm{GNN}(u, \mathbf{A}^{(S-\{v\})}) | u \in S\}) | v \in S\right\}\right)\\
&\quad = \mathrm{AGG}(\{\mathrm{AGG}(\{\mathrm{GNN}(\pi(u), \pi(\mathbf{A}^{(S-\{v\})}) | u \in S\}) | v \in S\})\\
&\quad = \mathrm{AGG}(\{\mathrm{AGG}(\{\mathrm{GNN}(\pi(u), \mathbf{A}'^{(\pi(S)-\{\pi(v)\})} | u \in S\}) | v \in S\})\\
&\quad = \mathrm{AGG}(\{\mathrm{AGG}(\{\mathrm{GNN}(u', \mathbf{A}'^{(S'-\{v'\})}) | u' \in S'\}) | v' \in S'\})
\end{aligned}$$

$\Rightarrow$:

$$\begin{aligned}
&\mathrm{AGG}(\{\mathrm{AGG}(\{\mathrm{GNN}(u, \mathbf{A}^{(S-\{v\})}) | u \in S\}) | v \in S\})\\
&\quad = \mathrm{AGG}(\{\mathrm{AGG}(\{\mathrm{GNN}(u', \mathbf{A}'^{(S'-\{v'\})}) | u' \in S'\}) | v' \in S'\}).
\end{aligned}$$

As AGG is injective,

$$\{\mathrm{AGG}(\{\mathrm{GNN}(u, \mathbf{A}^{(S-\{v\})}) | v \in S\}) | u \in S\} = \{\{\mathrm{AGG}(\{\mathrm{GNN}(u', \mathbf{A}^{(S'-\{v'\})}) | v' \in S'\})\}.$$

There exist $v_0 \in S, v_0' \in S'$,

$$\mathrm{AGG}(\{\mathrm{GNN}(u, \mathbf{A}^{(S-\{v_0\})}) | u \in S\}) = \mathrm{AGG}(\{\mathrm{GNN}(u', \mathbf{A}^{(S'-\{v_0'\})}) | u' \in S'\}).$$

Similarly, there exists $u_0' \in S'$

$$\mathrm{GNN}(v_0, \mathbf{A}^{(S-\{v_0\})}) = \mathrm{GNN}(u_0', \mathbf{A}^{(S'-\{v_0'\})}).$$

As GNN is node most expressive,

$$\exists \pi, \pi(v_0) = u_0', \pi(\mathbf{A}) = \mathbf{A'}, \pi(\mathbf{L}(S - \{v_0\}, \mathbf{A})) = \mathbf{L}(S' - \{v_0'\}, \mathbf{A'})).$$

Therefore, $\pi(S - \{v_0\}) = S' - \{v_0'\}$. Note that $v_0 \notin S - \{v_0\}$, so $u_0' = \pi(v_0) \notin S' - \{v_0'\}$, while $u_0' \in S'$, therefore $u_0' = v_0'$.

Therefore, $\pi(S) = S'$, and $\pi(\mathbf{A}) = \mathbf{A'}$, so $(S, \mathbf{A}) \simeq (S', \mathbf{A'})$. ■

A.9 Proof of Theorem 26

We prove it by contradiction: If $\exists v_0 \in S, v_0' \in S'$,

$$\mathrm{GNN}(S, \mathbf{A}^{(S-\{v_0\})}) = \mathrm{GNN}(S', \mathbf{A'}^{(S'-\{v_0'\})})$$

Therefore, there exists $u_0 \in S, u_0' \in S'$

$$\mathrm{GNN}(v_0, \mathbf{A}^{(S-\{v_0\})}) = \mathrm{GNN}(u_0', \mathbf{A}^{(S'-\{v_0'\})}).$$

As GNN is node most expressive,

$$\exists \pi, \pi(v_0) = u_0', \pi(\mathbf{A}) = \mathbf{A'}, \pi(\mathbf{L}(S - \{v_0\}, \mathbf{A})) = \mathbf{L}(S' - \{v_0'\}, \mathbf{A'})).$$

Therefore, $\pi(S - \{v_0\}) = S' - \{v_0'\}$. Note that $v_0 \notin S - \{v_0\}$, so $u_0' = \pi(v_0) \notin S' - \{v_0'\}$, while $u_0' \in S'$, therefore $u_0' = v_0'$.

