Rethinking Higher-order Representation Learning with Graph Neural Networks

Tuo Xu
Peking University
doujzc@stu.pku.edu.cn

Lei Zou
Peking University
zoulei@pku.edu.cn

Abstract

In the field of graph machine learning, graph neural networks (GNNs) are promising models for learning graph representations and node representations. However, many GNNs perform poorly on learning higher-order representations such as links due to their limited expressive power. Zhang et al. [42] summarize recent advances in link prediction and propose labeling trick as a common framework for learning node set representations with GNNs. However, their theory is limited to employing an ideally expressive GNN as the backbone, and can only justify a limited series of link prediction models. In this paper, we take a further step to study the expressive power of various higher-order representation learning methods. Our analysis begins with showing the inherent symmetry between node labeling and higher-order GNNs, which directly justifies node labeling methods and higher-order GNNs through a unified framework. Then, we study the utilization of MPNNs for computing representations in these methods, and show the expressive power upper bounds under these situations. After that, we provide a comprehensive analysis about how these previous methods surpass plain GNNs by showing their ability to capture *path* information. Finally, using the intuitions provided by the analysis, we propose an extremely simple method for link prediction tasks, which we believe could bring insights for designing more complicated and powerful models in the future.

1 Introduction

Graph neural networks (GNNs) are the dominant approaches for learning graph representations, and most of them follow the message passing mechanism, thus are also called message passing neural networks (MPNNs). MPNNs perform especially well for *node classification* tasks and *graph classification* tasks, but are less suitable for jointly predicting the properties of node tuples such as links, which is known as the problem of *higher-order representation learning* [26, 27]. Theoretically, MPNNs cannot express the complex dependencies between nodes and fail to capture simple patterns such as common neighbors or counting triangles [7, 21, 8].

There are three main classes of GNNs extensions designed for learning higher-order representations to overcome the deficiency of plain GNNs: *higher-order* GNNs, GNNs with *node labeling*, and *heuristics-enhanced* GNNs. *Higher-order GNNs* [26, 27, 3, 25, 30] match the higher-order WL (FWL) variants [34, 5, 9, 23]. Their architectures still follow the message passing paradigm, except that each message passing unit in k-order GNNs is a k-node tuple. While k-GNNs are highly expressive as k-WL (k-FWL) tests, they suffer from severe computation costs. For GNNs with *node labeling* [40, 42, 46, 37, 32, 19, 41, 33], there are two major variants focusing on link prediction, and representative methods are SEAL [40] and NBFNet [46]. Given the target link (u, v), SEAL adds additional labels to tag u and v differently from the other nodes. Then, SEAL runs a GNN on the induced graph to collect the node representations of u and v to obtain the link representation of (u, v). It is proved by [42] that this approach actually enables SEAL to learn structural link representations with an ideally expressive GNN, i.e., SEAL can output different representations for any two links that are not topologically identical in the graphs. However, we are still unclear about

T. Xu et al., Rethinking Higher-order Representation Learning with Graph Neural Networks. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

what patterns SEAL can capture when using a MPNN as its backbone GNN model, which is a more practical problem. Different from SEAL, NBFNet only adds additional labels to tag the source node u differently from the other nodes. Then, NBFNet runs a GNN on the induced graph and collects node representations for every node. NBFNet assumes that the node representation for a node w now represents the link (u, w). Although NBFNet is well-motivated by the Bellman-Ford algorithm, there is no theoretical justification for this specific class of node labeling techniques. *Heuristics-enhanced MPNNs* [18, 38, 6] combine MPNNs with heuristics methods such as Common Neighbors, Adamic-Adar [1], Resource Allocation [45], etc. Since there are plenty of choices, it is hard to provide a common and general analysis to the expressive power of heuristics-enhanced MPNNs. Overall, we find that only the labeling trick [42] in SEAL [40] was partly justified for its expressive power, and this motivates us to study the properties of the various higher-order representation learning methods.

In this paper, we systematically study the problem of designing expressive GNN extensions for learning higher-order representations, and propose a unified framework to explain the above classes of GNN extensions. Our theory mainly considers the connection between the first two GNN classes, i.e. higher-order GNNs and node labeling. First, we study the properties of node labeling methods and show their inherent symmetry with higher-order GNNs from a graph isomorphism perspective. Second, we take MPNNs into consideration and study the expressive power of the different node labeling / higher-order GNNs when we apply MPNNs as the backbone GNN models. Third, based on our findings, we further study the common advantages of various higher-order representation learning methods compared with the 1-WL test, and explain their strengths from a novel perspective of *paths*. This explanation also provides strong justification for various heuristics-enhanced models. With this observation we try to derive the simplest model that reserves the core components for link prediction, a common 2-order representation learning problem [43]. Our method shares similar spirits with SGC [35], and is realized with a pre-propagation procedure and a non-GNN predictor which can be implemented as a logistic regression or a MLP. The performance is comparable with state-of-the-art models. We believe our finding may provide intuitions for designing more efficient higher-order representation learning methods in the future.

2 Preliminary

In this section we describe the notations and some background of this paper. Note that the introduction of some definitions, such as k-WL tests and labeling trick, are slightly tuned to better fit in our theory framework.

2.1 Basic Concepts

Notations. We use $\{\}$ to denote sets and $\{\{\}\}$ to denote multisets. The cardinality of a (multi)set $\mathcal{S}$ is denoted as $|\mathcal{S}|$. The index set is denoted as $[n] = \{1, ..., n\}$. In the main body of this paper we consider simple undirected graphs $G = (\mathcal{V}, \mathcal{E})$ with node features $\boldsymbol{x}_u$ for each $u \in \mathcal{V}$ for clarity, but the theory can be equivalently applied to directed heterogeneous graphs (e.g., knowledge graphs). The nodes in a graph are indexed by $[|\mathcal{V}|]$. We denote the neighbors of a node $u \in \mathcal{V}$ as $\mathcal{N}(u) = \{v \in \mathcal{V} \mid (v, u) \in \mathcal{E}\}$ and denote its degree as $\deg(u) = |\mathcal{N}(u)|$. A node tuple is denoted as $\mathbf{u} = (u_1, u_2, ..., u_k)$ where $u_1, ..., u_k \in \mathcal{V}$. We express slices of $\mathbf{u}$ as $\mathbf{u}_{i:j} = (u_i, ..., u_j), \mathbf{u}_{i:} = (u_i, ..., u_k), \mathbf{u}_{:i} = (u_1, ..., u_i)$. A path $P = (u_0, u_1, u_2, ..., u_d)$ of G is a special node tuple where $(u_{i-1}, u_i) \in \mathcal{E}$ for all $i \in [d]$ and its length is $|P| = d$. P is simple if its nodes are distinct. We use $\boldsymbol{h}_u$ to represent the vector representation of some node u computed by GNNs. We denote the concatenation of vectors as $\boldsymbol{h} = [\boldsymbol{h}_1 \parallel \boldsymbol{h}_2 \parallel ...]$.

Isomorphism. An isomorphism is a structure-preserving mapping between two structures. We focus on graph isomorphism. Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$, a permutation $\pi : \mathcal{V}_G \to \mathcal{V}_H$ is an isomorphism from G to H if and only if $\forall u, v \in \mathcal{V}_G$, $(u, v) \in \mathcal{E}_G \iff (\pi(u), \pi(v)) \in \mathcal{E}_H$. In this case we also say G and H are isomorphic, denoted as $G \simeq H$. We define two nodes $u \in \mathcal{V}_G$, $v \in \mathcal{V}_H$ are isomorphic, denoted as $(u, G) \simeq (v, H)$, if and only if there is an isomorphism π such that $\pi(u) = v$. Extending the idea, we further define higher-order isomorphism, and denote two node tuples $\mathbf{u} = (u_1, ..., u_k)$ and $\mathbf{v} = (v_1, ..., v_k)$ being isomorphic as $(\mathbf{u}, G) \simeq (\mathbf{v}, H)$, if and only if there is an isomorphism π such that $\pi(u_i) = v_i$ for $i \in [k]$.

Message passing neural networks and Weisfeiler-Lehman tests. Message passing neural networks (MPNNs) are graph neural networks for learning node representations. Generally, they take

node features as inputs $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$ and execute

$$\boldsymbol{h}_u^{(l+1)} = \phi\left(\boldsymbol{h}_u^{(l)}, \psi\left(\{\boldsymbol{h}_v^{(l)} \mid v \in \mathcal{N}(u)\}\right)\right)$$

at iteration $l+1$, where ϕ is a *combine function*, and ψ is a *message function* that takes multisets as inputs. MPNNs implement ϕ and ψ as trainable functions. In the output iteration L, the representation $\boldsymbol{h}_u^{(L)}$ naturally expresses the node u, and we can obtain the representation for the entire graph by aggregating the node representations: $\boldsymbol{h}_G = \text{Readout}\left(\left\{\left\{\boldsymbol{h}_u^{(L)} \mid u \in \mathcal{V}\right\}\right\}\right)$.

The 1-Weisfeiler-Lehman (1-WL) [34] test, also known as the color refinement algorithm, is a principled algorithm for detecting graph isomorphism. It is exactly a MPNN as introduced above, but with infinite layers and all functions $\phi, \psi, \text{Readout}$ are injective. The iterations of the WL test end where the partition of the nodes no longer changes. MPNNs are at most as expressive as the WL test.

2.2 Learning higher-order representations

Node representations cannot express higher-order representations. GNNs are expert in learning graph and node representations. When learning representations for node tuples such as links, it might be intuitive to just aggregate the corresponding node representations [12, 16, 14]: for example in Graph Auto Encoders (GAE) [16], to predict the link (u, v), they first run a GNN to collect node representations $\boldsymbol{h}_u, \boldsymbol{h}_v$ for the nodes and compute the link representation as $\boldsymbol{h}_{uv} = f(\boldsymbol{h}_u, \boldsymbol{h}_v)$. However, just like we cannot obtain the joint distribution of two random variables by simply multiplying their marginal distributions, we cannot obtain feasible link representations in this way: GAE neglects the correlation between the nodes u, v in the target tuple (u, v) [42]. Consider the graph (a) in Figure 1, where all nodes are isomorphic with each other. Even if we apply a most powerful GNN model (which outputs different representations for non-isomorphic nodes) to (a), each node gets the same representation. Therefore, any two pairs of nodes also get the same representation and thus are indistinguishable. This is because when predicting a link (u, v), the node representations of u and v contain no information about the relative position between each other. Therefore, it's necessary to study how to design new models for learning higher-order representations. Note that the problem of *learning higher-order representations for predicting node-tuple properties* is different from *designing higher-order GNNs for learning graph representations*, which is also a widely studied problem. See Appendix G for more comparison.

Higher-order GNNs. A series of works [26, 27, 3, 22] improve the expressiveness of MPNNs by directly learning node tuple representations. These models generally follow the k-WL or k-Folklore WL (k-FWL) tests [5]. We focus on k-FWL here due to its superior expressiveness and efficiency, and the full discussion of k-WL and k-FWL is in Appendix D. Given a graph $G = (\mathcal{V}, \mathcal{E})$, a k-FWL test uses k-node tuples as update units, and its update procedure is exactly the same as 1-WL, except that we define the neighbor of a k-node tuple $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}^k$ to be $\mathcal{N}(\mathbf{u}) = \{\{\mathcal{N}_v(\mathbf{u}) \mid v \in \mathcal{V}\}\}$ where $\mathcal{N}_v(\mathbf{u}) = (\mathbf{u}_{1/v}, \mathbf{u}_{2/v}, ..., \mathbf{u}_{k/v})$ and $\mathbf{u}_{i/v} = (u_1, ..., u_{i-1}, v, u_{i+1}, ..., u_k)$ for $i \in [k]$. We can see that k-FWL is essentially a graph rewriting procedure: it *sets up a new structure G^k induced by G*, which is called a k-order super-graph, whose super-nodes correspond to k-tuples in G and has a separate definition of neighbors [44]. It is straightforward to apply plain GNNs on G^k without heavy modification [26, 27, 44]: we only need to encode the representations of neighbors as $\boldsymbol{h}_{\mathcal{N}_v(\mathbf{u})} = [\boldsymbol{h}_{\mathbf{u}_{1/v}} \parallel ... \parallel \boldsymbol{h}_{\mathbf{u}_{k/v}}]$. For example, a MPNN on G^k is implemented as

$$\boldsymbol{h}_{\mathbf{u}}^{(l+1)} = \phi\left(\boldsymbol{h}_{\mathbf{u}}^{(l)}, \psi\left(\left\{\left\{\left[\boldsymbol{h}_{\mathbf{u}_{1/v}}^{(l)} \parallel ... \parallel \boldsymbol{h}_{\mathbf{u}_{k/v}}^{(l)}\right] \mid v \in \mathcal{V}\right\}\right\}\right)\right).$$

Node labeling GNNs. Zhang et al. [42] summarize the previous link prediction approaches and propose a general framework for learning higher-order representations with node labeling. We restate the ordered node tuple version of the labeling trick here [32, 20]. Given a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ and the target tuple $\mathbf{v} = (v_1, v_2, ..., v_k)$, we stack a labeling vector $\boldsymbol{l}_u$ in the first dimensions of $\boldsymbol{x}_u$ to obtain $\hat{\boldsymbol{x}}_u = [\boldsymbol{l}_u \parallel \boldsymbol{x}_u]$ for each $u \in \mathcal{V}$ using some predefined mechanism $\boldsymbol{l}_u = L(u \mid \mathbf{v}, G)$. L should satisfy these properties: $\forall G, H, \mathbf{v} \in \mathcal{V}_G^k, \mathbf{w} \in \mathcal{V}_H^k, \pi : \mathcal{V}_G \to \mathcal{V}_H$,

1. $L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H)$ holds for all $u \in \mathcal{V}_G \Rightarrow \pi(\mathbf{v}) = \mathbf{w}$,
2. π is an isomorphism from $(\mathbf{v}, G)$ to $(\mathbf{w}, H) \Rightarrow \forall u \in \mathcal{V}_u, L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H)$.

