Going Deeper into Permutation-Sensitive Graph Neural Networks

Zhongyu Huang [1 2]   **Yingheng Wang** [3 4]   **Chaozhuo Li** [5]   **Huiguang He** [1 2 6]

Abstract

The invariance to permutations of the adjacency matrix, i.e., graph isomorphism, is an overarching requirement for Graph Neural Networks (GNNs). Conventionally, this prerequisite can be satisfied by the invariant operations over node permutations when aggregating messages. However, such an invariant manner may ignore the relationships among neighboring nodes, thereby hindering the expressivity of GNNs. In this work, we devise an efficient permutation-sensitive aggregation mechanism via permutation groups, capturing pairwise correlations between neighboring nodes. We prove that our approach is strictly more powerful than the 2-dimensional Weisfeiler-Lehman (2-WL) graph isomorphism test and not less powerful than the 3-WL test. Moreover, we prove that our approach achieves the linear sampling complexity. Comprehensive experiments on multiple synthetic and real-world datasets demonstrate the superiority of our model.

1. Introduction

The invariance to permutations of the adjacency matrix, i.e., graph isomorphism, is a key inductive bias for graph representation learning (Murphy et al., 2019a). Graph Neural Networks (GNNs) invariant to graph isomorphism are more amenable to generalization as different orderings of the nodes result in the same representations of the underlying graph. Therefore, many previous studies (Duvenaud et al.,

2015; Kipf & Welling, 2017; Gilmer et al., 2017; Zhang et al., 2018; Xu et al., 2019; Maron et al., 2019b) devote much effort to designing permutation-invariant aggregators to make the overall GNNs *permutation-invariant* (permutation of the nodes of the input graph does not affect the output) or *permutation-equivariant* (permutation of the input permutes the output) to node orderings.

Despite their great success, Kondor et al. (2018) and de Haan et al. (2020) expound that such a permutation-invariant manner may hinder the expressivity of GNNs. Specifically, the strong symmetry of these permutation-invariant aggregators presumes equal statuses of all neighboring nodes, ignoring the relationships among neighboring nodes. Consequently, the central nodes cannot distinguish whether two neighboring nodes are adjacent, failing to recognize and reconstruct the fine-grained substructures within the graph topology. As shown in Figure 1(a), the general Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) can only explicitly reconstruct a star graph from the 1-hop neighborhood, but are powerless to model any connections between neighbors (Chen et al., 2020). To address this problem, some latest advances (Thiede et al., 2021; Zhang & Li, 2021; Zhao et al., 2022; Bevilacqua et al., 2022) propose to use subgraphs or ego-nets to improve the expressive power while preserving the property of permutation-invariance. Unfortunately, they usually suffer from high time or memory complexity when operating on multiple subgraphs (Bevilacqua et al., 2022).

In contrast, the *permutation-sensitive* (as opposed to permutation-invariant) function can be regarded as a "symmetry-breaking" mechanism, which breaks the equal statuses of neighboring nodes. The relationships among neighboring nodes, e.g., the *pairwise correlation* between each pair of neighboring nodes, are explicitly modeled in the permutation-sensitive paradigm. These pairwise correlations help capture whether two neighboring nodes are connected, thereby exploiting the local graph substructures to improve the expressive power. We illustrate a concrete example in Appendix D.

Different permutation-sensitive aggregation functions behave variously when modeling pairwise correlations. GraphSAGE with an LSTM aggregator (Hamilton et al., 2017) in Figure 1(b) is capable of modeling some pairwise correlations among the *sampled subset* of neighboring nodes.