Therefore, $\pi(S) = S'$, and $\pi(\mathbf{A}) = \mathbf{A'}$, so $(S, \mathbf{A}) \simeq (S', \mathbf{A'})$, which contradicts to that $(S, \mathbf{A}) \not\simeq (S', \mathbf{A'})$.

A.10 Proof of Proposition 27

Figure 6 provides an example.

A.11 Proof of Proposition 28

Figure 5 and Figure 6 provide example.

A.12 Proof of Proposition 18

Due to the property 1 in Definition 16, $\mathbf{L}(S, \mathbf{A}) = \pi(\mathbf{L}(S', \mathbf{A'})) \Rightarrow S = \pi(S')$. Therefore, for all $v \in S$, $\pi^{-1}(v) \in S$. Moreover, $\forall v' \in S'$, $\exists v \in S, \pi^{-1}(v) = v'$.

Consider an edge (u, v) in $\mathcal{H}_S$. According to Definition 17, $u \neq v$,$u \leq_S v$, and there exists no node $w \in S, w \notin u, v$ that $u \leq_S w$ and $w \leq_S v$. As $\pi(S') = S$, $\pi^{-1}(u) \neq \pi^{-1}(v)$,$\pi^{-1}(u) \leq_{S'} \pi^{-1}(v)$, and there exists no node $\pi^{-1}(w) \in S', \pi^{-1}(w) \notin \pi^{-1}(u), \pi^{-1}(v)$ that $\pi^{-1}(u) \leq_{S'} \pi^{-1}(w)$ and $\pi^{-1}(w) \leq_{S'} \pi^{-1}(v)$. Therefore, when $S = \pi(S')$, for all edge (u, v) in $\mathcal{H}_S$, edge $(\pi^{-1}(u), \pi^{-1}(v))$ exists in $\mathcal{H}_{S'}$.

Similarly, as $S' = \pi^{-1}(S)$, for all edge $(\pi^{-1}(u), \pi^{-1}(v))$ in $\mathcal{H}_{S'}$, edge

$$((\pi^{-1})^{-1}(\pi^{-1}(u)), (\pi^{-1})^{-1}(\pi^{-1}(v))) = (u, v),$$

exists in $\mathcal{H}_{S'}$. So $\mathcal{H}_S = \pi(\mathcal{H}_{S'})$. Equivalently, for all $v \in S'$, $\pi(v)$ is in S, and $(\{v\}, \mathcal{H}_{S'}) \simeq (\{\pi(v)\}, \mathcal{H}_S)$.

Assume that u, v are not isomorphic in S, but $\mathbf{L}(S, \mathbf{A})_{u,u,:} = \mathbf{L}(S, \mathbf{A})_{v,v,:}$. Define permutation $\pi : V \to V$ as follows,

$$\pi(i) = \begin{cases} v & \text{if } i = u \\ u & \text{if } i = v \\ i & \text{otherwise} \end{cases}.$$

$\pi(\mathbf{L}(S, \mathbf{A})) = \mathbf{L}(S, \mathbf{A}) \Rightarrow \pi(S) = S \Rightarrow (v, \mathcal{H}_S) \simeq (u, \mathcal{H}_S)$. Equivalently, non-isomorphic nodes in the same hasse diagram should have different labels.

A.13 Proof of Theorem 39

The main gap between hypergraph isomorphism and corresponding graph isomorphism is that hypergraph permutation is composed of two permutation transforms node and edge order independently, while corresponding graph isomorphism is only related to one node permutation, so we first define ways to combine and split permutations.

Sorting of corresponding graph: Let $I_V(IG_H) = \{i | (IG_H)_{i,i,d+1} = 1\}$ denote nodes in $G(H)$ corresponding to nodes in H. Let $I_E(IG_H) = \{i | (IG_H)_{i,i,d+1} = 0\}$ denote the nodes representing hypergraph edges. We define a permutation $\pi^{I_V, I_E} \in \Pi_{n+m}$, π^{I_V, I_E}, $\pi^{I_V, I_E}(I_V) = [n], \pi^{I_V, I_E}(I_E) = \{n+1, n+2, ..., n+m\}$.