We can see that the labeling trick aims to tag the target nodes differently from the rest of the nodes, and is also a graph rewriting method. With the augmented node features, we obtain the *node labeling induced graphs* $G_{(\mathbf{v})}, H_{(\mathbf{w})}$. Then, we have

$$\text{for } i = 1, ..., k, \ \ \left(v_i, G_{(\mathbf{v})}\right) \simeq \left(w_i, H_{(\mathbf{w})}\right) \quad \Longleftrightarrow \quad (\mathbf{v}, G) \simeq (\mathbf{w}, H). \tag{1}$$

The theory adequately justifies previous subgraph based link prediction methods including [40, 32] and provide a new method for learning higher-order representations with plain GNNs. However, it doesn't justify other link prediction methods that only partly use additional node labels including [37, 46]. Also, we find that the result in Eq. 1 rather redundant. In this paper, we refer to these label augmentation methods (including labeling trick and other labeling methods in [37, 46]) as *node labeling* methods.

Heuristic-enhanced GNNs. Recent methods [18, 6, 38] apply a different strategy for learning link representations. Given a graph $G = (\mathcal{V}, \mathcal{E})$ and the target link (u, v), they first run GNNs to collect node representations $\boldsymbol{h}_u, \boldsymbol{h}_v$. Then, they incorporate various heuristic methods such as Common Neighbor, Adamic-Adar [1], Resource Allocation [45], etc., into the decision process: $p(u, v) = \phi\left(\boldsymbol{h}_u, \boldsymbol{h}_v, \text{Heuristic}(u, v, G)\right)$, where Heuristic is some pre-defined heuristic method. Despite the concise design of these methods, it is hard to provide a strict theoretical description for the expressiveness of these methods due to the various choices of the heuristic methods.

Problem setup. Our analysis contains three parts. In Section 3, we first introduce a series of models, namely i-j-NLs, to summarize various higher-order representation learning methods such as labeling trick [42, 40, 32], variants of node labeling methods [46, 37, 20], k-IGN [26], Edge Transformer [3], etc., into a unified framework. We first assume an imaginary expressive GNN as the backbone for these methods to purely study the properties of these methods themselves from an isomorphism perspective without being affected by the limited expressiveness of MPNNs. In Section 4, we plug in MPNNs as the backbone GNN models and study the properties of i-j-NLs under such situations. In Section 5, based on the previous analysis we try to dissect out the core factor that extends MPNNs to surpass 1-WL for higher-order prediction tasks, which is the *awareness of paths* between the nodes within the target node tuple. This finding is consistent with the various designs of heuristic-enhanced GNNs [38, 18, 6]. Based on the findings we further demonstrate that with a simple modification we can improve the link prediction power of MPNNs, which we believe might bring inspirations for future studies.

3 On the symmetry between node labeling and higher-order GNNs

In this section, we focus on the relation between node labeling and higher-order GNNs. Given a graph $G = (\mathcal{V}, \mathcal{E})$, suppose we want to learn the tuple representation of a k-node tuple $\mathbf{u} = (u_1, ..., u_k)$. We already know that by applying labeling trick, we can first assign additional labels to $u_1, ..., u_k$ to obtain the induced graph $G_{(\mathbf{u})}$, then learn the node representations $u_1, ..., u_k$ separately and aggregate these representations as a surrogate of the target tuple representation. In this section, we tend to answer the following questions: Is there a more simple and general mechanism that underlies the labeling trick? Can higher-order GNNs also learn the structural information and distinguish all non-isomorphic k-node tuples? Can other node labeling variants such as the partial node labeling in NBFNet [46] also distinguish all non-isomorphic tuples? Answering the above questions is non-trivial if we treat them separately. In fact, they can be answered in a single, unified theoretical framework which reflects the *symmetry* between node labeling and higher-order GNNs.

3.1 A general unification of higher-order representation learning methods

In the following, we extensively employ a hybrid representation learning method to summarize both node labeling and higher-order GNNs, which is defined below.

Definition 1. (i-j-NLs.) Given a graph $G = (\mathcal{V}, \mathcal{E})$ and the target node tuple $\mathbf{u} = (u_1, ..., u_k)$. We define a method that hybridizes i-order node labeling and j-order GNNs for $\mathbf{u}$, denoted as i-j-NL (i-order **N**ode **L**abeling and j-order GNNs) where $i + j = k$, to be the following procedure for computing the representation of $\mathbf{u}$. This includes two steps.

- In the first step, we use the first i nodes of $\mathbf{u}$, i.e. $\mathbf{u}_{:i} = (u_1, ..., u_i)$, to label the nodes of the graph, leading to the induced graph $G_{(\mathbf{u}_{:i})}$ with additional node labels.

- In the second step, we apply a j-order GNN on $G_{(\mathbf{u}_{:i})}$: that is, if $j > 1$ we construct $G^j_{(\mathbf{u}_{:i})}$ as the j-order induced super-graph of $G_{(\mathbf{u}_{:i})}$, and then apply a GNN to learn the representations of super-nodes in $G^j_{(\mathbf{u}_{:i})}$.

The representation of the corresponding super-node $\mathbf{u}_{(i+1):}$ in $G^j_{(\mathbf{u}_{:i})}$ is the final representation for the tuple $\mathbf{u}$.

We can now describe various higher-order representation learning method in a unified framework. For example, suppose we want to learn the link representation of (u, v). Partial node labeling methods (NBFNet [46], ID-GNN [37], PGNN [36], etc.) assign labels to the source node u and then learn the representation of the tail node v, thus are corresponded to 1-1-NLs. Higher-order GNNs (Edge Transformers [3], etc.) directly utilize GNNs to learn the higher-order representations and thus are corresponded to 0-2-NLs (or 0-k-NLs). Variants of the subgraph GNNs [4, 30] are corresponded to k-1-NLs or k-2-NLs. The ordered labeling trick methods (SEAL [40], GraIL [32], etc.) assign labels to both u and v and then apply a GNN to learn node representations. They can be regarded as a 2-1-NL. Although 2-1-NL actually learns 3-tuple representations, we can obtain the corresponding 2-tuple representations by simply pooling the 3-tuple representations: the representation of $\mathbf{u} = (u, v)$ is obtained by aggregating all representations of $\{\{(u, v, w) \mid w \in \mathcal{V}\}\}$. Therefore, to study the properties of various different higher-order representation learning methods, we can instead only focus on the i-j-NLs with varying i and j. More importantly, with i-j-NLs we can *go beyond* previous higher-order representation learning methods: We can not only unify them into a single framework, but also study more general and flexible situations by simply varying i and j.

The base GNN model in i-j-NLs for computing representations is of vital importance for discussing their expressiveness. In this paper we consider two situations: a *most-expressive* GNN and a *MPNN*. A most-expressive GNN always maps non-isomorphic nodes (super-nodes) to different representations, therefore we can purely focus on the functionalities of i-j-NLs themselves from an isomorphism perspective. [1] We denote i-j-NLs with most-expressive GNNs as i-j-$\mathrm{NL_E}$ (i-order **N**ode **L**abeling and j-order **E**xpressive GNNs), and denote i-j-NLs with MPNNs as i-j-$\mathrm{NL_{MP}}$ (i-order **N**ode **L**abeling and j-order **M**essage **P**assing neural networks) for notation clarity.

3.2 The properties of i-j-$\mathrm{NL_E}$

We begin with assuming $k = i + j$ to be a fixed value, which gives a consistent analysis of different i-j-$\mathrm{NL_E}s$ that learn the tuples of the same order k. All proofs are provided in the Appendix A.

Theorem 2. (The symmetry of i-j-$\mathrm{NL_E}$.) *Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$, and let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}^k_G, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}^k_H$ where k is a fixed value to be the target tuples we want to learn from G and H respectively. Then, the following statements are equivalent for any i, j satisfying $i \geq 0, j > 0$[2],$i + j = k$:*

- *A i-j-$\mathrm{NL_E}$ model gives $\mathbf{u}$ and $\mathbf{v}$ the same representation.*
- $(\mathbf{u}, G) \simeq (\mathbf{v}, H)$.

Theorem 2 directly points out the invariant property of i-j-$\mathrm{NL_E}$: they are all able to distinguish any non-isomorphic $(i + j)$-order tuples. Moreover, it states that all i-j-$\mathrm{NL_E}$ satisfying $i + j = k$ are equivalent in expressive power, which implies the symmetry between node labeling (corresponding to i) and higher-order GNNs (corresponding to j). Continuing from Theorem 2, we can deduce more general results about i-j-$\mathrm{NL_E}$.

Proposition 3. *Suppose $i, p \geq 0, j, q > 0$ are integers. A p-q-$\mathrm{NL_E}$ can distinguish any non-isomorphic node tuples that a i-j-$\mathrm{NL_E}$ can distinguish, if and only if $p + q \geq i + j$.*

Recall that if $p+q > i+j$, we can always pool the $(p+q)$-order representations learnt by a p-q-$\mathrm{NL_E}$ to obtain the corresponding $(i + j)$-order representations learnt by a i-j-$\mathrm{NL_E}$. With Theorem 2 and Corollary 3 we establish a general connection between all previous node labeling / higher-order

[1] Even a most-expressive GNN cannot learn higher-order representations for node tuples. Since we view node labeling and k-GNNs as *graph rewriting* techniques, using a most-expressive GNN is equivalent to asking the question: *Whether the nodes in the induced super-graphs contain enough information for identifying higher-order graph patterns.*

[2] We limit j to be strictly larger than 0 only because there are no "0-order" GNNs.

GNNs that can be expressed as i-j-NL_{E}. It is now straightforward to answer the above questions at the beginning of Section 3: with a most-expressive GNN, a k-order GNN distinguishes all non-isomorphic k-order tuples; NBFNet distinguishes all non-isomorphic links. We can also deduce a more concise result for the labeling trick:

Corollary 4. (Labeling trick.) *Given the conditions in Theorem 2 hold. Then, we have*

$$G_{(\mathbf{u})} \simeq H_{(\mathbf{v})} \iff (\mathbf{u}, G) \simeq (\mathbf{v}, H),$$

where $G_{(\mathbf{u})}, H_{(\mathbf{v})}$ are node labeling induced graphs by labeling $\mathbf{u}$ and $\mathbf{v}$ respectively.

The above theories provide strong justifications for previous node labeling and higher-order GNNs. Corollary 4 also provides a more concise version of the labeling trick. Together, we establish a clear picture describing the capabilities and limitations of general higher-order representation learning methods when equipped with most-expressive GNNs.

4 Message passing breaks the symmetry

In this section, we investigate the expressiveness of i-j-NLs from a more practical view, i.e., when we apply MPNNs as the backbone GNN models. As we shall see later, the incorporation of MPNNs actually breaks the symmetry between node labeling and higher-order GNNs. Without loss of generality, we assume the node labeling mechanism we applied to be zero-one node labeling and its extensions [40, 19, 42], and the full discussion about different labeling methods is in Appendix E.

4.1 The monotonicity of i-j-$\mathrm{NL}_{\mathrm{MP}}$

With the results in Section 3 one may assume that all i-j-$\mathrm{NL}_{\mathrm{MP}}$ with the same $k = i + j$ are still equivalent in their expressive power. However, this is not the case. In fact, although it's obvious that the expressive power upper bound of 0-k-$\mathrm{NL}_{\mathrm{MP}}$ is the k-FWL test, this by no means tells that $(k-1)$-1-$\mathrm{NL}_{\mathrm{MP}}$ can also reach the k-FWL expressive power. We still begin with assuming $k = i + j$ to be a fixed value, and summarize the expressive power of i-j-$\mathrm{NL}_{\mathrm{MP}}$ as follows. All proofs are in Appendix B.

Theorem 5. *Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$, and let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$ where k is a fixed value to be the target tuples we want to learn from G and H respectively. Then, for any i, j satisfying $i > 0, j > 0, i + j = k$, the expressive power of $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$ is strictly higher than i-j-$\mathrm{NL}_{\mathrm{MP}}$, that is:*

- *There is a i-j-$\mathrm{NL}_{\mathrm{MP}}$ that distinguishes $\mathbf{u}$ and $\mathbf{v} \Longrightarrow$ There is a $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$ that distinguishes $\mathbf{u}$ and $\mathbf{v}$.*
- *There exists $G, H, \mathbf{u}, \mathbf{v}$ such that a $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$ distinguishes $\mathbf{u}$ and $\mathbf{v}$ but i-j-$\mathrm{NL}_{\mathrm{MP}}$ cannot.*

Theorem 5 establishes a strict order of the expressive powers of different i-j-$\mathrm{NL}_{\mathrm{MP}}$ given fixed $k = i + j$: The expressive power grows monotonically w.r.t. j, and is bounded by the k-FWL test. Continuing from Theorem 5, we can obtain the following more general results.

Proposition 6. *Suppose $i, p \geq 0, j, q > 0$ are integers. There is a p-q-$\mathrm{NL}_{\mathrm{MP}}$ that distinguishes any non-isomorphic node tuples that a i-j-$\mathrm{NL}_{\mathrm{MP}}$ can distinguish if and only if $p + q \geq i + j$ and $q \geq j$. Otherwise, there exists non-isomorphic node tuples that a p-q-$\mathrm{NL}_{\mathrm{MP}}$ cannot distinguish but a i-j-$\mathrm{NL}_{\mathrm{MP}}$ can distinguish.*

Theorem 5 together with Proposition 6 establishes a general connection between i-j-$\mathrm{NL}_{\mathrm{MP}}$. This also implies that with MPNNs as the backbone GNN models, SEAL [40] is actually more expressive than NBFNet [46]. Edge Transformers [3] are also more expressive than NBFNet. However, when it comes to comparing SEAL and Edge Transformers, Proposition 6 indicates that each model has its advantages: examples are provided in Appendix B.