---

[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China [2]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China [3]Department of Electronic Engineering, Tsinghua University, Beijing, China [4]Department of Biomedical Engineering, Johns Hopkins University, Baltimore, Maryland, USA [5]Microsoft Research Asia, Beijing, China [6]Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Beijing, China. Correspondence to: Zhongyu Huang <huangzhongyu2020@ia.ac.cn>, Huiguang He (primary contact) <huiguang.he@ia.ac.cn>.
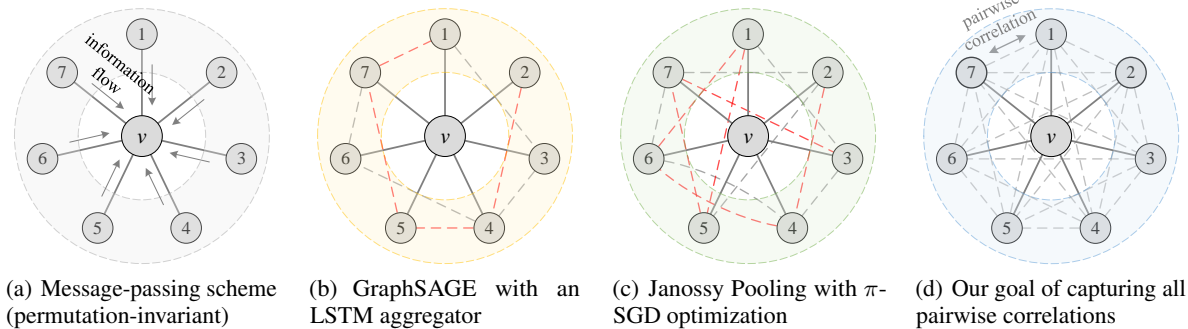
(a) Message-passing scheme (permutation-invariant)

(b) GraphSAGE with an LSTM aggregator

(c) Janossy Pooling with π-SGD optimization

(d) Our goal of capturing all pairwise correlations

Figure 1. Comparison of the pairwise correlations modeled by various aggregation functions in 1-hop neighborhood. Here we illustrate with one central node v and $n = 7$ neighbors. The dashed lines represent the pairwise correlations between neighbors modeled by the aggregators, the real topological connections between neighbors are hidden for clarity. Subfigure (b) shows 2 sampled batches with the neighborhood sample size $k = 5$. Subfigure (c) shows 2 sampled permutations. Dashed lines - - and - - in (b)/(c) denote different batches/permutations.

Janossy Pooling with the π-SGD strategy (Murphy et al., 2019b) in Figure 1(c) samples random permutations of *all* neighboring nodes, thus modeling pairwise correlations more efficiently. The number of modeled pairwise correlations is proportional to the number of sampled permutations. After sampling permutations with a costly nonlinear complexity of $\mathcal{O}(n \ln n)$ (see Appendix K for detailed analysis), all the pairwise correlations between n neighboring nodes can be modeled and all the possible connections are covered.

In fact, previous works (Murphy et al., 2019b;a) have explored that incorporating permutation-sensitive functions into GNNs is indeed an effective way to improve their expressive power. Janossy Pooling (Murphy et al., 2019b) and Relational Pooling (Murphy et al., 2019a) both design the most powerful GNN models by exploiting permutation-sensitive functions to cover all $n!$ possible permutations. They explicitly learn all representations of the underlying graph with possible $n!$ node orderings to guarantee the permutation-invariance and generalization capability, overcoming the limited generalization of permutation-sensitive GNNs (Vignac et al., 2020). However, the complete modeling of all $n!$ permutations also leads to an intractable computational complexity $\mathcal{O}(n!)$. Thus, we expect to design a powerful yet efficient GNN, which can guarantee the expressive power, and significantly reduce the complexity with a minimal loss of generalization capability.

Different from explicitly modeling all $n!$ permutations, we propose to sample a small number of representative permutations to cover all $n(n-1)/2$ pairwise correlations (as shown in Figure 1(d)) by the permutation-sensitive functions. Accordingly, the permutation-invariance is approximated by the invariance to pairwise correlations. Moreover, we mathematically analyze the complexity of permutation sampling and reduce it from $\mathcal{O}(n \ln n)$ to $\mathcal{O}(n)$ via a well-designed *Permutation Group* (PG). Based on the proposed permuta-

tion sampling strategy, we then devise an aggregation mechanism and theoretically prove that its expressivity is strictly more powerful than the 2-WL test and not less powerful than the 3-WL test. Thus, our model is capable of significantly reducing the computational complexity while guaranteeing the expressive power. To the best of our knowledge, our approach achieves the lowest time and space complexity among all the GNNs beyond 2-WL test so far.

2. Related Work

Permutation-Sensitive GNNs. Loukas (2020) first analyzes that it is necessary to sacrifice the permutation-invariance and permutation-equivariance of MPNNs to improve their expressive power when nodes lose discriminative attributes. However, only a few models (GraphSAGE with LSTM aggregators (Hamilton et al., 2017), RP with π-SGD (Murphy et al., 2019b), CLIP (Dasoulas et al., 2020)) are permutation-sensitive GNNs. These studies provide either theoretical proofs or empirical results that their approaches can capture some substructures, especially triangles, which can be served as special cases of our Theorem 4. Despite their powerful expressivity, the nonlinear complexity of sampling or coloring limits their practical application.