Concatenation of permutation: Let $\pi_1 \in \Pi_n, \pi_2 \in \Pi_m$. Their concatenation $\pi_1 \| \pi_2 \in \Pi_{m+n}$

$$\pi_1 \| \pi_2(i) = \begin{cases} \pi_1(i) & i \leq n \\ n + \pi_2(i - n) & \text{otherwise} \end{cases}$$

When S_1, S_2 have different sizes, or H_1, H_2 have different number of nodes or hyperedges, two poset are non-isomorphic. So we only discuss the case that the poset and hypergraph sizes are the same. Let n, m denote the number of nodes and hyperedges in the hypergraph. Then the corresponding graph has $n + m$ nodes.

We first prove $\Rightarrow$: When $(S, H) \sim (S', H')$, according to Definition 37, there exists $\pi_1 \in \Pi_n, \pi_2 \in \Pi_m, (\pi_1, \pi_2)(H) = H', \pi_1(S) = S'$. Then, $(\pi_1 \| \pi_2)(IG_H) = IG_{H'}$ and $(\pi_1 \| \pi_2)(S) = S'$.

Then we prove $\Leftarrow$: When $(S, IG_H) \simeq (S', IG_{H'})$. We can first sort two incidence graph. Let $\pi = \pi^{I_V(IG_H), I_E(IG_H)}$ and $\pi' = \pi^{I_V(IG_{H'}), I_E(IG_{H'})}$. Then two posets and graphs are still isomorphic.

$$(\pi(S), \pi(IG_H)) \simeq (\pi'(S'), \pi'(IG_{H'}))$$

Therefore, $\exists \pi_0 \in \Pi_{n+m}, \pi(S) = \pi_0(\pi'(S')), \pi(IG_H) = \pi_0(\pi'(IG_{H'}))$. Let $\mathbf{A}, \mathbf{A'} \in \mathbb{R}^{(n+m)\times(n+m)\times d+1}$ denote the adjacency tensor of $\pi(IG_H), \pi'(IG_{H'})$ respectively. Therefore,

$$\mathbf{A} = \pi_0(\mathbf{A'}) \Rightarrow \mathbf{A}_{\pi_0(i), \pi_0(i), d+1} = \mathbf{A'}_{i,i,d+1}, \forall i \in \{1, 2, ..., m+n\}.$$

As the nodes in $\mathbf{A}, \mathbf{A'}$ are sorted, $\mathbf{A}_{i,i,d+1} = 1, \mathbf{A'}_{i,i,d+1} = 1$ if $i \leq n$, and $\mathbf{A}_{i,i,d+1} = 0, \mathbf{A'}_{i,i,d+1} = 0$ if $i > n$. Therefore, π_0 maps $\{1, 2, ..., n\}$ to $\{1, 2, ..., n\}$ and $\{n+1, n+2, ..., n+m\}$ to

$\{n+1, n+2, ..., n+m\}$. Therefore, we can decompose π_0 into two permutation π_1, π_2.

$$\pi_1(i) = \pi_0(i), i \in \{1, 2, ..., n\}$$
$$\pi_2(i) = \pi_0(i+n) - n, i \in \{1, 2, ..., m\}$$

Then, $S = \pi_1(S')$ and $H = (\pi_1, \pi_2)(H')$.

A.14 Proof for Section 7

We first define some notations

Isomorphism type of node tuple k, l-WL and k-WL use the isomorphism type of tuple to initialize colors, which is defined as follows:

Given graphs $G^1 = (V^1, \mathbf{A}^1), G^2 = (V^2, \mathbf{A}^2)$ and k-tuples S^1, S^2 in G^1, G^2 respectively. S^1, S^2 have the same isomorphism type iff

1. $\forall i_1, i_2 \in [k]$, $S^1_{i_1} = S^1_{i_2} \leftrightarrow \mathbf{S}^2_{i_1} = S^2_{i_2}$.
2. $\forall i, j \in [k], \mathbf{A}^1_{S^1_i S^1_j} = \mathbf{A}^2_{S^2_i S^2_j}$.