No free lunch: The principle behind the expressive powers. A more interesting phenomenon is revealed when we check the algorithm complexities of i-j-$\mathrm{NL}_{\mathrm{MP}}$. Suppose we are given a graph G with N nodes. Then, using a i-j-$\mathrm{NL}_{\mathrm{MP}}$ to evaluate every node tuple of G takes $\mathcal{O}\left(N^{i+j+1}\right)$ time and $\mathcal{O}\left(N^{j}\right)$ space. (We assume $j > 1$ for simplicity. The complexities are w.r.t. N. See Appendix C for more discussion.) Theorem 5 and Proposition 6 can be explained with the *no free lunch theorem*:

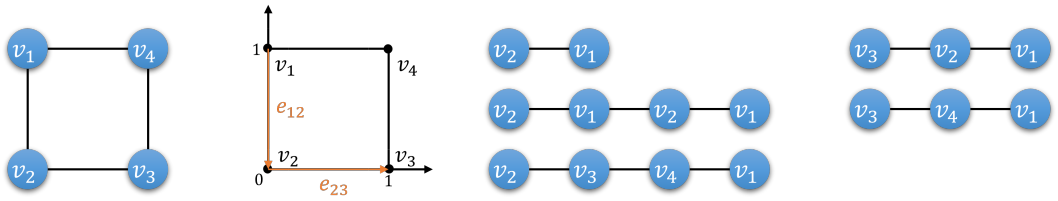


Figure 1: An illustration of how MPNNs with node labeling surpass vanilla 1-WL. (a) is the original graph with nodes colored by 1-WL. (b) indicates that the correlation of two nodes in Cartesian coordinate can be expressed by the paths between them. (c) contains (part of) the paths from v_2 to v_1 with lengths less than 4. (d) contains the paths from v_3 to v_1 with lengths less than 4.

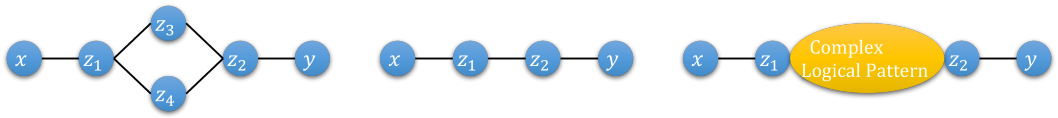


Figure 2: An illustration of what logical patterns 1-1-NL_{MP} can and cannot capture. (a) cannot be captured by 1-1-NL_{MP}. More generally, logical patterns in the forms of (c) cannot be captured by 1-1-NL_{MP}. (b) can be captured by 1-1-NL_{MP}. The reason is that logical patterns in (a) or more generally (c) are defined in the form of $\varphi(x,y) := \exists z_1, z_2(E(x,z_1) \wedge \varphi'(z_1,z_2) \wedge E(z_2,y))$. 1-1-$\text{NL}_{\text{MP}}$ can only capture single-source logical patterns defined by $\varphi(x,y) := \exists z(\varphi'(x,z) \wedge E(z,y)))$.

if a p-q-NL_{MP} takes either less time or space complexity compared with another one i-j-NL_{MP}, this is not free: it must also lose some distinguishing power. In turn, if both the time and the space complexities of p-q-NL_{MP} are larger than that of i-j-NL_{MP}, then the p-q-NL_{MP} is strictly more expressive than i-j-NL_{MP}.

4.2 The logical expressiveness of i-j-NL_{MP}

After discussing the limitations of NL_{MP}, here we prove a better understanding about what patterns i-j-NL_{MP} can and cannot learn. Our description is closely connected with FOC_k, a special fragment of the First-Order Logic. Each formula in FOC_k contains at most k variables. In addition, FOC_k allows the use of counting quantifier $\exists^N$, which expresses *at least N different nodes*, with arbitrary positive integer N. An example of the FOC_2 formulas would be: $\varphi(x) := \exists y(\text{E}(x,y) \wedge \varphi'(y))$,$\varphi'(x) := \exists y(\text{E}(x,y) \wedge \text{Red}(y))$. $\varphi(x)$ expresses that x has a 2-hop Red neighbor, and it is proved [2, 10] that a MPNN with global readout can capture such FOC_2 formulas. Similarly, we define $\text{FOC}_{k+1,i}$ to be a fragment of FOC_{k+1}, where we further restrict that the first i variables of all predicates in the same formula to be the same. For example, $\varphi(x,y,z) := \exists u, v(P_1(x,u,v) \wedge P_2(x,u,v,z))$ is a $\text{FOC}_{4,1}$ but not $\text{FOC}_{4,2}$ because only the first variable x is the same. The relation between $\text{FOC}_{k+1,i}$ and i-j-NL_{MP} is summarized below. (Assume $j > 1$ for simplicity. More discussion at Appendix H.)

Proposition 7. *For any k-tuples $\mathbf{u}, \mathbf{v}$ and $i + j = k$: There is a i-j-NL_{MP} that distinguishes $\mathbf{u}$ and $\mathbf{v}$ $\iff$ There is a $\text{FOC}_{k+1,i}$ formula that distinguishes $\mathbf{u}$ and $\mathbf{v}$.*

A direct result is that the i-j-NL_{MP} can no longer model complex correlations between *any* k-tuples in G. More specifically, we illustrate our ideas with 1-1-MP_{NL}. Consider the logic formula $\varphi(x,y) := \exists z_1, z_2(\text{E}(x,z_1) \wedge \text{E}(z_2,y) \wedge \exists^2 z_3(\text{E}(z_1,z_3) \wedge \text{E}(z_3,z_2))$, which can be expressed as Figure 2 (a). φ is expressible in FOC_3, and thus can be captured by 0-2-NL_{MP} or 2-FWL. However, it cannot be captured by 1-1-NL_{MP}, because to express $\varphi(x,y)$, one need to first define the correlation between z_1, z_2 with another logic classifier which is complex. In turn, 1-1-NL_{MP} can capture Figure 2 (b), since it can be expressed in a single-source manner $\varphi(x,y) := \exists z_2(\text{E}(z_2,y) \wedge (\exists z_1(\text{E}(z_1,z_2) \wedge \exists x(\text{E}(x,z_1)))))$, where the correlations between any intermediate node pairs are simple: only edges E. In general, we summarize that all logical patterns captured by 1-1-NL_{MP} must can be described in a single-source manner, and logical patterns like Figure 2 (c) cannot be captured by 1-1-NL_{MP}.

5 How do $\mathrm{NL_{MP}}$s surpass 1-WL?

From the discussions in Sections 3 and 4, it is evident that designing more expressive GNNs for learning higher-order representations can be achieved by setting large values for i and j in i-j-$\mathrm{NL_{MP}}$. However, this approach also results in higher space and time complexities. In this section, we take the opposite direction and attempt to identify the core property that enables GNNs to perform well on higher-order representation problems without being overly burdensome. Having observed that plain MPNNs, whose expressive power is limited by the 1-WL test, perform poorly on higher-order representation learning tasks, but even the relatively weak 1-1-$\mathrm{MP_{NL}}$ variants [46, 37] perform well on link prediction tasks, we try to find out the *common properties* that i-j-$\mathrm{NL_{MP}}$ methods possess while the 1-WL test lacks.

5.1 i-j-$\mathrm{NL_{MP}}$ encodes path information

In Section 2 we showed that the plain GNNs cannot learn higher-order representations because it cannot capture the correlation between the nodes in the target node tuple. For example in the graph (a) of Figure 1, where all nodes are isomorphic. Plain GNNs fail to distinguish (v_1, v_2) between (v_1, v_3) because all nodes share the same representation, and therefore aggregating two nodes' representations always produces the same result. This indicates that when predicting different links $(v_1, v_2), (v_1, v_3)$, plain GNNs do not capture the correlations between (v_1, v_2) and (v_1, v_3) respectively. So how do we model the correlations between the target nodes? Intuitively, in Euclidean Space the correlations can be expressed by relative positions: consider Figure 1 (b). The correlation between the adjacent nodes v_1, v_2 can be expressed with $e_{12} = [0, -1]$ which is the vector from v_1 to v_2. Similarly, the correlation between v_1, v_3 is expressed with $e_{12} + e_{23} = [1, -1]$ which is the concatenation of two edges. Thus we distinguish (v_1, v_2) from (v_1, v_3) in this way. Following this intuition, in graphs, *edges* express the relation between two adjacent nodes. Can we assume that the correlation between two arbitrary nodes can also be expressed by the concatenation of edges, i.e. *paths*? In Figure 1 (c) we list all paths from v_1 to v_2 with lengths less than 4. Figure 1 (d) lists all paths from v_1 to v_3 with lengths less than 4. We can also distinguish (v_1, v_2) from (v_1, v_3) in this way. A following question is, do i-j-$\mathrm{NL_{MP}}$ methods also distinguish paths and surpass the 1-WL in this way? The question is answered in the proposition below.

Proposition 8. *Given a graph $G = (\mathcal{V}_G, \mathcal{V}_E)$, we run 1-WL to assign node colors for G, denoted as $\mathrm{Col}(\cdot)$. The color of a path $P = (w_1, ..., w_d)$ is obtained by hashing the corresponding node color sequence $\mathrm{Col}(P) = \mathrm{Hash}(\mathrm{Col}(w_1), ..., \mathrm{Col}(w_d))$. Then, the correlation between two nodes (u, v), as discussed above, is expressed by all paths between u, v,*

$$\mathrm{Corr}(u, v) = \mathrm{Hash}\left(\{\{\mathrm{Col}(P) \mid P \in \mathrm{Paths}(u, v)\}\}\right).$$

Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$. Let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$ be the target node tuples. If there exists $1 \leq p, q \leq k$ such that $\mathrm{Corr}(u_p, u_q) \neq \mathrm{Corr}(v_p, v_q)$, then any i-j-$\mathrm{NL_{MP}}$ with $i + j \geq k$ with sufficient injective layers always distinguishes $\mathbf{u}$ and $\mathbf{v}$.

With Proposition 8 it is straightforward to see that i-j-$\mathrm{NL_{MP}}$ methods can capture path information between target nodes. More interestingly, various heuristic-enhanced GNNs also encodes path information. For example, traditional link prediction methods CN, AA [1], and RA [45] utilize 2-hop path information between two nodes. More recently, Neo-GNN [38], ELPH and BUDDY [6] count the number of multi-hop common neighbors of target nodes, and thus are corresponded to encoding multi-hop path information between target nodes; GDGNN [18] obtain the link representations by encoding the shortest path between the target nodes. This inspires us to design a simple method to test whether simply incorporating path information between target nodes can help MPNNs to learn higher-order representation.

5.2 Implementation for link prediction

We aim to derive the simplest model for link prediction without unnecessary components. Our method is partly inspired by SGC [35], an extremely concise and elegant GNN model whose predictor function is $\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{A}^k \boldsymbol{X} \boldsymbol{\Theta})$ where $\boldsymbol{\Theta}$ is a trainable weight matrix. Our model contains two steps, one corresponds to computing the 1-WL color and the other corresponds to hashing path information in Proposition 8. We name our methods Simplified Link Prediction neural networks (SLP).

Table 1: Results on link prediction tasks. All baseline results are taken from [6]. **Best results** are bold, and secondary results are underlined.

Model	Method	Cora	Citeseer	Pubmed
CN	Heuristic	33.92±0.46	29.79±0.90	23.13±0.15
AA	Heuristic	39.85±1.34	35.19±1.33	27.38±0.11
RA	Heuristic	41.07±0.48	33.56±0.17	27.03±0.35
GCN	MPNN	66.79±1.65	67.08±2.94	53.02±1.39
SAGE	MPNN	55.02±4.03	57.01±3.74	39.66±0.72
SEAL	2-1-NL_{MP}	81.71±1.30	83.89±2.15	**75.54**±1.32
NBFNet	1-1-NL_{MP}	71.65±2.27	74.07±1.75	58.73±1.99
Neo-GNN	Heuristic-enhanced MPNN	80.42±1.31	84.67±2.16	73.93±1.19
ELPH	Heuristic-enhanced MPNN	87.72±2.13	93.44±0.53	72.99±1.43
BUDDY	Heuristic-enhanced MPNN	88.00±0.44	92.93±0.27	74.10±0.78
SLP	Heuristic-enhanced MPNN	**89.83**±1.09	**93.61**±1.10	74.94±0.93

Step 1: Node feature propagation. Suppose $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ is the input node feature matrix where N is the number of nodes, d is the feature dimension. Let $\boldsymbol{S} \in \mathbb{R}^{N\times N}$ be the (normalized) adjacency matrix. In the first step, we propagate node features and obtain $\boldsymbol{Z} = \boldsymbol{S}^k\boldsymbol{X}$, where k is the number of propagation steps.

Step 2: Path feature propagation. We assume the sequence $(\boldsymbol{S}, \boldsymbol{S}^2, ..., \boldsymbol{S}^k)$ already contains enough information for identifying path information. Therefore, we can compute the path representation of (u, v) within one step: $\boldsymbol{h}_{uv} = [\boldsymbol{S}_{uv}, \boldsymbol{S}^2_{uv}, .., \boldsymbol{S}^k_{uv}]$. Note that $\boldsymbol{S}^k$ can be evaluated efficiently using sparse matrix multiplication.

Step 3: Compute predictions. The final prediction for the link (u, v) is directly realized by $p(u, v) = \psi(\boldsymbol{z}_u, \boldsymbol{z}_v, \boldsymbol{h}_{uv})$. Since in step 1, 2 there is no trainable parameters, the prediction model ψ is realized as a logistic regression or a MLP.

6 Empirical Evaluation

We conduct synthetic experiments with variants of i-j-NL_{MP} to verify the theoretical expressiveness results in this paper. Due to space limits the results are in Appendix F. We also apply SLP on widely used Planetoid citation network Cora [24], Citeseer [31] and Pubmed [28]. We report results for traditional heuristic methods including Common Neighbors (CN), Adamic-Adar (AA) [1] and Resource Allocation (RA) [45]. We report results for MPNNs including Graph Convolutional Networks (GCN) [16] and GraphSAGE [14]. We report results for 1-1-NL_{MP} NBFNet [46] and 2-1-NL_{MP} for undirected graphs SEAL [40]. We also report results for heuristic-enhanced MPNNs Neo-GNN [38], ELPH and BUDDY [6].

The main results are in Table 1, with all metrics being H@100. Additional details are in Appendix F. From the results we can see that even with simple model design, SLP is able to reach competitive results. On the one hand, SLP surpasses plain MPNNs such as GCN [17] and SAGE [14] by a large margin. This indicates the important of capturing path information for link prediction. On the other hand, although SLP is not as expressive as SEAL [40], NBFNet [46], etc., it still reaches competitive results. We believe the reason is that SLP captures the necessary features for link prediction.

7 Conclusion

In this paper, we developed theoretical foundations for higher-order representation learning methods with GNNs, and provide a better understanding of the expressive power and the connection within them, and point out the common properties of popular GNN variants and design a simple link prediction method based on our findings. We only focus on most-expressive GNNs and MPNNs as the backbone model respectively. Recently, people have developed many advanced GNN models that surpass the 1-WL test, but discussing them is out of the scope of this paper.