Expressive Power of GNNs. Xu et al. (2019) and Morris et al. (2019) first investigate the GNNs' ability to distinguish non-isomorphic graphs and demonstrate that the traditional message-passing paradigm (Gilmer et al., 2017) is at most as powerful as the 2-WL test (Weisfeiler & Leman, 1968), which cannot distinguish some graph pairs like regular graphs with identical attributes. In order to theoretically improve the expressive power of the 2-WL test, a direct way is to equip nodes with distinguishable attributes, e.g., identifier (Murphy et al., 2019a; Loukas, 2020), port numbering (Sato et al., 2019), coloring (Sato et al., 2019; Dasoulas

et al., 2020), distance feature (Li et al., 2020; You et al., 2021), and random feature (Sato et al., 2021; Abboud et al., 2021). Another series of researches (Morris et al., 2019; Maron et al., 2019b;a; Azizian & Lelarge, 2021) consider high-order relations to design more powerful GNNs but suffer from high computational complexity when handling high-order tensors and performing global computations on the graph. Some pioneering works (de Haan et al., 2020; Thiede et al., 2021) use the automorphism group of local subgraphs to obtain more expressive representations and overcome the problem of global computations, but their preprocessing stages still require solving the NP-hard subgraph isomorphism problem. Recent studies (Sato et al., 2019; Garg et al., 2020; Loukas, 2020; Tahmasebi et al., 2020) also characterize the expressive power of GNNs from the perspectives of what they cannot learn.

Leveraging Substructures for Learning Representations. Previous efforts mainly focused on the isomorphism tasks, but did little work on understanding their capacity to capture and exploit the graph substructure. Recent studies (Chen et al., 2020; de Haan et al., 2020; Vignac et al., 2020; Sato et al., 2021; Balcilar et al., 2021; Barceló et al., 2021; Bouritsas et al., 2022) show that the expressive power of GNNs is highly related to the local substructures in graphs. Chen et al. (2020) demonstrate that the substructure counting ability of GNN architectures not only serves as an intuitive theoretical measure of their expressive power but also is highly relevant to practical tasks. Barceló et al. (2021) and Bouritsas et al. (2022) propose to incorporate some handcrafted subgraph features to improve the expressive power, while they require expert knowledge to select task-relevant features. Several latest advances (Vignac et al., 2020; Balcilar et al., 2021; Bodnar et al., 2021b;a) have been made to enhance the standard MPNNs by leveraging high-order structural information while retaining the locality of message-passing. However, the complexity issue has not been satisfactorily solved because they introduce memory/time-consuming context matrices (Vignac et al., 2020), eigenvalue decomposition (Balcilar et al., 2021), and lifting transformation (Bodnar et al., 2021b;a) in pre-processing.

Relations to Our Work. Some crucial differences between related works (Hamilton et al., 2017; Murphy et al., 2019a; Dasoulas et al., 2020; Chen et al., 2020; Vignac et al., 2020; Sato et al., 2021; Balcilar et al., 2021; Bodnar et al., 2021b) and ours can be summarized as follows: (*i*) we propose to design powerful permutation-sensitive GNNs while approximating the property of permutation-invariance, balancing the expressivity and computational efficiency; (*ii*) our approach realizes the linear complexity of permutation sampling and reaches the theoretical lower bound; (*iii*) our approach can directly learn substructures from data instead of pre-computing or strategies based on handcrafted

structural features. We also provide detailed discussions in Appendix H.3 for (Hamilton et al., 2017), K.2 for (Murphy et al., 2019b;a), and L.1 for (Bodnar et al., 2021b;a).

3. Designing Powerful Yet Efficient GNNs via Permutation Groups

In this section, we begin with the analysis of theoretically most powerful but intractable GNNs. Then, we propose a tractable strategy to achieve linear permutation sampling and significantly reduce the complexity. Based on this strategy, we design our permutation-sensitive aggregation mechanism via permutation groups. Furthermore, we mathematically analyze the expressivity of permutation-sensitive GNNs and prove that our proposed model is more powerful than the 2-WL test and not less powerful than the 3-WL test via incidence substructure counting.