A.14.1 Expressivity comparison

Given two function f, g, f can be expressed by g means that there exists a function ϕ that $\phi \circ g = f$, which is equivalent to given arbitrary input H, G, $f(H) = f(G) \Rightarrow g(H) = g(G)$. We use $f \to g$ to denote that f can be expressed with g. If both $f \to g$ and $g \to f$, there exists a bijective mapping between the output of f to the output of g, denoted as $f \leftrightarrow g$.

Here are some basic rule.

- $g \to h \Rightarrow f \circ g \to f \circ h$.
- $g \to h, f \to s \Rightarrow f \circ g \to s \circ h$.
- f is bijective, $f \circ g \to g$

A.14.2 Proof of Proposition 32

The graph color of k-WL with l-pooling is

$$c_k^{(l)}(G)\text{Hash}(\{\{\text{Hash}(\{\{c_k(S\|S', G)|S' \in V(G)^{k-l}\}\})|S \in V(G)^l\}\})$$

The graph color of k-WL with is

$$c_k(G) = \text{Hash}(\{\{c_k(S, G)|S \in V(G)^k\}\}).$$

$$c_k^{(l)}(G) \to \{\{c_k(S, G)|S \in V(G)^k\}\} \to c_k(G)$$

Moreover, as

$$\begin{aligned} c_k(S\|S', G) &\to \{\{c_k(S\|\phi_0(S', v), G)|v \in V(G)\}\} \\ &\to \{\{c_k(S\|\phi_2(\phi_1(S', v_1), v_2), G)|v_1, v_2 \in V(G)\}\} \\ &\to ... \to \{\{c_k(S\|S')|S' \in V(G)^{k-l}\}\} \end{aligned}$$

	Data	BaseGNN	#layer	hiddim	bs	lr	#hop
Table 1	PB, Ecoli	GIN	3	32	32	1e-4	2
	Others	GIN	3	32	32	1e-4	1
Table 2	collab	GIN	3	256	32	1e-4	1
	ddi	GIN	3	96	32	1e-4	1
	citation2	GIN	3	32	32	1e-4	1
	ppa	GIN	3	32	32	1e-4	1
Table 3	All	GIN	3	32	48	3e-3	-1
Table 6	NDC-s, Email-Eu	max	4	64	96	5e-3	-1
	Others	max	4	64	96	4e-3	-1
Table 5	All	GIN	1	64	64	1e-3	-1

Table 6: Hyperparameters for our models. BaseGNN: GNN used to encoding graph and labels, max means using max aggregator. #layer: the number of GNN layers. hiddim: hidden dimension. bs: batch size, lr: learning rate. #hop: the number of hops for sampling subgraph, -1 means using whole graph.

Therefore,

$$\begin{aligned} c_k(G) &\to \{\{c_k(S\|S'', G)|S \in V(G)^k, S'' \in V(G)^{k-l}\}\} \\ &\to \{\{\{\{c_k(S\|S', G)|S' \in V(G)^{k-l}\}\}S \in V(G)^k, S'' \in V(G)^{k-l}\}\}\} \\ &\to \{\{c_k^{(l)}(G)|S'' \in V(G)^{k-l}\}\} \to c_k^{(l)}(G) \end{aligned}$$

Appendix B. Experimental settings

Computing infrastructure. We leverage Pytorch Geometric and Pytorch for model development. All our models run on an Nvidia 3090 GPU on a Linux server.

Hyperparameters We use Adam optimizer and constant learning rate for all our models. Main hyperparameters for our models are listed in Table 6. More detailed configuration of each experiments is provided in our code.

Model Implementation. For undirected link prediction tasks, our implementation is based on the code of SEAL (Zhang and Chen, 2018), which segregates an ego subgraph from the whole graph for each link. For other tasks, our model runs on the whole graph. We use optuna to perform random search. Hyperparameters were selected to optimize scores on the validation sets.