References

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Soc. Networks*, 25:211–230, 2003.

[2] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. P. Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[3] L. Bergen, T. J. O'Donnell, and D. Bahdanau. Systematic generalization with edge transformers. In *NeurIPS*, 2021.

[4] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021.

[5] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1989.

[6] B. P. Chamberlain, S. Shirobokov, E. Rossi, F. Frasca, T. Markovich, N. Y. Hammerla, M. Bronstein, and M. Hansmire. Graph neural networks for link prediction with subgraph sketching. In *ICLR*, 2023.

[7] Z. Chen, L. Chen, S. Villar, and J. Bruna. Can graph neural networks count substructures? In *NeurIPS*, 2020.

[8] V. Garg, S. Jegelka, and T. Jaakkola. Generalization and representational limits of graph neural networks. In *ICML*, 2020.

[9] F. Geerts. The expressive power of kth-order invariant graph networks. *arXiv preprint arXiv:2007.12035*, 2020.

[10] M. Grohe. The logic of graph neural networks. In *LICS*, 2021.

[11] M. Grohe and M. Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80: 797 – 844, 2012.

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

[13] A. G. Hamilton. *Logic for Mathematicians*. 1978.

[14] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[15] X. Huang, M. Romero, I. I. Ceylan, and P. Barceló. A theory of link prediction via relational weisfeiler-leman. *ArXiv*, abs/2302.02209, 2023. URL https://api.semanticscholar.org/CorpusID:256616025.

[16] T. Kipf and M. Welling. Variational graph auto-encoders. In *NeurIPS Workshop*, 2016.

[17] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[18] L. Kong, Y. Chen, and M. Zhang. Geodesic graph neural network for efficient graph representation learning. In *NeurIPS*, 2022.

[19] P. Li, Y. Wang, H. Wang, and J. Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *NeurIPS*, 2020.

[20] S. Liu, B. Grau, I. Horrocks, and E. Kostylev. Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding. In *NeurIPS*, 2021.

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

[22] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

[23] H. Maron, E. Fetaya, N. Segol, and Y. Lipman. On the universality of invariant networks. In *ICML*, 2019.

[24] A. McCallum, K. Nigam, J. D. M. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

[25] C. Morris, K. Kersting, and P. Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, 2017.

[26] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

[27] C. Morris, G. Rattan, and P. Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

[28] G. Namata, B. London, L. Getoor, B. Huang, and U. Edu. Query-driven active surveying for collective classification. In *10th international workshop on mining and learning with graphs*, volume 8, page 1, 2012.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *ArXiv*, abs/1912.01703, 2019.

[30] C. Qian, G. Rattan, F. Geerts, M. Niepert, and C. Morris. Ordered subgraph aggregation networks. In *NeurIPS*, 2022.

[31] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. In *The AI Magazine*, 2008.

[32] K. K. Teru, E. Denis, and W. L. Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, 2019.

[33] C. Wan, M. Zhang, W. Hao, S. Cao, P. Li, and C. Zhang. Principled hyperedge prediction with structural spectral features and neural networks. *arXiv preprint arXiv:2106.04292*, 2021.

[34] B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, pages 12–16, 1968.

[35] F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

[36] J. You, R. Ying, and J. Leskovec. Position-aware graph neural networks. In *ICML*, 2019.

[37] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec. Identity-aware graph neural networks. In *AAAI*, 2021.

[38] S. Yun, S. Kim, J. Lee, J. Kang, and H. J. Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. In *NeurIPS*, 2021.

[39] B. Zhang, S. Luo, L. Wang, and D. He. Rethinking the expressive power of gnns via graph biconnectivity. *ArXiv*, abs/2301.09505, 2023.

[40] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In *NeurIPS*, 2018.

[41] M. Zhang and Y. Chen. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

[42] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *NeurIPS*, 2020.

[43] M. Zhang, P. Li, Y. Xia, K. Wang, and L. Jin. Revisiting graph neural networks for link prediction. 2020.

[44] L. Zhao, L. Härtel, N. Shah, and L. Akoglu. A practical, progressively-expressive gnn. In *NeurIPS*, 2022.

[45] T. Zhou, L. Lü, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71:623–630, 2009.

[46] Z. Zhu, Z. Zhang, L.-P. Xhonneux, and J. Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, 2021.

A Proof of Theorem 2, Proposition 3 and Corollary 4

A.1 Proof of Theorem 2

We first restate Theorem 2 here.

Theorem 2. *Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$, and let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$ where k is a fixed value to be the target tuples we want to learn from G and H respectively. Then, the following statements are equivalent for any i, j satisfying $i \geq 0, j > 0, i + j = k$:*

- *A i-j-$\mathrm{NL_E}$ model gives $\mathbf{u}$ and $\mathbf{v}$ the same representation.*
- *$(\mathbf{u}, G) \simeq (\mathbf{v}, H)$.*

Proof. To prove the theorem, we need to first use the following fact about higher-order GNNs:

Lemma 9. *A k-GNN with most expressive GNNs distinguishes any non-isomorphic k-tuples, and assigns isomorphic k-tuples with the same representation.*

With Lemma A.1 we are now able to prove Theorem 2.

$1 \to 2$: Suppose a i-j-$\mathrm{NL_E}$ gives $\mathbf{u}$ and $\mathbf{v}$ the same representation. From Lemma A.1 it is obvious that we have

$$\left(\mathbf{u}_{(i+1):}, G_{(\mathbf{u}_{:i})}\right) \simeq \left(\mathbf{v}_{(i+1):}, H_{(\mathbf{v}_{:j})}\right).$$

From the definition of node labeling, we know that there exists an isomorphism π from G to H satisfying

$$\pi(u_i) = v_i \text{ for all } i \in \{i+1, ..., k\}, L(w \mid \mathbf{u}_{:i}, G) = L(\pi(w) \mid \mathbf{v}_{:i}, H) \text{ for all } w \in \mathcal{V}_G.$$

Since $L(\pi(w) \mid \mathbf{v}_{:i}, H)$ for all $w \in \mathcal{V}_G \Rightarrow \pi(\mathbf{u}_{:i}) = \mathbf{v}_{:i}$, we have

$$\begin{aligned}
&\left(\mathbf{u}_{(i+1):}, G_{(\mathbf{u}_{:i})}\right) \simeq \left(\mathbf{v}_{(i+1):}, H_{(\mathbf{v}_{:j})}\right) \\
&\Rightarrow \exists \text{ an isomorphism } \pi : \mathcal{V}_G \to \mathcal{V}_H, \pi(\mathbf{u}_{:i}) = (\mathbf{v}_{:i}), \pi(\mathbf{u}_{(i+1):}) = (\mathbf{v}_{(i+1):}) \\
&\Rightarrow \exists \text{ an isomorphism } \pi : \mathcal{V}_G \to \mathcal{V}_H, \pi(\mathbf{u}) = (\mathbf{v}) \\
&\Rightarrow (\mathbf{u}, G) \simeq (\mathbf{v}, H).
\end{aligned}$$

$2 \to 1$: With Lemma A.1 we can deduce

$$\begin{aligned}
&(\mathbf{u}, G) \simeq (\mathbf{v}, H) \\
\Longrightarrow\ &\exists \text{ an isomorphism } \pi : \mathcal{V}_G \to \mathcal{V}_H, \pi(\mathbf{u}_{:i}) = \mathbf{v}_{:i} \\
\Longrightarrow\ &\forall w \in \mathcal{V}_G, L(w \mid \mathbf{u}_{:i}, G) = L(w \mid \mathbf{v}_{:i}, H) \\
\Longrightarrow\ &\left(\mathbf{u}_{(i+1):}, G_{(\mathbf{u}_{:i})}\right) \simeq \left(\mathbf{v}_{(i+1):}, H_{(\mathbf{v}_{:j})}\right) \\
\Longrightarrow\ &i\text{-}j\text{-}\mathrm{NL_E} \text{ gives } \mathbf{u} \text{ and } \mathbf{v} \text{ the same representation.}
\end{aligned}$$

Since the above equations hold for all $i \geq 0, j > 0, i + j = k$, Theorem 2 is proved. □

A.2 Proof of Proposition 3

We first restate Proposition 3 here.

Proposition 3. *Suppose $i, p \geq 0, j, q > 0$ are integers. A p-q-$\mathrm{NL_E}$ can distinguish any non-isomorphic node tuples that a i-j-$\mathrm{NL_E}$ can distinguish, if and only if $p + q \geq i + j$.*

Proof. The proof is straightforward. Since the discriminative power of i-j-$\mathrm{NL_E}$ is strictly captured by the $(i + j)$-order isomorphism, the proof is also built on this fact.

$2 \to 1$: We first prove that $p + q \geq i + j \Rightarrow p$-$q$-$\mathrm{NL_E}$ distinguishes any non-isomorphic node tuples that a i-j-$\mathrm{NL_E}$ distinguishes. Given $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$ and $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$), we first prove the situation where $p + q = i + j + 1$.

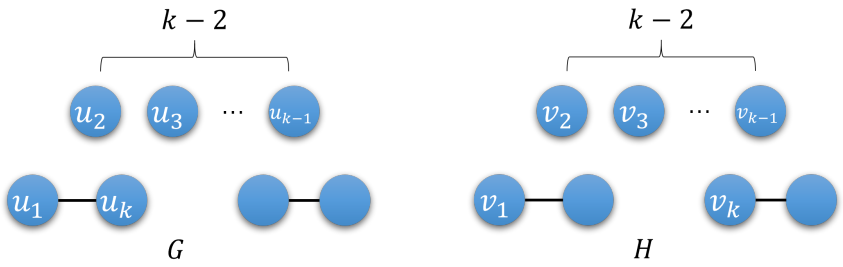


Figure 3: Example of non-isomorphic tuples.

- if $k = i + j$: Suppose $\mathbf{u}$ and $\mathbf{v}$ get different representations by i-j-NL$_{\mathrm{E}}$. By pooling the representations, p-q-NL$_{\mathrm{E}}$ computes the representation of $\mathbf{u}$ as $\{\{\boldsymbol{h}_{(u_1,\ldots,u_k,w)} \mid w \in \mathcal{V}\}\}_G$. Then,

$$\begin{aligned}
&\mathbf{u} \text{ and } \mathbf{v} \text{ get different representations by } i\text{-}j\text{-NL}_{\mathrm{E}} \\
\Longrightarrow\ & (\mathbf{u}, G) \ncong (\mathbf{v}, H) \\
\Longrightarrow\ & \forall w_1 \in \mathcal{V}_G, w_2 \in \mathcal{V}_H, ((u_1, ..., u_k, w_1), G \ncong (v_1, ..., v_k, w_2)) \\
\Longrightarrow\ & \forall w_1 \in \mathcal{V}_G, w_2 \in \mathcal{V}_H, \boldsymbol{h}_{(u_1,\ldots,u_k,w_1)} \neq \boldsymbol{h}_{(u_1,\ldots,u_k,w_2)} \\
\Longrightarrow\ & \{\{\boldsymbol{h}_{(u_1,\ldots,u_k,w)} \mid w \in \mathcal{V}_G\}\} \neq \{\{\boldsymbol{h}_{(v_1,\ldots,v_k,w)} \mid w \in \mathcal{V}_H\}\} \\
\Longrightarrow\ & p\text{-}q\text{-NL}_{\mathrm{E}} \text{ distinguishes } \mathbf{u}, \mathbf{v}.
\end{aligned}$$

- if $k \neq i + j$: From the above deduction we can see that for any pair of $(i + j)$-tuples, if i-j-NL$_{\mathrm{E}}$ assigns different representations for them, then so does p-q-NL$_{\mathrm{E}}$. Obviously, this leads to the fact that p-q-NL$_{\mathrm{E}}$ is always more expressive than i-j-NL$_{\mathrm{E}}$.

Note that when $p + q = i + j$ the result naturally holds. Then by proof by induction we have proved all situations for $p + q \geq i + j$.

$\neg 2 \rightarrow \neg 1$: Similar as before, we first assume $p + q = i + j - 1$. We build graphs $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$ and $\mathbf{u} \in \mathcal{V}_G^k, \mathbf{v} \in \mathcal{V}_H^k$ such that i-j-NL$_{\mathrm{E}}$ assigns different representations for $\mathbf{u}, \mathbf{v}$. p-q-NL$_{\mathrm{E}}$ also computes the representation of $\mathbf{u}$ by aggregation as $\left(\boldsymbol{h}_{(u_1,\ldots,u_{k-1})}, \boldsymbol{h}_{(u_2,\ldots,u_k)}\right)$. The graphs are constructed as in Figure 3. It is easy to see that $((u_1, ..., u_{k-1}), G) \simeq ((v_1, ..., v_{k-1}), H)$ and $((u_2, ..., u_k), G) \simeq ((v_2, ..., v_k), H)$ but clearly $(\mathbf{u}, G)$ and $(\mathbf{v}, H)$ are not isomorphic.

With proof by induction, the direction $\neg 2 \rightarrow \neg 1$ is proved. □

A.3 Proof of Corollary 4

We first restate Corollary 4 here.

Corollary 4. *Given the conditions in Theorem 2 hold. Then, we have*

$$G_{(\mathbf{u})} \simeq H_{(\mathbf{v})} \iff (\mathbf{u}, G) \simeq (\mathbf{v}, H),$$

where $G_{(\mathbf{u})}, H_{(\mathbf{v})}$ are node labeling induced graphs by labeling $\mathbf{u}$ and $\mathbf{v}$ respectively.

Proof. Corollary 4 is a special situation of the proof step 1→2 in the proof of Theorem 2.

$$\begin{aligned}
& G_{(\mathbf{u})} \simeq H_{(\mathbf{v})} \\
\Longleftrightarrow\ & \exists \text{ isomorphism } \pi, \forall w \in \mathcal{V}_G:\ L(w \mid \mathbf{u}, G) = L(\pi(w) \mid \mathbf{v}, H) \\
\Longleftrightarrow\ & \exists \text{ isomorphism } \pi(\mathbf{u}) = \mathbf{v} \\
\Longleftrightarrow\ & (\mathbf{u}, G) \simeq (\mathbf{v}, H).
\end{aligned}$$

□

A.4 Proof of Lemma A.1

Proof. Proving Lemma A.1 is equivalent to showing that for any graphs G, H, the target tuples $\mathbf{u}, \mathbf{v}$ and G^k, H^k, $(\mathbf{u}, G) \simeq (\mathbf{v}, H) \iff (\mathbf{u}, G^k) \simeq (\mathbf{v}, H^k)$.