3.1. Preliminaries

Let $G = (\mathcal{V}, \mathcal{E}) \in \mathcal{G}$ be a graph with vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ and edge set $\mathcal{E}$, directed or undirected. Let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ be the adjacency matrix of G. For a node $v \in \mathcal{V}$, d_v denotes its degree, i.e., the number of 1-hop neighbors of node v, which is equivalent to n in this section for simplicity. Suppose these n neighboring nodes of the central node v are randomly numbered as $u_1, \ldots, u_n$ (also abbreviated as $1, \ldots, n$ in the following), the set of neighboring nodes is represented as $\mathcal{N}(v)$ (or $S = [n] = \{1, \ldots, n\}$). Given a set of graphs $\{G_1, G_2, \ldots, G_M\} \subseteq \mathcal{G}$, each graph G has a label y_G. Our goal is to learn a representation vector $\boldsymbol{h}_G$ of the entire graph G and classify it into the correct category from C classes. In this paper, we use the normal G to denote a *graph* and the Gothic $\mathfrak{G}$ to denote a *group*. The necessary backgrounds of graph theory and group theory are attached in Appendixes A and B. The rigorous definition of the k-WL test is provided in Appendix C.

3.2. Theoretically Most Powerful GNNs

Relational Pooling (RP) (Murphy et al., 2019a) proposes the theoretically most powerful permutation-invariant model by averaging over all permutations of the nodes, which can be formulated as follows:

$$\boldsymbol{h}_G = \frac{1}{|\mathcal{S}_N|} \sum_{\pi \in \mathcal{S}_N} \vec{f}(\boldsymbol{h}_{\pi v_1}, \boldsymbol{h}_{\pi v_2}, \cdots, \boldsymbol{h}_{\pi v_N}) \quad (1)$$

where $\pi v_i (i = 1, \ldots, N)$ denotes the result of acting $\pi \in \mathcal{S}_N$ on $v_i \in \mathcal{V}$, $\mathcal{S}_N$ is the symmetric group on the set $[N]$ (or $\mathcal{V}$), $\vec{f}$ is a sufficiently expressive (possibly permutation-sensitive) function, $\boldsymbol{h}_{v_i}$ is the feature vector of node v_i.

The permutation-sensitive functions, especially sequential models, are capable of modeling the *k-ary dependency* (Murphy et al., 2019b;a) among k input nodes. Meanwhile, the

different input node orderings will lead to a total number of $k!$ different k-ary dependencies. These k-ary dependencies indicate the relations and help capture the topological connections among the corresponding k nodes, thereby exploiting the substructures within these k nodes to improve the expressive power of GNN models. For instance, the expressivity of Eq. (1) is mainly attributed to the modeling of all possible N-ary dependencies (*full-dependencies*) among all N nodes, which can capture all graphs isomorphic to G. However, it is intractable and practically prohibitive to model all permutations ($N!$ N-ary dependencies) due to the extremely high computational cost. Thus, it is necessary to design a tractable strategy to reduce the computational cost while maximally preserving the expressive power.

3.3. Permutation Sampling Strategy

Intuitively, the simplest way is to replace N-ary dependencies with *2-ary dependencies*, i.e., the *pairwise correlations* in Section 1. Moreover, since the inductive bias of locality results in lower complexity on sparse graphs (Battaglia et al., 2018; Balcilar et al., 2021), we restrict the permutation-sensitive functions to aggregate information and model the 2-ary dependencies in the 1-hop neighborhoods. Thus, we will further discuss how to model all 2-ary dependencies between n neighboring nodes with the lowest sampling complexity $\mathcal{O}(n)$.

Suppose n neighboring nodes are arranged as a ring, we define this ring as an *arrangement*. An initial arrangement can be simply defined as $1 - 2 - \cdots - n - 1$, including an n-ary dependency $\{1 - 2 - \cdots - n - 1\}$ and n 2-ary dependencies $\{1 - 2, 2 - 3, \cdots, n - 1\}$. Since a permutation adjusts the node ordering in the arrangement, we can use a permutation to generate a new arrangement, which corresponds to a new n-ary dependency covering n 2-ary dependencies. The following theorem provides a lower bound of the number of arrangements to cover all 2-ary dependencies.