Appendix C. More Details about the Datasets

C.1 Undirected Link Prediction

We use eight real-world datasets from SEAL (Zhang and Chen, 2018): USAir is a network of US Air lines. NS is a collaboration network of researchers. PB is a network of US political blogs. Power is an electrical grid of western US. Router is a router-level Internet.

Ecoli is a metabolic network in E.coli. Cele is a neural network of C.elegans. Yeast is a protein-protein interaction network in yeast.

We also use OGB datasets (Hu et al., 2020): `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 datasets in Table 7. More information about these datasets can be found in (Hu et al., 2020).

Table 7: Statistics and evaluation metrics of undirected link prediction datasets.

Dataset	**#Nodes**	**#Edges**	**Avg. node deg.**	**Split ratio**	**Metric**
USAir	332	2,126	12.81	0.85/0.05/0.10	auroc
NS	1,589	2,742	3.45	0.85/0.05/0.15	auroc
PB	1,222	16,714	27.36	0.85/0.05/0.15	auroc
Yeast	2,375	11,693	9.85	0.85/0.05/0.15	auroc
C.ele	297	2,148	14.46	0.85/0.05/0.15	auroc
Power	4,941	6,594	2.67	0.85/0.05/0.15	auroc
Router	5,022	6,258	2.49	0.85/0.05/0.15	auroc
E.coli	1,805	14,660	16.24	0.85/0.05/0.15	auroc
ogbl-ppa	576,289	30,326,273	105.25	fixed	Hits@100
ogbl-collab	235,868	1,285,465	10.90	fixed	Hits@50
ogbl-ddi	4,267	1,334,889	625.68	fixed	Hits@20
ogbl-citation2	2,927,963	30,561,187	20.88	fixed	MRR

C.2 Directed Link Prediction

We use the same settings and datasets as He et al. (2022). The task is to predict whether a directed link exists in a graph. Texas, Wisconsin, and Cornell consider websites as nodes and links between websites as edges. Cora-ML and CiteSeer are citation networks. Telegram is an influence graph between Telegram channels. Their statistics are shown in Table 8.

Table 8: Statistics and evaluation metrics of directed link prediction datasets.

Dataset	**#Nodes**	**#Edges**	**Avg. node deg.**	**Split ratio**	**Metric**
wisconsin	251	515	4.10	0.80/0.05/0.15	accuracy
cornell	183	298	3.26	0.80/0.05/0.15	accuracy
texas	183	325	3.55	0.80/0.05/0.15	accuracy
cora_ml	2,995	8,416	5.62	0.80/0.05/0.15	accuracy
telegram	245	8,912	72.75	0.80/0.05/0.15	accuracy
citeseer	3,312	4,715	2.85	0.80/0.05/0.15	accuracy

C.3 Hyperedge Prediction Datasets

We use the datasets and baselines in (Srinivasan et al., 2021). NDC-c (NDC-classes) and NDC-s (NDC-substances) are both drug networks. NDC-c takes each class label as a node and the set of labels applied to a drug as a hyperedge. NDC-s takes substances as nodes and the set of substances contained in a drug as a hyperedge. Tags-m (tags-math-sx) and tags-a (tags-ask-ubuntu) are from online Stack Exchange forums, where nodes are tags and hyperedges are sets of tags for the same questions. Email-En (email-Enron) and email-Eu are two email networks where each node is a email address and email hyperedge is the set of all addresses on an email. Congress (congress-bills) takes Congress members as nodes, and each hyperedge corresponds to the set of members in a committe or cosponsoring a bill. Their statistics are shown in Table 9.

Table 9: Statistics and evaluation metrics of directed link prediction datasets.

Dataset	**#Nodes**	**#Hyperdges**	**Split ratio**	**Metric**
NDC-c	6,402	1,048	5-fold	f1-score
NDC-s	49,886	6,265	5-fold	f1-score
tags-m	497,129	145,054	5-fold	f1-score
tags-a	591,904	169,260	5-fold	f1-score
email-En	4,495	1,458	5-fold	f1-score
email-EU	85,109	24,400	5-fold	f1-score
congress	732,300	83,106	5-fold	f1-score

C.4 Subgraph Prediction Tasks

Following (Wang and Zhang, 2022), we use three synthetic datasets: density, cut ratio, coreness. The task is to predict the corresponding properties of randomly selected subgraphs in random graphs. Their statistics are shown in Table 10.