1→2: There exists an isomorphism π from G to H, such that $\pi(\mathbf{u}) = \mathbf{v}$. We assume a k-order permutation π^k which satisfies $\pi^k(w_1, ..., w_k) = (\pi(w_1), ..., \pi(w_k))$. Then we prove that π^k is a valid isomorphism from G^k to H^k. For all $\mathbf{w} \in \mathcal{V}_G^k$, we have

$$\begin{aligned}
&\pi^k(\mathcal{N}(\mathbf{w})) \\
=&\left\{\left\{(\pi^k(i, ..., w_k), ..., \pi^k(w_1, ..., i)) \mid i \in \mathcal{V}_G\right\}\right\} \\
=&\{\{((\pi(i), ..., \pi(w_k)), ..., (\pi(w_1), ..., \pi(i))) \mid i \in \mathcal{V}_G\}\} \\
=&\mathcal{N}(\pi(\mathbf{w})).
\end{aligned}$$

Note that π is structure-preserving: this indicates that for all $(w_1, ..., w_k)$, the label of the corresponding $(\pi(w_1), ..., \pi(w_k))$ must also be the same. Since we also have $\pi(\mathbf{u}) = \mathbf{v}$, we have $(\mathbf{u}, G^k) \simeq (\mathbf{v}, H^k)$.

2→1:

$$\begin{aligned}
&(\mathbf{u}, G^k) \simeq (\mathbf{v}, H^k) \\
\implies &\exists \pi^k : \mathcal{V}_G^k \to \mathcal{V}_H^k, \forall \mathbf{w} \in \mathcal{V}_G^k : \pi^k(\mathbf{u}) = \mathbf{v}, \pi^k(\mathcal{N}(\mathbf{w})) = \mathcal{N}(\pi^k(\mathbf{w})).
\end{aligned}$$

From $\pi^k(\mathcal{N}(\mathbf{w})) = \mathcal{N}(\pi^k(\mathbf{w}))$ we can further deduce

$$\begin{aligned}
&\pi^k(\mathcal{N}(\mathbf{w})) = \mathcal{N}(\pi^k(\mathbf{w})) \\
\implies &\left\{\left\{(\pi^k(i, ..., w_k), ..., \pi^k(w_1, ..., i)) \mid i \in \mathcal{V}_G\right\}\right\} = \\
&\left\{\left\{\left((j, \pi^k(\mathbf{w})_2, ..., \pi^k(\mathbf{w})_k), ..., (\pi^k(\mathbf{w})_1, ...j)\right) \mid j \in \mathcal{V}_H\right\}\right\}.
\end{aligned}$$

Now we show that π^k must can be expressed by another $\pi : \mathcal{V}_G \to \mathcal{V}_H$ such that $\pi^k(w_1, ..., w_k) = (\pi(w_1), ..., \pi(w_k))$. First from the above equation we can directly observe that $\pi^k(w_1, ..., w_{k-1}, i)_{:k-1} = \pi^k(w_1, ..., w_{k-1}, w_k)_{:k-1}$ for all $w_1, ..., w_k, i \in \mathcal{V}_G$. This directly indicates that we can break π^k into two parts: $\pi^k(w_1, ..., w_k) = (\pi^{k-1}(w_1, ..., w_{k-1}), \pi_k(w_k))$ where $\pi_k : \mathcal{V}_G \to \mathcal{V}_H$. The same way, by substituting the result into $\pi^k(w_1, ..., i, w_k)_{:k-2} = \pi^k(w_1, ..., w_{k-1}, w_k)_{:k-2}$ and continuing we can finally obtain $\pi^k(w_1, ..., w_k) = (\pi_1(w_1), ..., \pi_k(w_k))$. Next, we need to show that $\pi_1, ..., \pi_k$ are all equivalent. By substituting into $\pi^k(\mathcal{N}(\mathbf{w})) = \mathcal{N}(\pi^k(\mathbf{w}))$ we have

$$\begin{aligned}
&\left\{\left\{(\pi^k(i, ..., w_k), ..., \pi^k(w_1, ..., i)) \mid i \in \mathcal{V}_G\right\}\right\} = \\
&\left\{\left\{\left((j, \pi^k(\mathbf{w})_2, ..., \pi^k(\mathbf{w})_k), ..., (\pi^k(\mathbf{w})_1, ...j)\right) \mid j \in \mathcal{V}_H\right\}\right\} \\
\implies &\{\{((\pi_1(i), ..., \pi_k(w_k)), ..., (\pi_1(w_1), ..., \pi_k(i))) \mid i \in \mathcal{V}_G\}\} = \\
&\{\{((j, ..., \pi_k(w_k)), ..., (\pi_1(w_1), ..., j)) \mid j \in \mathcal{V}_H\}\}.
\end{aligned}$$

Therefore, for all $i \in \mathcal{V}_G$, there must exists a corresponding $j \in \mathcal{V}_H$ such that $\pi_1(i) = \pi_2(i) = ... = \pi_k(i) = j$. As a result, we proved that there exists $\pi : \mathcal{V}_G \to \mathcal{V}_H$ such that $\pi^k(w_1, ..., w_k) = (\pi(w_1), ..., \pi(w_k))$. Next we need to show that π is an isomorphism from G to H. This is easily done by noticing that for all $w_1, ..., w_k \in \mathcal{V}_G$, the structures between $w_1, ..., w_k$ and $\pi(w_1), ..., \pi(w_k)$ must be the same, otherwise $(w_1, ..., w_k)$ and $(\pi(w_1), ..., \pi(w_k))$ would have different labels. Therefore, for any w_1, w_2 we have $(w_1, w_2) \in \mathcal{E}_G \iff (\pi(w_1), \pi(w_2)) \in \mathcal{E}_H$, and π is an isomorphism from G to H. Together, we have $(\mathbf{u}, G) \simeq (\mathbf{v}, H)$. □

B Proof of Theorem 5 and Proposition 6, 7, 8

B.1 Proof of Theorem 5

We first restate Theorem 5 here.

Theorem 5. *Given two graphs $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$, and let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k, \mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$ where k is a fixed value to be the target tuples we want to learn from G and H respectively. Then, for any i, j satisfying $i > 0, j > 0, i + j = k$, the expressive power of $(i-1)$-$(j+1)$-$\mathrm{NL_{MP}}$ is strictly higher than i-j-$\mathrm{NL_{MP}}$, that is:*

- *There is a i-j-$\mathrm{NL_{MP}}$ that distinguishes $\mathbf{u}$ and $\mathbf{v} \Longrightarrow$ There is a $(i-1)$-$(j+1)$-$\mathrm{NL_{MP}}$ that distinguishes $\mathbf{u}$ and $\mathbf{v}$.*
- *There exists $G, H, \mathbf{u}, \mathbf{v}$ such that a $(i-1)$-$(j+1)$-$\mathrm{NL_{MP}}$ distinguishes $\mathbf{u}$ and $\mathbf{v}$ but i-j-$\mathrm{NL_{MP}}$ cannot.*

Proof. We prove the two results separately. To prove the first result, we first recall the procedure of 0-k-$\mathrm{NL_{MP}}$ (or k-FWL). Given $G = (\mathcal{V}, \mathcal{E})$, at each layer it evaluates

$$\mathrm{Col}^{(l+1)}(u_1, ..., u_k) = \mathrm{Hash}\Big(\mathrm{Col}^{(l)}(u_1, ..., u_k), \Big\{\Big\{\Big(\mathrm{Col}^{(l)}(v, u_2, ..., u_k), \mathrm{Col}^{(l)}(u_1, v, ..., u_k), ..., \mathrm{Col}^{(l)}(u_1, ..., u_{k-1}, v)\Big) \mid v \in \mathcal{V}\Big\}\Big\}\Big).$$

If we consider a fragment of the above procedure where we replace the first i colors in each neighbor with the initial colors

$$\begin{aligned}\mathrm{Col}^{(l+1)}(u_1, ..., u_k) = \mathrm{Hash}\Big(&\mathrm{Col}^{(l)}(u_1, ..., u_k), \\ &\Big\{\Big\{\Big(\mathrm{Col}^{(0)}(v, u_2, ..., u_k), ..., \mathrm{Col}^{(0)}(u_1, ..., u_{i-1}, v, u_{i+1}, ..., u_k), \\ &\mathrm{Col}^{(l)}(u_1, ..., u_i, v, u_{i+2}, ..., u_k), ..., \mathrm{Col}^{(l)}(u_1, ..., u_{k-1}, v)\Big) \mid v \in \mathcal{V}_G\Big\}\Big\}\Big).\end{aligned}$$

Clearly, this variant is less expressive than the original one. We can rewrite this variant to make it strictly corresponded to i-$(k-i)$-$\mathrm{NL_{MP}}$. First, note that the first i variables in $\mathrm{Col}^{(l)}$ and $\mathrm{Col}^{(l+1)}$ are the same. This inspires that we can rewrite the equations when we fix the first i variables $u_1, ..., u_i$ as

$$f^{(l+1)}(u_{i+1}, ..., u_k) = \phi\left(f^{(l)}(u_{i+1}, ..., u_k), \left\{\left\{\left(f^{(l)}(v, ..., u_k), ..., f^{(l)}(u_{i+1}, ..., v)\right) \mid v \in \mathcal{V}\right\}\right\}\right),$$

where we initialize $f^{(0)}$ as $f^{(0)}(u_{i+1}, ..., u_k) = \mathrm{Col}^{(0)}(u_1, ..., u_k)$. The computation of $f^{(l)}$ is exactly the same as i-$(k-i)$-$\mathrm{NL_{MP}}$ when we apply node labeling on $u_1, ..., u_k$. Since i-j-$\mathrm{NL_{MP}}$ is clearly corresponded to a more "small" fragment of the above equations compared with $(i-1)$-$(j+1)$-$\mathrm{NL_{MP}}$, we have proved the first result.

To prove the second result, we need to utilize the results from Grohe and Otto [11]. Cai et al. [5] designed a construction of a series of pairs of non-isomorphic graph CFI(k) such that $(k-1)$-FWL fails to distinguish them. Further more, Grohe and Otto [11] proposed a variant of CFI graphs and showed that there are graphs such that $(k-1)$-FWL cannot distinguish them but k-FWL can. Our proof is based on this result. Suppose G, H are non-isomorphic graphs than cannot by distinguished by j-FWL but are distinguished by $(j+1)$-FWL. We first obtain the following property of FWL

Lemma 10. *If two graphs $G_1 = (\mathcal{V}_{G_1}, \mathcal{E}_{G_1}), H_1 = (\mathcal{V}_{H_1}, \mathcal{E}_{H_1})$ are not distinguished by k-FWL, $G_2 = (\mathcal{V}_{G_2}, \mathcal{E}_{G_2}), H_2 = (\mathcal{V}_{H_2}, \mathcal{E}_{H_2})$ are also not distinguished by k-FWL, then we let $G = (\mathcal{V}_G, \mathcal{E}_G)$ where $\mathcal{V}_G = \mathcal{V}_{G_1} \cup \mathcal{V}_{G_2}, \mathcal{E}_G = \mathcal{E}_{G_1} \cup \mathcal{E}_{G_2}$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$ where $\mathcal{V}_H = \mathcal{V}_{H_1} \cup \mathcal{V}_{H_2}, \mathcal{E}_H = \mathcal{E}_{H_1} \cup \mathcal{E}_{H_2}$. G and H are still not distinguished by k-FWL.*

It is easy to prove Lemma 10 by induction. Suppose φ is a FOC_{k+1} formula. Then,

- If φ is of the form $\varphi := C$, i.e. node colors or edges, then obviously φ produces the same results on G, H.
- If $\varphi := \varphi' \wedge \varphi''$ and φ', φ'' produce the same results on G, H, then obviously φ also produces the same results on G, H.
- if $\varphi := \neg\varphi'$, the situation is the same as before.

- if $\varphi := \exists^N \varphi'$, and φ' produces the same results on G, H. We let m, n to be the number of distinct groundings of φ' on G_1, G_2 respectively. Then, the number of distinct groundings of φ' on H_1, H_2 are also m, n. (otherwise a formula $\varphi'' = (\exists^m \varphi') \wedge \neg(\exists^{m+1} \varphi')$ differs on G_1, H_1 or $\varphi'' = (\exists^n \varphi') \wedge \neg(\exists^{n+1} \varphi')$ differs on G_2, H_2.) Therefore, on both G, H φ still produces the same results.

Therefore, any FOC_{k+1} formula, if produces the same results on G_1, H_1 and G_2, H_2, also produces the same results on G, H. As a result k-FWL cannot distinguishes G, H.

With Lemma 10 we can construct counterexamples for i-j-$\mathrm{NL}_{\mathrm{MP}}$ and $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$. Suppose G, H cannot be distinguished by j-FWL but are distinguished by $(j+1)$-FWL. We add k isolated nodes $u_1, ...u_k$ to G and $v_1, ...v_k$ to H, obtaining G', H'. Furthermore, the labels of $u_1, ..., u_k$ are distinct and different from the rest of the nodes, and we let the label of v_l to be the same with u_l for $l \in [k]$. We then apply i-j-$\mathrm{NL}_{\mathrm{MP}}$ and $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$ to learn the representation of $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k$ and $\mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$. Apparently from Lemma 10 we know that i-j-$\mathrm{NL}_{\mathrm{MP}}$ cannot distinguish them. Since a $(j+1)$-FWL can distinguish G and H, it also distinguishes G' and H'. This indicates that there is a FOC_{j+1} formula φ such that $G' \vDash \varphi$ and $H' \nvDash \varphi$. By letting $\psi(x_1, ..., x_j) := \varphi()$ we can see that $\psi(u_{i+1}, ..., u_k) \neq \psi(v_{i+1}, ..., v_k)$. Therefore, the $j+1$-FWL also assign different colors to $(u_{i+1}, ..., u_k)$ and $(v_{i+1}, ..., v_k)$. As a result $(i-1)$-$(j+1)$-$\mathrm{NL}_{\mathrm{MP}}$ distinguishes them. □

B.2 Proof of Proposition 6

To prove Proposition 6 is to construct a counterexample for i-j-$\mathrm{NL}_{\mathrm{MP}}$ and p-q-$\mathrm{NL}_{\mathrm{MP}}$. Specially, from the proof of Theorem 5 we know that we only need to construct a counterexample for k-1-$\mathrm{NL}_{\mathrm{MP}}$ and 0-k-$\mathrm{NL}_{\mathrm{MP}}$ for any k. We first introduce a series of graph pairs proposed by Grohe and Otto [11].