Theorem 1. *Let $n(n \geq 4)$ denote the number of 1-hop neighboring nodes around the central node v. There are $\lfloor (n-1)/2 \rfloor$ kinds of arrangements in total, satisfying that their corresponding 2-ary dependencies are disjoint. Meanwhile, after at least $\lfloor n/2 \rfloor$ arrangements (including the initial one), all 2-ary dependencies have been covered at least once.*

We first give a sketch of the proof, and the detailed proof of Theorem 1 is provided in Appendix E. Construct a simple undirected graph $G' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}'$ denotes the n neighboring nodes (abbreviated as nodes in the following), and $\mathcal{E}'$ represents an edge set in which each edge indicates the corresponding 2-ary dependency has been covered in some arrangements. Each arrangement corresponds to a Hamiltonian cycle in graph G'. In addition, we define the

following permutation σ to generate new arrangements:

$$\sigma = \begin{cases} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & n-1 & n \\ 1 & 4 & 2 & 6 & 3 & \cdots & n & n-2 \end{pmatrix}, n \text{ is odd,} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & n-1 & n \\ 3 & 1 & 5 & 2 & \cdots & n & n-2 \end{pmatrix}, n \text{ is even.} \end{cases} \tag{2}$$

After performing the permutation σ once, a new arrangement is generated and a Hamiltonian cycle is constructed. Since every pair of nodes can form a 2-ary dependency, covering all 2-ary dependencies is equivalent to constructing a complete graph K_n. Besides, as a K_n has $n(n-1)/2$ edges and each Hamiltonian cycle has n edges, a K_n can only be constructed with at least $\lceil n(n-1)/2n \rceil = \lceil (n-1)/2 \rceil = \lfloor n/2 \rfloor$ Hamiltonian cycles. It can be proved that after performing the permutation σ for $\lfloor n/2 \rfloor - 1 = \mathcal{O}(n)$ times in succession (excluding the initial one), all 2-ary dependencies are covered at least once.

Note that Theorem 1 has the constraint $n \geq 4$ because all 2-ary dependencies have already been covered in the initial arrangement when $1 < n < 4$, and there is only a single node when $n = 1$. If $n = 2, 3, 4$, $\sigma = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\begin{pmatrix} 2 & 3 \end{pmatrix}$, $\begin{pmatrix} 1 & 3 & 4 & 2 \end{pmatrix}$, respectively (the case of $n = 1$ is trivial). Thus the permutation σ defined in Theorem 1 is available for an arbitrary n, while Eq. (2) shows the general case with a large n.

According to the ordering of n neighboring nodes in the arrangement, we can apply a permutation-sensitive function to model an n-ary dependency among these n nodes while covering n 2-ary dependencies. Since the input orderings $a \to b$ and $b \to a$ lead to different results in the permutation-sensitive function, these dependencies and the corresponding Hamiltonian cycles (the solid arrows in Figure 2) are modeled in a directed manner. We continue performing the permutation σ for $\lfloor n/2 \rfloor$ times successively to get additional $\lfloor n/2 \rfloor$ arrangements (the dashed lines in Figure 2) and reversely directed Hamiltonian cycles (not shown in Figure 2). After the bi-directional modeling, edges in Hamiltonian cycles are transformed into undirected edges. Figure 2 briefly illustrates the above process when $n = 5$ to 8. In conclusion, all 2-ary dependencies can be modeled in an undirected manner by the tailored permutations. The number of permutations is n if n is even and $(n - 1)$ if n is odd, ensuring the linear sampling complexity $\mathcal{O}(n)$.

In fact, all permutations above form a permutation group. In order to incorporate the strategy proposed by Theorem 1 into the aggregation process of GNN, we propose to use the permutation group and group action, defined as follows.