Table 10: Statistics and evaluation metrics of directed link prediction datasets.

Dataset	**#Nodes**	**#Edges**	**#Subgraphs**	**Split ratio**	**Metric**
density	5,000	29,521	250	0.50/0.25/0.25	f1-score
cut-ratio	5,000	83,969	250	0.50/0.25/0.25	f1-score
coreness	5,000	118,785	221	0.50/0.25/0.25	f1-score

Appendix D. Time and GPU Memory in Link Prediction Task

To illustrate the scalability of GNNs, we measure the time and GPU memory consumption on ppa dataset. The process we measure including all precomputation and prediction a number of edges in one batch. The results are shown in Figure 7. For GNNs with labeling

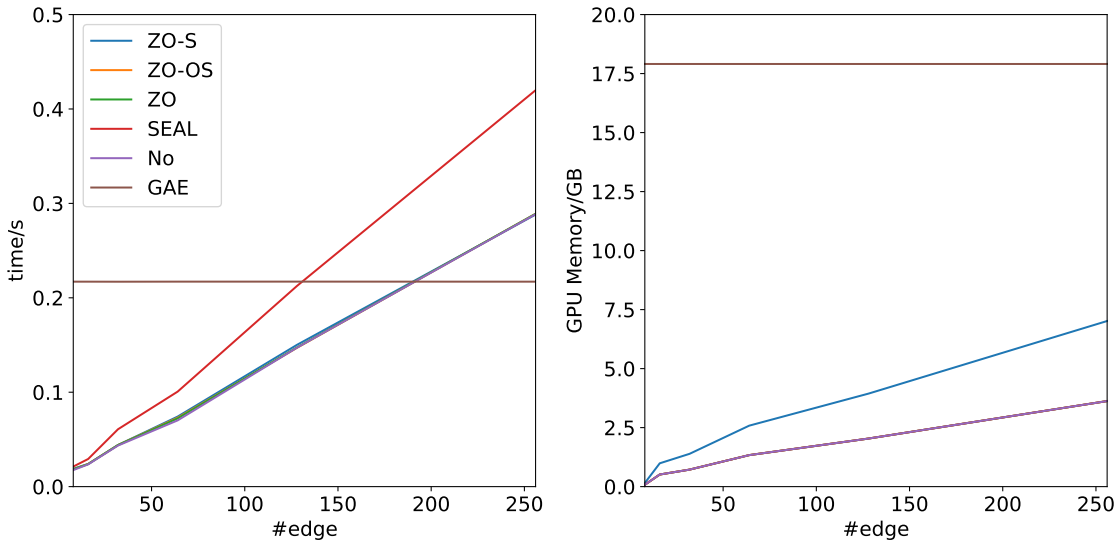

Figure 7: Time and GPU memory consumption for link prediction task on ppa dataset.

tricks (ZO-S, ZO-OS, ZO, SEAL) and GNN without labeling trick for ablation (No), they all have nearly the same time and memory consumption, as the only difference is integer label computation and one embedding layer for encoding labels. They all sample subgraphs from the whole graph and do not need to precompute embeddings for all nodes in the graph, so when the number of edges is small, the time and memory approaches 0. In contrast, GAE precomputes all nodes' embeddings, leading to large time and GPU consumption even for few edges. It has lower time and GPU consumption after the precomputation. For large real-world graphs, putting whole graphs into memory is impossible and thus sampling subgraphs is a must (even for GNNs without labeling trick), so labeling trick will not introduce a high extra cost.

References

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

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B. Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *KDD*, 2018.

Sameer Agarwal, Kristin Branson, and Serge J. Belongie. Higher order learning with graphs. In *ICML*, 2006.

Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *NeurIPS*, 2020.

Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *VLDB*, 2(1):754–765, 2009.

Waïss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *ICLR*, 2021.

László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *sfcs*, pages 39–46. IEEE, 1979.

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

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

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *ICLR*, 2022.