Let $\mathcal{K} = (\mathcal{V}, \mathcal{E})$ be the complete graph on k nodes. We further assume the nodes in $\mathcal{K}$ are $v_1, ..., v_k$. We define the construction of a structure $\mathcal{X}(\mathcal{K})$, which is the *CFI-companions* of $\mathcal{K}$, as follows. It is convenient to call the nodes from $\mathcal{X}(\mathcal{K})$ *vertices*, to distinguish them from the nodes of $\mathcal{K}$.

For every $v \in \mathcal{V}$, the graph $\mathcal{X}(\mathcal{K})$ has a vertex v^S, where S is a subset of $\mathcal{E}(v)$ of even cardinality. We use $\mathcal{E}(v)$ to denote the edges connected with v. For every edge $e \in \mathcal{E}$, the graph $\mathcal{X}$ has two vertices e^0, e^1. Vertices of the form v^S are called node vertices and vertices e^i edge vertices. Formally, the set of vertices in $\mathcal{X}(\mathcal{K})$ is

$$\{v^S \mid v \in \mathcal{V}, S \subseteq \mathcal{E}(v) \text{ such that } |S| \equiv 0 \bmod 2\} \cup \{e^0, e^1 \mid e \in \mathcal{E}\}. \tag{2}$$

The edges of $\mathcal{X}(\mathcal{K})$ link node vertices and edge vertices according to

$$(v^S, e_i) \text{ is an edge if } \begin{cases} i = 1 \text{ and } e \in S \\ i = 0 \text{ and } e \notin S \end{cases} \tag{3}$$

In $\mathcal{X}(\mathcal{K})$, the vertices of the form v^S are colored C_v, and the vertices of the form e^i are colored C_e. After defining $\mathcal{X}(\mathcal{K})$, we also define its variant $\widehat{\mathcal{X}}(\mathcal{K})$ whose node set and edge set are the same except that for the node v_1 we take nodes v_1^S from the subsets of $\mathcal{E}(v_1)$ of odd cardinality. It is proved by Grohe and Otto [11] that $\mathcal{X}(\mathcal{K}), \widehat{\mathcal{X}}(\mathcal{K})$ are distinguished by $(k-1)$-FWL but not $(k-2)$-FWL.

Before we proceed, we would like to provide a $k = 3$ example to illustrate $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$ in Figure 4. The graphs $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$ are distinguished by 2-FWL but not 1-WL.

Now we proceed to prove Proposition 6. We first restate it here.

Proposition 6. *Suppose $i, p \geq 0, j, q > 0$ are integers. There is a p-q-$\mathrm{NL}_{\mathrm{MP}}$ that distinguishes any non-isomorphic node tuples that a i-j-$\mathrm{NL}_{\mathrm{MP}}$ can distinguish if and only if $p + q \geq i + j$ and $q \geq j$. Otherwise, there exists non-isomorphic node tuples that a p-q-$\mathrm{NL}_{\mathrm{MP}}$ cannot distinguish but a i-j-$\mathrm{NL}_{\mathrm{MP}}$ can distinguish.*

Proof. We first show that there are graphs distinguished by $(k-2)$-1-$\mathrm{NL}_{\mathrm{MP}}$ but not 0-$(k-2)$-$\mathrm{NL}_{\mathrm{MP}}$. We construct CFI graphs $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$. From Grohe and Otto [11] we know that $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$ cannot be distinguished by 0-$(k-2)$-$\mathrm{NL}_{\mathrm{MP}}$, whose expressive power is bounded by $(k-2)$-FWL.

Next we show that $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$ can be distinguished by a $(k-2)$-1-$\mathrm{NL}_{\mathrm{MP}}$. To do so we need to introduce the concept of *pebble games*[11, 5]. The readers are welcome to check [11] for a more detailed introduction. Given two structures $\mathcal{A}, \mathcal{B}$, the *bijective k-pebble game* is played by two players by placing k pairs of pebbles on a pair of structures $\mathcal{A}, \mathcal{B}$. The rounds of the game are as follows. Player **I** picks up one of his pebbles, and player **II** picks up her corresponding pebble. Then player **II** chooses a bijection f between $\mathcal{A}$ and $\mathcal{B}$ (if no such bijection exists player **II** immediately loses). Then player **I** places his pebble on an element a of $\mathcal{A}$, and player **II** places her pebble on $f(a)$. After each round there is a subset $p \subseteq \mathcal{A} \times \mathcal{B}$ consisting of the at most k pairs of elements corresponding to the pebbles placed. Player **II** wins a play if every position p is a local isomorphism.

Theorem 11. *(*Cai et al. [5]*)* $\mathcal{A} \equiv_C^k \mathcal{B}$ *if and only if player* ***II*** *has a winning strategy for the bijective k-pebble game on* $\mathcal{A}, \mathcal{B}$*.*

Theorem 11 indicates that $(k-1)$-FWL can distinguish $\mathcal{A}, \mathcal{B}$ if and only if player **II** has a winning strategy for the bijective k-pebble game on $\mathcal{A}, \mathcal{B}$. However, although it justifies $(k-1)$-FWL, it has nothing to do with the $(k-2)$-1-$\mathrm{NL}_{\mathrm{MP}}$ here. We propose a variant of the bijective k-pebble game, namely restricted bijective k-i-pebble game, described as follows. Given two structures $\mathcal{A}, \mathcal{B}$, the *restricted bijective k-i-pebble game* is also played by two players by placing k pairs of pebbles on a pair of structures $\mathcal{A}, \mathcal{B}$. The rounds of the game are as follows. Player **I** picks up one of his pebbles, and player **II** picks up her corresponding pebble. Then player **II** chooses a bijection f between $\mathcal{A}$ and $\mathcal{B}$ (if no such bijection exists player **II** immediately loses). Then player **I** places his pebble on an element a of $\mathcal{A}$, and player **II** places her pebble on $f(a)$. The difference is, i pairs of the pebbles are static, as when the players place these pebbles on the elements of $\mathcal{A}, \mathcal{B}$, they can no longer pick and replace them on other elements. Then, we have

Theorem 12. *There is a* $(k-2)$-1-$\mathrm{NL}_{\mathrm{MP}}$ *that distinguishes the* $(k-2)$*-tuples* $\mathbf{u}, \mathbf{v}$ *from* $\mathcal{A}, \mathcal{B}$ *if and only if player* ***II*** *has a winning strategy for the restricted bijective k-(k-2)-pebble game on* $\mathcal{A}, \mathcal{B}$ *which initially places* $(k-2)$ *static pebbles on* $\mathbf{u}, \mathbf{v}$*, if* $\mathcal{A}, \mathcal{B}$ *are connected graphs.*

Theorem 12 is proved in the next section. With Theorem 12 we can now prove that the graphs $\mathcal{X}(\mathcal{K})$ and $\widehat{\mathcal{X}}(\mathcal{K})$ can be distinguished by $(k-2)$-1-$\mathrm{NL}_{\mathrm{MP}}$. The prove steps are exactly the same as in [11], as the steps in [11] naturally follow the constraints of the bijective k-$(k-2)$-pebble game.

We give a winning strategy for player **I** in the bijective k-$(k-2)$-pebble game. In the first $k-1$ rounds of the game, player **I** picks his $k-1$ pebbles on $v_2^{\emptyset}, ..., v_k^{\emptyset}$ and suppose $p(v_i^{\emptyset}) = v_i^{S_i}$ is the corresponding position for some sets S_i. That is,

$$p = \{v_2^{\emptyset} v_2^{S_2}, ..., v_k^{\emptyset} v_k^{S_k}\}.$$

We now assume that the pebbles at $v_3^{\emptyset}, ..., v_k^{\emptyset}$ are static. Therefore $v_3^{\emptyset}, ..., v_k^{\emptyset}$ compose all $k-2$ static pebbles and the pebble at $v_2^{\emptyset}$ is still movable. In the next round of the game player **I** starts by selecting this pebble, and places it on e_{12}^0. In the next round, player **I** starts by selecting the pebble on e_{12}^0 and places it on $v_1^{\emptyset}$. It is proved by Grohe and Otto [11] that player **I** wins at this time.

□

B.3 Proof of Proposition 7

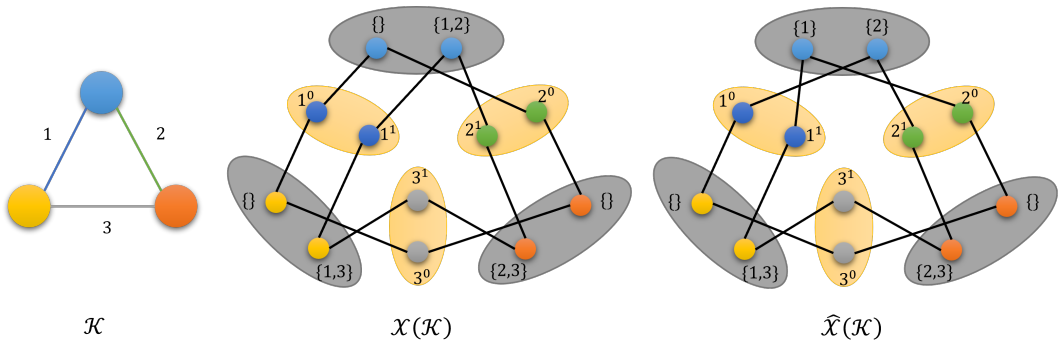
We first restate Proposition 7 here.

Proposition 7. *For any* k*-tuples* $\mathbf{u}, \mathbf{v}$ *and* $i + j = k$*: There is a* i-j-$\mathrm{NL}_{\mathrm{MP}}$ *that distinguishes* $\mathbf{u}$ *and* $\mathbf{v}$ $\iff$ *There is a* $\mathrm{FOC}_{k+1,i}$ *formula that distinguishes* $\mathbf{u}$ *and* $\mathbf{v}$*.*

Proof. To prove Proposition 7 we can first use the results from [5].

Theorem 13. (Cai et al. [5]) *Let* G, H *be a pair of colored graphs and let* $\mathbf{u} \in \mathcal{V}_G^k, \mathbf{v} \in \mathcal{V}_H^k$ *be* k*-tuples. The following are equivalent:*

- k*-FWL assigns the same color for* $\mathbf{u}, \mathbf{v}$*.*
- *All* FOC_{k+1} *produce the same result for* $\mathbf{u}, \mathbf{v}$*.*



$\mathcal{K}$ $\mathcal{X}(\mathcal{K})$ $\widehat{\mathcal{X}}(\mathcal{K})$

Figure 4: Example of non-isomorphic graphs.

The above equivalence can be extend to node labeling situations. Formally, given $G = (\mathcal{V}, \mathcal{E})$ and $\mathbf{u} = (u_1, ..., u+k) \in \mathcal{V}^k$, and let $i + j = k$. Then, applying a i-j-$\mathrm{NL}_{\mathrm{MP}}$ is essentially first assigning additional predicates $\mathrm{IsTarget}_l(\cdot)$ for $l \in [i]$. Further more, we have $\mathrm{IsTarget}_l(v) = \mathbf{1}_{v=u_l}$. Then, it applies a j-MPNN on the augmented graph $G_{(\mathbf{u}_{:i})}$ whose expressive power is bounded by j-FWL, and thus is be described by FOC_{j+1}.

Now we need to prove the equivalence between FOC_{j+1} on the augment graphs $G_{(\mathbf{u}_{:i})}$ and $\mathrm{FOC}_{k+1,i}$ on the original graph G. Specially, we want to prove

1. Given any FOC_{j+1} formula φ on $G_{(\mathbf{u}_{:i})}$, there is a corresponding $\mathrm{FOC}_{k+1,i}$ formula ψ on G that expresses φ.
2. Given any $\mathrm{FOC}_{k+1,i}$ formula ψ on G, there exists $\mathbf{u}$ such that a FOC_{j+1} formula φ on $G_{(\mathbf{u}_{:i})}$ expresses ψ.

1. It is simple to create the corresponding ψ. We only need to replace all unary predicates $C(v)$ with an invented one $C'(x_1, x_2, ..., x_i, v)$, and replace all binary predicates $E'(v, w)$ with an invented one $E'(x_1, x_2, ..., x_i, v, w)$. We define

$$C'(x_1, x_2, ..., x_i, v) := \begin{cases} \mathbf{1}_{x_l=v}, & \text{if } C(v) := \mathrm{IsTarget}_l(v), \\ C(v), & \text{else.} \end{cases}$$
$$E'(x_1, x_2, ..., x_i, v, w) := E(v, w).$$

Therefore, we effectively remove all emergence of the additional predicates $\mathrm{IsTarget}_l(\cdot)$.

2. Suppose the target tuple is fixed $\mathbf{u}$. This indicates that all predicates in ψ share the same first i variables $\mathbf{u}_{:i}$. The same way, we replace predicates in ψ with our invented ones. Note that a predicate $P(u_1, ..., u_k)$ is defined by any permutation-invariant functions over the subgraph induced by $u_1, ..., u_k$. Therefore we initialize $P'(u_{i+1}, ..., u_k) = P(u_1, ..., u_k)$ for all P. Since all predicates in ψ share the same first i variables $\mathbf{u}_{:i}$, we can replace all emergence of predicates in ψ and eliminate the first i variables in this way. We have constructed the corresponding FOC_{j+1} formula φ on $G_{\mathbf{u}_{:i}}$. □

B.4 Proof of Proposition 8

We first restate Proposition 8 here.