Lemma 2. *For the permutation σ of n indices, $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-2}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_{n-1}$ if n is odd. And $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-1}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_n$ if n is even.*
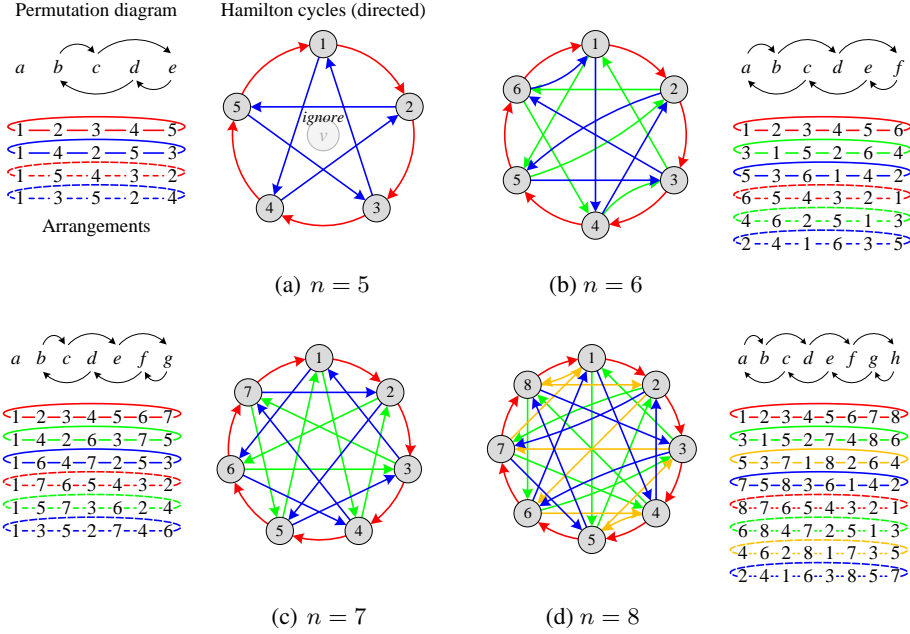
Figure 2. Modeling all pairwise correlations between n neighboring nodes via permutations. Subfigures (a) to (d) characterize the cases when $n = 5$ to 8 (ignoring the central node v). The monochrome permutation diagram illustrates the mapping process of permutation σ, where the directed arc $a \to b$ indicates that moving a to the original position of b. All arrangements generated by σ^i are shown in color below the diagram. The first and the last $\lfloor n/2 \rfloor$ arrangements are marked with solid and dashed lines, respectively. Solid and dashed lines with the same color indicate that they correspond to a pair of bi-directional Hamiltonian cycles. Only the Hamiltonian cycles corresponding to the first $\lfloor n/2 \rfloor$ arrangements are displayed for clarity. For a further explanation and the relationships among Theorem 1, Lemma 2, Corollary 3, Figure 2, and Eq. (3), please refer to Appendix J.1 and Figure 8.

Corollary 3. *The map $\alpha : \mathfrak{G} \times S \to S$ denoted by $(g, s) \mapsto gs$ is a group action of $\mathfrak{G}$ on S.*

To better illustrate the results of Lemma 2 and Corollary 3, the detailed discussion and diagram are attached in Appendix G. Next, we apply the permutation group and group action to design our permutation-sensitive aggregation mechanism.

3.4. Network Architecture

Without loss of generality, we apply the widely-used Recurrent Neural Networks (RNNs) as the permutation-sensitive function to model the dependencies among neighboring nodes. Let the group elements (i.e., permutations) in $\mathfrak{G}$ act on S, our proposed strategy in Section 3.3 is formulated as:

$$\boldsymbol{h}_v^{(k)} = \sum_{g \in \mathfrak{G}} \text{RNN} \left(\boldsymbol{h}_{gu_1}^{(k-1)}, \boldsymbol{h}_{gu_2}^{(k-1)}, \cdots, \boldsymbol{h}_{gu_n}^{(k-1)}, \boldsymbol{h}_{gu_1}^{(k-1)} \right)$$
$$+ \boldsymbol{W}_{\text{self}}^{(k-1)} \boldsymbol{h}_v^{(k-1)}, u_{1:n} \in \mathcal{N}(v) \quad (3)$$

where $gu_i (i = 1, \ldots, n)$ denotes the result of acting $g \in \mathfrak{G}$ on $u_i \in S$, and $\boldsymbol{h}_v^{(k)} \in \mathbb{R}^{d_k}$ is the feature vector of central node v at the k-th layer. We provide more discussion on the groups and model variants in Appendixes J.2 and J.3. Eq. (3)

takes advantage of the locality and permutation group $\mathfrak{G}$ to simplify the group actions in Eq. (1), which acts the symmetric group $\mathcal{S}_N$ on vertex set $\mathcal{V}$, thereby avoiding the complete modeling of $N!$ permutations. Meanwhile, Eq. (3) models all 2-ary dependencies and achieves the invariance to 2-ary dependencies. Thus, we can conclude that Eq. (3) realizes the efficient approximation of permutation-invariance with low complexity. In practice, we merge the central node v into RNN for simplicity:

$$\boldsymbol{h}_v^{(k)} = \sum_{g \in \mathfrak{G}} \text{RNN} \left(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{h}_{gu_1}^{(k-1)}, \boldsymbol{h}_{gu_2}^{(k-1)}, \cdots, \right.$$
$$\left. \boldsymbol{h}_{gu_n}^{(k-1)}, \boldsymbol{h}_v^{(k-1)} \right), u_{1:n} \in \mathcal{N}(v) \quad (4)$$