Alain Bretto. *Hypergraph Theory*. 2013.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014.

Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *NeurIPS*, 2019.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *NeurIPS*, 2020.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order, Second Edition*. 2002. ISBN 978-0-521-78451-1.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

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 *NeurIPS*, 2015.

Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. *NeurIPS*, 2022.

Fabrizio Frasca, Beatrice Bevilacqua, Michael M. Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. In *NeurIPS*, 2022.

Floris Geerts. The expressive power of kth-order invariant graph networks. *CoRR*, abs/2007.12035, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Yixuan He, Xitong Zhang, Junjie Huang, Mihai Cucuringu, and Gesine Reinert. Pytorch geometric signed directed: A survey and software on graph neural networks for signed and directed graphs. *arXiv preprint arXiv:2202.10793*, 2022.

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. *NeurIPS*, 2020.

Xingyue Huang, Miguel Romero, İsmail İlkan Ceylan, and Pablo Barceló. A theory of link prediction via relational weisfeiler-leman on knowledge graphs. In *NeurIPS*, 2023.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *NeurIPS*, 2021.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *NeurIPS*, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *ICLR*, 2016.

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.

Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess E. Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. 2023.

Yunyu Liu, Jianzhu Ma, and Pan Li. Neural predicting higher-order patterns in temporal networks. In *WWW*, 2022.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *ICLR*, 2019b.

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

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 *AAAI*, 2019.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016.

Omri Puny, Matan Atzmon, Edward J. Smith, Ishan Misra, Aditya Grover, Heli Ben-Hamu, and Yaron Lipman. Frame averaging for invariant and equivariant network design. In *ICLR*, 2022.

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.

Chendi Qian, Gaurav Rattan, Floris Geerts, Mathias Niepert, and Christopher Morris. Ordered subgraph aggregation networks. In *NeurIPS*, 2022.

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.

Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *ICLR*, 2020.

Balasubramaniam Srinivasan, Da Zheng, and George Karypis. Learning over families of sets-hypergraph representation learning for higher order tasks. In *SDM*, 2021.

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

Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, 2020.

Zekun Tong, Yuxuan Liang, Changsheng Sun, Xinke Li, David S. Rosenblum, and Andrew Lim. Digraph inception convolutional networks. In *NeurIPS*, 2020a.

Zekun Tong, Yuxuan Liang, Changsheng Sun, David S. Rosenblum, and Andrew Lim. Directed graph convolutional network. *CoRR*, abs/2004.13970, 2020b.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

Changlin Wan, Muhan Zhang, Wei Hao, Sha Cao, Pan Li, and Chi Zhang. Principled hyperedge prediction with structural spectral features and neural networks. *arXiv preprint arXiv:2106.04292*, 2021.

Sheng Wang, Emily R Flynn, and Russ B Altman. Gaussian embedding for large-scale gene set analysis. *Nature machine intelligence*, 2(7):387–395, 2020.

Xiyuan Wang and Muhan Zhang. GLASS: GNN with labeling tricks for subgraph representation learning. In *ICLR*, 2022.

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.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *ICML*, 2019.

Jiaxuan You, Jonathan Michael Gomes Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. 2021.

Bohang Zhang, Guhao Feng, Yiheng Du, Di He, and Liwei Wang. A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests. In *ICML*, 2023.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *NeurIPS*, 2018.

Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *ICLR*, 2020.

Muhan Zhang and Pan Li. Nested graph neural networks. In *NeurIPS*, 2021.

Muhan Zhang, Zhicheng Cui, Shali Jiang, and Yixin Chen. Beyond link prediction: Predicting hyperlinks in adjacency space. In *AAAI*, 2018a.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018b.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *NeurIPS*, 2021a.

Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. Magnet: A neural network for directed graphs. *NeurIPS*, 2021b.

Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew J. Hirn. Magnet: A neural network for directed graphs. In *NeurIPS*, 2021c.

Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *ICLR*, 2022.

Cai Zhou, Xiyuan Wang, and Muhan Zhang. From relational pooling to subgraph gnns: A universal framework for more expressive graph neural networks. In *ICML*, 2023.

Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal A. C. Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, 2021.