Proposition 8. *Given a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$, we run 1-WL to assign node colors for G, denoted as $\mathrm{Col}(\cdot)$. The color of a path $P = (w_1, ..., w_d)$ is obtained by hashing the corresponding node color sequence $\mathrm{Col}(P) = \mathrm{Hash}(\mathrm{Col}(w_1), ..., \mathrm{Col}(w_d))$. Then, the correlation between two nodes (u, v), as discussed above, is expressed by all paths between u, v,*

$$\mathrm{Corr}(u, v) = \mathrm{Hash}\left(\{\{\mathrm{Col}(P) \mid P \in \mathrm{Paths}(u, v)\}\}\right).$$

Given two connected graphs $G = (\mathcal{V}_G, \mathcal{E}_G)$ and $H = (\mathcal{V}_H, \mathcal{E}_H)$. Let $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}_G^k$, $\mathbf{v} = (v_1, ..., v_k) \in \mathcal{V}_H^k$ be the target node tuples. If there exists $1 \le p, q \le k$ such that $\mathrm{Corr}(u_p, u_q) \neq$

$\mathrm{Corr}(v_p, v_q)$*, then any* i-j-$\mathrm{NL}_{\mathrm{MP}}$ *with* $i + j \geq k$ *with sufficient injective layers always distinguishes* $\mathbf{u}$ *and* $\mathbf{v}$.

Proof. Since the weakest i-j-$\mathrm{NL}_{\mathrm{MP}}$ variant given fixed $k = i + j$ is the $(k-1)$-1-$\mathrm{NL}_{\mathrm{MP}}$, we only prove the result for $(k-1)$-1-$\mathrm{NL}_{\mathrm{MP}}$.

Suppose $\mathrm{Corr}(u_p, u_q) \neq \mathrm{Corr}(v_p, v_q)$. Without loss of generality we may assume that u_p, v_p are with node labeling. By the definition of Corr we know that there are at least one type of color of paths $C = \mathrm{Hash}(C_1, ..., C_d)$ such that the number of C-colored paths between (u_p, u_q) are different from that between (v_p, v_q). We manually construct a i-j-$\mathrm{NL}_{\mathrm{MP}}$ model that distinguishes u_q, v_q from $G_{(\mathbf{u})}, H_{(\mathbf{v})}$. First, we add sufficient injective layers to the model, but these layers are not aware of the additional node labels. In other words, we compute the 1-WL color Col in this way. Then, we let the next layers to detect the paths as

$$\mathrm{Layer}^{(1)}(x) = (\mathrm{Col}(x), \mathbf{1}(x \text{ is the } p\text{-th node labeling}) \cdot \mathbf{1}(\mathrm{Col}(x) = C_1)),$$

where $\mathbf{1}$ is the indicator function. Similarly, we define the l layer for $l \leq d$ as

$$\begin{aligned}
&\mathrm{ColLayer}^{(l)}(x) = \mathrm{Layer}^{(l-1)}(x)[0] = \mathrm{Col}(x),\\
&\mathrm{PathLayer}^{(l)}(x) = \mathrm{Layer}^{(l-1)}(x)[1],\\
&\mathrm{Layer}^{(l)}(x) = \left(\mathrm{ColLayer}^{(l)}(x), \mathbf{1}\left(\mathrm{ColLayer}^{(l)}(x) = C_l\right) \cdot \sum_{y \in \mathcal{N}(x)} \left(\mathrm{PathLayer}^{(l)}(x)\right)\right).
\end{aligned}$$

Therefore, $\mathrm{Layer}^{(\mathrm{d})}(x)$ counts the number of C-colored paths between x and u_p or v_p. Thus $\mathrm{Layer}^{(\mathrm{d})}(u_q) \neq \mathrm{Layer}^{(\mathrm{d})}(v_q)$. If $q = k$, then this directly indicates that our $(k-1)$-1-$\mathrm{NL}_{\mathrm{MP}}$ distinguishes them. If $q < k$, then we further add sufficient identical layers as follows.

$$\mathrm{IdLayer}^{(0)}(x) = \mathbf{1}(x \text{ is the } q\text{-th node labeling}) \odot \mathrm{Layer}^{(\mathrm{d})}(x)[1],$$

and

$$\mathrm{IdLayer}^{(l)}(x) = \max_{y \in \mathcal{V}(x)} \mathrm{IdLayer}^{(l-1)}(y).$$

After sufficient layers, IdLayer gives different results on u_k and v_k. □

B.5 Proof of Theorem 12

Proof. Let G, H be a pair of connected graphs and let $\mathbf{u}, \mathbf{v}$ be the target k-tuples. Let $G_{(\mathbf{u})}, H_{(\mathbf{v})}$ be the corresponding node labeling induced graphs. Since G, H are connected, we need to show the following statements are equivalent.

1. 1-WL cannot distinguish $G_{(\mathbf{u})}, H_{(\mathbf{v})}$
2. 2-WL cannot distinguish $G_{(\mathbf{u})}, H_{(\mathbf{v})}$
3. No FOC_2 formula distinguishes $G_{(\mathbf{u})}, H_{(\mathbf{v})}$
4. Player **II** has a winning strategy for the $(k+2)$-k pebble game on G, H which initially places the k pairs of static pebbles on $\mathbf{u}, \mathbf{v}$.

$1 \iff 2 \iff 3$: $2 \iff 3$ is proved as a special case in Cai et al. [5]. Since G, H are connected, $1 \iff 2$ also holds.

$2 \Rightarrow 4$: Suppose after r iterations the 2-WL still assigns the same color to $G_{(\mathbf{u})}, H_{(\mathbf{v})}$. We instead prove the following statement:

- After $r + k$ iterations 2-WL gives $(x_1, y_1) \in \mathcal{V}^2_{G_{(\mathbf{u})}}$ and $(x_2, y_2) \in \mathcal{V}^2_{H_{(\mathbf{v})}}$ the same color $\Longrightarrow$ Player **II** has a winning strategy for the $(k+2)$-k pebble game on G, H which initially places the k pairs of static pebbles on $\mathbf{u}, \mathbf{v}$ and place the other 2 pairs of static pebbles on (x_1, y_1) and (x_2, y_2) in r moves.

We assume W^r to be the color assignment of 2-WL at iteration $r+k$. Clearly player **I** can only chooses one of the pebbles on x_1, y_1. Without loss of generality suppose he picks up x_1. The player **II** answers with the bijective mapping that maps node pairs with the same W^{r-1} color, that is, $f(t_1) \in \{t_2 \mid W^{r-1}(t_1, y_1) = W^{r-1}(t_2, y_2)\}$. Note that such mapping must exist because

$$W^r(x,y) = \text{Hash}\left(W^{r-1}(x,y), \left\{\left\{W^{r-1}(x,z) \mid z \in \mathcal{V}_G\right\}\right\}, \left\{\left\{W^{r-1}(z,y) \mid z \in \mathcal{V}_G\right\}\right\}\right)$$

is the same for (x_1, y_1) and (x_2, y_2). No matter which node player **I** places his pebble on, player **II** places her pebble on the corresponding node. Player **II** has not yet lost: the structure between (t_1, y_1) and (t_2, y_2) must be the same, otherwise they have different W^{r-1} colors. The structure between $(t_1, \mathbf{u})$ and $(t_2, \mathbf{v})$ are also the same: This is because that at the start we added unique labels to the k-tuples $\mathbf{u}$ and $\mathbf{v}$. Therefore, after the first k rounds of 2-WL iterations if the subgraphs induced by $t_1, y_1, \mathbf{u}$ from $G_{(\mathbf{u})}$ and $t_2, y_2, \mathbf{v}$ from $H_{(\mathbf{v})}$ are different, (t_1, y_1) and (t_2, y_2) will also have different 2-WL colors. Now, since (t_1, y_1) and (t_2, y_2) have the same W^{r-1} colors, by induction on r we proved the above statement.

By further induction on r in the statement, we can prove 2⇒4 because we have showed that for any node pairs the results of 2-WL and the pebble game are always consistent.

¬3⇒ ¬4: Suppose for some FOC_2 formula φ, $G_{(\mathbf{u})} \models \varphi$ and $H_{(\mathbf{v})} \not\models \varphi$. If φ is a conjunction then $G_{(\mathbf{u})}, H_{(\mathbf{v})}$ must differs on at least one of the conjuncts, so we may assume φ is of the form $\exists^N x\psi$. Without loss of generality we assume the quantifier depth of φ is r. Note that there are total 2 free pairs of pebbles that can be placed to nodes, which exactly corresponds to the number of free variables in FOC_2 formula. Player **I** takes a pebble, corresponding to the variable x in φ. Player **II** must respond with a bijective mapping f. Since $G_{(\mathbf{u})} \models \varphi$ and $H_{(\mathbf{v})} \not\models \varphi$, we know that there are at least N nodes satisfying ψ in $G_{(\mathbf{u})}$ but less than N nodes satisfying ψ in $H_{(\mathbf{v})}$. Player **I** then picks the node w in $G_{(\mathbf{u})}$ such that $\psi(w)$ is true but $\psi(f(w))$ is false. By induction we can see that at quantifier depth 0 player **II** loses the game. □

C The algorithm complexities of i-j-NL_{MP}

In this section we derive the algorithm complexities of i-j-NL_{MP}.

C.1 The case of $j \geq 2$

We first study the situation where we assume $j \geq 2$. Suppose we are given a graph $G = (\mathcal{V}, \mathcal{E})$ with N nodes and M edges. Our model is a i-j-NL_{MP}, and we let $k = i + j$. Suppose we want to evaluate every k-tuple $\mathbf{u} \in \mathcal{V}^k$.

Obviously there are total N^k target tuples in the graph. For learning one tuple $\mathbf{u} \in \mathcal{V}^k$, we need to first assign node labeling to $\mathbf{u}_{:i}$, then apply a j-MPNN on $G_{(\mathbf{u}_{:i})}$. The numbers of nodes and edges in $G_{(\mathbf{u}_{:i})}$ are still N, M respectively. Since we assume the j-MPNN simulate the j-FWL, applying the j-MPNN on $G_{(\mathbf{u}_{:i})}$ and simultaneously learning representations all j-tuples requires $\mathcal{O}(N^j)$ space and $\mathcal{O}(N^{j+1})$ time. After this procedure, we actually learns all k-tuples with the same first i nodes $\mathbf{u}_{:i}$, so we only need to apply this procedure for N^i times to learn all representations for all k-tuples. Therefore, we need $\mathcal{O}(N^j)$ space and $\mathcal{O}(N^{i+j+1})$ time.

C.2 The case of $j = 1$

When $j = 1$ the time and space complexities of MPNN are $\mathcal{O}(M)$ and $\mathcal{O}(N)$ respectively. Therefore, the total process for computing all k-tuple takes $\mathcal{O}(N + M)$ space and $\mathcal{O}(MN^i)$ time. As a result, the conclusion in Section 4.1 still holds.

D The Weisfeiler-Lehman Algorithms

In this section we briefly introduce the k-FWL graph isomorphism tests.

D.1 1-WL

The 1-WL test is also known as color refinement and shares similar message passing process with node-level GNNs. To begin with, each node v is assigned with a color c_v. The 1-WL test can then be summarized as follows:

Algorithm 1 1-WL test

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the input graph; c_v for $v \in \mathcal{V}$: initial colors;
Output: the final colors;
1: $c_v^0 \leftarrow c_v$ for all $v \in \mathcal{V}$;
2: **repeat**
3: $\quad \forall v \in \mathcal{V}, c_v^{l+1} \leftarrow \text{hash}(c_v^l, \{\{c_w^l \mid w \in N(v)\}\})$;
4: **until** $\forall v \in \mathcal{V}, c_v^{l+1} = c_v^l$;
5: **return** c_v^l for every $v \in \mathcal{V}$;

Here, $N(v)$ is the set of the neighbors of v in $\mathcal{G}$. The critical part is the hash function hash. It needs to be injective in order to fully express the discriminative power of the 1-WL test.

D.2 k-FWL

The k-FWL test is summarized as follows.

Algorithm 2 k-FWL test

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the input graph; $c_{\mathbf{u}}$ for $\mathbf{u} \in \mathcal{V}^k$: initial colors;
Output: the final colors;
1: $c_{\mathbf{u}}^0 \leftarrow c_{\mathbf{u}}$ for all $\mathbf{u} \in \mathcal{V}^k$;
2: **repeat**
3: $\quad \forall \mathbf{u} \in \mathcal{V}^k, c_{\mathbf{u}}^{l+1} \leftarrow \text{hash}(c_{\mathbf{u}}^l, \{\{c_{\mathbf{v}}^l \mid \mathbf{v} \in N(\mathbf{u})\}\})$;
4: **until** $\forall \mathbf{u} \in \mathcal{V}, c_{\mathbf{u}}^{l+1} = c_{\mathbf{v}}^l$;
5: **return** $c_{\mathbf{u}}^l$ for every $\mathbf{u} \in \mathcal{V}^k$;

The difference between the k-FWL test and the 1-WL is that we now assign a color for each node tuple $\mathbf{u}$ instead of a single node. Generally, the k-FWL test follows the same computation procedure with the 1-WL, but we need to redefine the neighbors, i.e. $N(\mathbf{u})$, of a node tuple $\mathbf{u}$. In the k-FWL test, we set each node pair $\mathbf{u}$ to have $|\mathcal{V}|$ neighbors, with the i-th neighbor being

$$((i, u_2, ..., u_k), (u_1, i, u_3, ..., u_k), ..., (u_1, ..., u_{k-1}, i)).$$

E Different node labeling methods

In this section we discuss about different node labeling methods. Generally, the methods we discuss here all follows the constraints in Section 2:

1. $L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H)$ holds for all $u \in \mathcal{V}_G \Rightarrow \pi(\mathbf{v}) = \mathbf{w}$,
2. π is an isomorphism from $(\mathbf{v}, G)$ to $(\mathbf{w}, H) \Rightarrow \forall u \in \mathcal{V}_G, L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H)$.

E.1 0-1 node labeling

We first introduce the basic 0-1 node labeling method introduced in SEAL. We modify the original 0-1 node labeling for node tuples here.

Given a graph $G = (\mathcal{V}, \mathcal{E})$ and suppose $\mathbf{u} = (u_1, ..., u_k) \in \mathcal{V}^k$ be the target tuple. We define the node labeling mechanism L to be

$$L(v \mid \mathbf{u}, G) = \text{Hash}(\mathbf{1}_{v=u_1}, \mathbf{1}_{v=u_2}, ..., \mathbf{1}_{v=u_k}),$$

where $\mathbf{1}$ is the indicator function. For example in the graph G in Figure 3, we can assign labels $1, 2, ..., k$ to the nodes $u_1, u_2, ..., u_k$ respectively and assign label 0 to the rest of the nodes.