Then, we apply a READOUT function (e.g., SUM($\cdot$)) to obtain the graph representation $\boldsymbol{h}_G^{(k)}$ at the k-th layer and combine the representations learned by different layers to get the score $\boldsymbol{s}$ for classification:

$$\boldsymbol{h}_G^{(k)} = \sum_{v \in \mathcal{V}} \boldsymbol{h}_v^{(k)}, \quad \boldsymbol{s} = \sum_k \boldsymbol{W}^{(k)} \boldsymbol{h}_G^{(k)} \quad (5)$$

here $\boldsymbol{W}^{(k)} \in \mathbb{R}^{C \times d_k}$ represents a learnable scoring matrix for the k-th layer. Finally, we input score $\boldsymbol{s}$ to the softmax function and obtain the predicted class of graph G.

Table 1. Memory and time complexity per layer.

Model	Memory	Time complexity
GIN	$\Theta(Nc)$	$\Theta(Mc + Nc^2)$
MPNN	$\Theta(Nc)$	$\Theta(Mc^2)$
Fast SMP	$\Theta(N^2c)$	$\Theta(MNc + N^2c^2)$
SMP	$\Theta(N^2c)$	$\Theta(MNc^2)$
PPGN	$\Theta(N^2c)$	$\Theta(N^3c + N^2c^2)$
3-WL-GNN	$\Theta(N^3c)$	$\Theta(N^4c + N^3c^2)$
Ours (serial)	$\Theta(Nc)$	$\Theta(N\Delta^2c^2)$
Ours (parallel)	$\Theta(N\Delta c)$	$\Theta(N\Delta c^2)$

Complexity. We briefly analyze the computational complexity of Eq. (3). Suppose the input and output dimensions are both c for each layer, let Δ denote the maximum degree of graph G. In the worst-case scenario, Eq. (3) requires summing over Δ terms processed in a serial manner. Since there is no interdependence between these Δ terms, they can also be computed in a parallel manner with the time complexity of $\Theta(\Delta c^2)$ (caused by RNN computation), while sacrificing the memory to save time. Let M denote the number of edges. Table 1 compares our approach with other powerful GNNs on the per-layer space and time complexity. The results of baselines are taken from Vignac et al. (2020). Since the complexity analysis of GraphSAGE (Hamilton et al., 2017), MPSN (Bodnar et al., 2021b), and CWN (Bodnar et al., 2021a) involves many other notations, we analyze GraphSAGE in Appendix H.3, and MPSN and CWN in Appendix L.1. In a nutshell, our approach theoretically outperforms other powerful GNNs in terms of time and space complexity, even being on par with MPNN.

3.5. Expressivity Analysis

In this subsection, we theoretically analyze the expressive power of a typical category of permutation-sensitive GNNs, i.e., GNNs with RNN aggregators (Theorem 4), and that of our proposed PG-GNN (Proposition 5). We begin with GIN (Xu et al., 2019), which possesses the equivalent expressive power as the 2-WL test (Xu et al., 2019; Azizian & Lelarge, 2021). In fact, the variants of GIN can be recovered by GNNs with RNN aggregators (see Appendix I for details), which implies that this category of permutation-sensitive GNNs can be at least as powerful as the 2-WL test. Next, we explicate why they go beyond the 2-WL test from the perspective of substructure counting.

Triangular substructures are rich in various networks, and counting triangles is an important task in network analysis (Al Hasan & Dave, 2018). For example, in social networks, the formation of a triangle indicates that two people with a common friend will also become friends (Mitzenmacher & Upfal, 2017). A triangle $\triangle u_i v u_j$ is *incident* to the node v if u_i and u_j are adjacent and node v is their common neighbor.

We define the triangle $\triangle u_i v u_j$ as an *incidence triangle* over node v (also u_i and u_j), and denote the number of incidence triangles over node v as τ_v. Formally, the number of incidence triangles over each node in an **undirected** graph can be calculated as follows (proof and discussion for the directed graph are provided in Appendix H.1):

$$\boldsymbol{\tau} = \frac{1}{2}\boldsymbol{A}^2 \odot \boldsymbol{A} \cdot \mathbf{1}_N \tag{6}$$

where $\boldsymbol{\tau} \in \mathbb{R}^N$ and its i-th element τ_i represents the number of incidence triangles over node i, $\odot$ denotes the element-wise product (i.e., Hadamard product), $\mathbf{1}_N = (1, 1, \cdots, 1)^\top \in \mathbb{R}^N$ is a sum vector.