Now we show that the above 0-1 node labeling satisfies the constraints. Given $G = (\mathcal{V}_G, \mathcal{E}_G), H = (\mathcal{V}_H, \mathcal{E}_H)$ and $\mathbf{v} \in \mathcal{V}_G^k, \mathbf{w} \in \mathcal{V}_H^k$,

1. We have
$$\begin{aligned} & L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H) \text{ holds for all } u \in \mathcal{V}_G \\ \implies & L(\pi(v_i) \mid \mathbf{w}, H) = L(\pi(w_i) \mid \mathbf{w}, H) \text{ holds for all } i \in [k] \\ \implies & \pi(\mathbf{v}) = \mathbf{w}. \end{aligned}$$
2. Suppose π is an isomorphism and $\pi(\mathbf{v}) = \mathbf{w}$, then for any $u \in \mathcal{V}_G$, if u is not in $\mathbf{v}$, obviously $\pi(u)$ is also not in $\mathbf{w}$ (because $\pi(\mathbf{v}) = \mathbf{w}$)). Therefore, both u and $\pi(u)$ are assigned with label 0. If u is in $\mathbf{v}$, without loss of generality we suppose $u = v_i$. Then $\pi(\mathbf{v}) = \mathbf{w} \Rightarrow \pi(v_i) = w_i$. Note that $L(v_i \mid \mathbf{v}, G) = L(w_i \mid \mathbf{w}, G)$ for any $i \in [k]$, therefore $\forall u \in \mathcal{V}_G, L(u \mid \mathbf{v}, G) = L(\pi(u) \mid \mathbf{w}, H)$.

E.2 0-1 induced node labeling

There are also some node labeling method that can be induced by the 0-1 node labeling with message passing. In this section we take Distance Encoding (DE) [19] and Double Radius Node Labeling (DRNL) [40] as examples and show how they are induced by 0-1 node labeling with message passing.

A generalized version. We introduce a k-tuple version of DRNL and DE. It's labeling function is

$$L(v \mid \mathbf{u}, G) = \mathrm{Hash}(d(v, u_1), d(v, u_2), ..., d(v, u_k)),$$

where $d(x, y)$ is the shortest distance between x and y. Clearly, DRNL can be computed by MPNNs with 0-1 node labeling. We only need to set $\boldsymbol{h}_v^{(0)} = [\mathrm{Ind}(v, u_1), ..., \mathrm{Ind}(v, u_k)]$ where $\mathrm{Ind}(v, u_i) = 0$ if $v = u_i$ and ∞ otherwise. Therefore it a valid 0-1 node labeling. We let

$$\boldsymbol{h}_v^{(l+1)} = \min\left(\boldsymbol{h}_v^{(l)}, \min\left\{\boldsymbol{h}_u^{(l)} + \mathbf{1} \mid u \in \mathcal{N}(v)\right\}\right),$$

where $\min$ is element-wise. This corresponds to the multi-source shortest path and directly corresponds to the above labeling function L.

F Additional experimental details

F.1 Verifying the expressive power of i-j-$\mathrm{NL}_{\mathrm{MP}}$

We consider popular variants including 1-1-$\mathrm{NL}_{\mathrm{MP}}$, 2-1-$\mathrm{NL}_{\mathrm{MP}}$ and 0-2-$\mathrm{NL}_{\mathrm{MP}}$. We test whether they are able to distinguish:

- Cut edges
- Cut vertices.
- Rook's 4×4 graph and Shrikhande graph

These tasks are with increasing difficulties. The problems of distinguishing cut edges and cue vertices are taken from Zhang et al. [39]. Rook's 4×4 graph and Shrikhande graph are well-known non-isomorphic graphs that cannot be distinguished by 3-WL. The problem of detecting cut vertices is shown to be more complex than detecting cut edges [39]. Note that the problem of detecting cut edges in this paper is more difficult than [39]: they assume that target nodes must be adjacent while we do not. Distinguishing Rook's 4×4 graph and Shrikhande graph is even more complex: it cannot be done by the 2-FWL test. Table 2 lists the results. We can see that the results are consistent with our theoretical findings.

F.2 Experiment configurations

The experiment configurations follow the settings in Chamberlain et al. [6]. Results are taken from [6]. For all tasks we use negative sampling to generate negative targets. At training time the message passing links are equal to the supervision links, while at test time disjoint sets of links are held out that are never seen at training time. We random generate 70-10-20 percent train-val-test splits which is the same as [6]. The metrics are H@100, and is computed by ranking the target links with

Graph type	MPNN	1-1-NL_{MP}	0-2-NL_{MP}	2-1-NL_{MP}
Cut edges	×	✓	✓	✓
Cut vertices	×	×	✓	✓
Rook's vs. Shrikhande	×	×	×	✓

Table 2: Results of detecting different graph patterns.

Figure 5: Logical classifiers over graphs.

randomly sampled negative links by the scores computed by the models. For each target link (s, t) we compute its rank rank_{st}. We then compute $\text{H@100} = \sum_{(s,t)\in\mathcal{D}_{test}} \mathbf{1}_{\text{rank}_{st}}$. The number of the max hops of the paths is 5. We search for hyperparameters using the valid dataset and set learning rate to be 0.0001, dropout 0.5, feature propagation layer 2, weight decay 0. The code is implemented in PyTorch [29] and based on the implementation of Chamberlain et al. [6]. The predictor of the model is designed as $p(u, v) = \text{MLP}(\boldsymbol{h}_u \odot \boldsymbol{h}_v \odot \boldsymbol{h}_{uv})$ where $\boldsymbol{h}_u, \boldsymbol{h}_v$ are node representations of u, v respectively and $\boldsymbol{h}_{uv}$ is the representations of the paths between u, v.

G The problems related to higher-order representations

The problems of learning higher-order representations with GNNs have been extensively studied. However, in fact these methods should be divided into two parts: *learning higher-order representations for predicting node-tuple properties* and *designing higher-order GNNs for learning graph representations*. Although they both learn higher-order representations, they actually study different problems which requires different techniques and solutions.

The goal of the first approaches is to overcome GNNs' inherent weakness on predicting node tuples. This includes the well-known automorphism problem, such as Figure 1. u_1, u_k, v_1, v_k always share the same node representation, so it is impossible to distinguish between them using a vanilla GNN. Therefore, we need to design more powerful GNN variants to give GNNs new abilities to consider the correlation between u_1, u_k and v_1, v_k.

The goal of the second approaches is to improve GNNs' expressive power on graph classification and node classification. Therefore, learning higher-order representations is nothing but a method for strengthening GNNs. In other words, if there are better methods for learning node-level or graph-level representations, higher-order GNNs are not required for this goal.

The difference between the two goals is that, suppose we have a powerful GNN that is able to distinguishes any non-isomorphic nodes (or graphs). Then, the second goal is accomplished: we already reached the upper bound of the expressive power. However, the first goal is far from being accomplished: the automorphism problem still remains. As a result, although higher-order GNNs play an important role in both two problems, the ultimate target and solutions are much different.

H Complete logical description of i-j-NL_{MP}

We first give definition of $\text{FOC}_{k,i}$ by restricting the first-order logic formulas step by step. Our results are similar to [15] in the case of 1-1-NL_{MP}.

First-order logic and graphs. First-order logic is an extension of propositional logic with *predicates* and *quantification*, enabling the evaluation over variables. We focus on the logic classifiers that are expressed in the first-order logic. An example would be:

$$\varphi(x) := \text{Green}(x) \wedge \exists y \left(\text{E}(x, y) \wedge \text{Blue}(y)\right).$$

For more introduction of the first-order logic please refer to textbooks such as Hamilton [13]. The formula $\varphi(x)$ contains one free variable x, therefore we can treat φ as a *classifier*: it takes in one variable x and output true or false based on its evaluation.

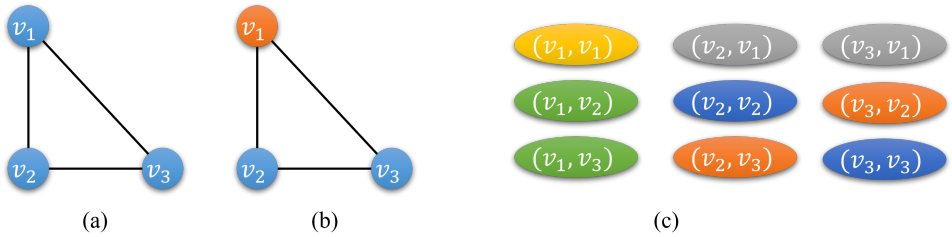


Figure 6: Example of i-j-NLs.

We can evaluate φ over graphs by instituting the *variables* with *nodes*. Consider Figure 5. Each node is regarded as a constant, therefore we have $\mathrm{Green}(u) = \mathrm{True}, \mathrm{Blue}(v) = \mathrm{True}, \mathrm{Blue}(v) = \mathrm{False}, ...$ Additionally the edges between (u, v) and (v, w) are expressed as $\mathrm{E}(u, v) = \mathrm{True}, \mathrm{E}(v, u) = \mathrm{True}, \mathrm{E}(v, w) = \mathrm{True}, \mathrm{E}(w, v) = \mathrm{True}$ but $\mathrm{E}(u, w) = \mathrm{False}, \mathrm{E}(w, u) = \mathrm{False}$. In this manner we have $\varphi(u) = \mathrm{True}, \varphi(v) = \mathrm{False}, \varphi(w) = \mathrm{False}$. Therefore the first-order logic classifiers can be evaluated over graphs.

Formally, given an attributed graph $G = (\mathcal{V}, \mathcal{E})$, we first instantiate the input node labels as unary logical classifiers $\mathrm{Label}(x)$ (e.g., $\mathrm{Green}(x)$). Edges are expressed by a binary logical classifier $\mathrm{E}(x, y)$ where $\mathrm{E}(x, y) = \mathrm{True} \iff (x, y) \in \mathcal{E}$. In addition, if there are multiple types of edges we can simply express them by using different binary predicates $\mathrm{E}_1, \mathrm{E}_2, ...$

FOC_k **formulas.** A well-known restriction of the first-order logic is the FOC_k family which shares a close connection with k-WL tests. In FOC_k, each formula contains at most k variables. This restriction largely reduce the expressive power of the first-order logic. In compensation, FOC_k allows the use of *counting* quantifiers $\exists^N$, which expresses *exists at least N different nodes*.

$\mathrm{FOC}_{k,i}$ **formulas.** The $\mathrm{FOC}_{k,i}$ formulas only add one restriction over the FOC_k: the first i variables of predicates from each formula must be the same. Obviously each predicate is at least i-ary. One straight-forward method to initialize them would be:

$$\mathrm{Label}_p(x_1, x_2, ..., x_m) := \mathrm{Label}(x_p),$$
$$\mathrm{E}_{pq}(x_1, x_2, ..., x_m) := \mathrm{E}(x_p, x_q),$$

where $m \in [i, k], p, q \in [1, m]$. Other initialization methods are also applicable, as long as they exclude the usage of quantifiers.

I An illustration of i-j-NLs

We provide an example of the i-j-NL frameworks applied on a simple graph. Consider the graph (a) in Figure 6, containing three nodes v_1, v_2 and v_3. The node features are represented by colors. Suppose we want to use 1-2-NL to learn the tuple (v_1, v_2, v_3). In the first step, we apply node labeling to v_1, leading to the graph (b), where only v_1 is tagged with a different color. Note that the graph (b) is nothing but a node-featured graph. In the second step, we apply 2-GNNs on the graph (b). We then output the representations of (v_2, v_3) in (c) as the final representation of the target node tuple (v_1, v_2, v_3).

J Recent GNN variants and i-j-NLs

In this section we show how we describe various GNN variants via i-j-NLs.

J.1 Partial node labeling methods

This includes NBFNet [46], ID-GNN [37], etc. We show that they are 1-1-NLs. To learn the target link (s, t), NBFNet first initialize node representations $\boldsymbol{h}_u^0 = \mathrm{Indicator}(s, u)$ for $u \in \mathcal{V}$, where $\mathrm{Indicator}$ is a learned function. It then updates node representations using a MPNN. ID-GNNs use a similar strategy method for learning (s, t), which they refer to as conditional node embeddings. Both

these methods apply node labeling on the source node s [42], and they use the node representation of t to predict the link (s, t), thus they are 1-1-NLs.

J.2 Node labeling methods

This includes SEAL [40] and its variants GraIL [32], INDIGO [20]. SEAL focuses on undirected graphs and learns representations for node sets, but we can easily obtain its directed variant by simply assigning specific orders for the target node tuples, leading to the definition in Section 2, which are also the labeling methods used in GraIL and INDIGO. To learn the target link (u, v), these methods apply node labeling on both s and t, and then aggregate the representations of the nodes or the subgraph as the representation of (u, v). This procedure is equivalent to 2-1-NLs, where we learn representations for all (u, v, w) for $w \in \mathcal{V}$ and then aggregate the representations of $\{(u, v, w) \mid w \in \mathcal{V}\}$ or $\{(u, v, u), (u, v, v)\}$ to represent (u, v).

J.3 Subgraph GNNs

We mainly considers the subgraph GNN frameworks proposed by [30]. Although Qian et al. [30] only apply their methods for graph-level tasks, in fact their frameworks are equivalent to k-1-NLs. Given a graph G, a k-order subgraph GNN first considers N^k different subgraphs, each is tagged by a special k-tuple of nodes $(u_1, ..., u_k)$. The k-order subgraph GNN then runs MPNNs parallel on these subgraphs. Since the subgraph generation methods in [30] can be regarded as special variants of the node labeling methods, this method is equivalent to applying k-1-NLs to G and learn representations for all $(u_1, ..., u_k, v)$ for $u_1, ..., u_k, v \in \mathcal{V}$.

Note that there are also other variants of subgraph GNNs [4]. These variants are slightly more powerful than the variants in [30], and we leave the discussion of these variants as future work. Bevilacqua et al. [4] also proposed to use higher-order GNNs to learn representations for the subgraphs, which is corresponded to general i-j-NLs.