Besides the WL-test, the capability of counting graph substructures also characterizes the expressive power of GNNs (Chen et al., 2020). Thus, we verify the expressivity of permutation-sensitive GNNs by evaluating their abilities to count triangles.

Theorem 4. *Let $x_v, \forall v \in \mathcal{V}$ denote the feature inputs on graph $G = (\mathcal{V}, \mathcal{E})$, and $\mathbb{M}$ be a general GNN model with RNN aggregators. Suppose that x_v is initialized as the degree d_v of node v, and each node is distinguishable. For any $0 < \epsilon \le 1/8$ and $0 < \delta < 1$, there exists a parameter setting Θ for $\mathbb{M}$ so that after $\mathcal{O}\left(\frac{d_v(2d_v+\tau_v)\mathfrak{t}}{d_v+\tau_v}\right)$ samples,*

$$\Pr\left(\left|\frac{z_v}{\tau_v} - 1\right| \le \epsilon\right) \ge 1 - \delta, \forall v \in \mathcal{V},$$

where $z_v \in \mathbb{R}$ is the final output value generated by $\mathbb{M}$ and τ_v is the number of incidence triangles.

Detailed proof can be found in Appendix H.2. Theorem 4 concludes that, for any graph, if the input node features are node degrees and nodes are distinguishable, there exists a parameter setting for a *general* GNN with RNN aggregators such that it can approximate the number of incidence triangles to arbitrary precision for every node. Since 2-WL and MPNNs cannot count triangles (Chen et al., 2020), we conclude that this category of permutation-sensitive GNNs is more powerful. However, the required samples are related to τ_v and proportional to the mixing time $\mathfrak{t}$ (see Appendix H.2), leading to a practically prohibitive aggregation complexity. Many existing permutation-sensitive GNNs like GraphSAGE with LSTM and RP with π-SGD suffer from this issue (see Appendixes H.3 and K.2 for more discussion).

On the contrary, our approach can estimate the number of incidence triangles in linear sampling complexity $\mathcal{O}(n) = \mathcal{O}(d_v)$. According to the definition of incidence triangles and the fact that they always appear within v's 1-hop neighborhood, we know that *the number of connections between the central node v's neighboring nodes is equivalent to the number of incidence triangles over v*. Meanwhile, Theorem

1 and Eq. (3) ensure that all 2-ary dependencies between n neighboring nodes are modeled with $\mathcal{O}(n)$ sampling complexity. These dependencies capture the information of whether two neighboring nodes are connected, thereby estimating the number of connections and counting incidence triangles in linear sampling complexity.

Recently, Balcilar et al. (2021) claimed that the trace (tr) and Hadamard product ($\odot$) operations are crucial requirements to go further than 2-WL to reach 3-WL from the perspective of Matrix Language (Geerts, 2021). In fact, for any two neighbors u_i and u_j of the central node v, the locality and 2-ary dependency of Eq. (3) introduce the information of $\boldsymbol{A}^2$ (i.e., $u_i - v - u_j$) and $\odot \boldsymbol{A}$ (i.e., $u_i \overset{?}{-} u_j$), respectively. Thus Eq. (3) can mimic Eq. (6) to count incidence triangles. Moreover, we also prove that $\mathbf{1}_N^\top \boldsymbol{\tau} = \frac{1}{2}\mathrm{tr}(\boldsymbol{A}^3)$ (see Appendix H.1 for details), which indicates that PG-GNN can realize the trace (tr) operation when we use SUM($\cdot$) or MEAN($\cdot$) (i.e., $\mathbf{1}_N$) as the graph-level READOUT function. Note that even though MPNNs and 2-WL test are equipped with distinguishable attributes, they still have difficulty performing triangle counting since they cannot implement the trace or Hadamard product operations (Balcilar et al., 2021).

Beyond the incidence triangle, we can also leverage 2-ary dependencies of $u_i \overset{?}{-} u_j$, $u_i \overset{?}{-} u_k$, and $u_j \overset{?}{-} u_k$ to discover the *incidence 4-clique* $\bigoplus v u_i u_j u_k$, which is completely composed of triangles and only appears within v's 1-hop neighborhood. In this way, the expressive power of PG-GNN can be further improved by its capability of counting incidence 4-cliques. As illustrated in Figure 7, these incidence 4-cliques help distinguish some pairs of non-isomorphic strongly regular graphs while the 3-WL test fails. Consequently, the expressivity of our model is guaranteed to be not less powerful than 3-WL[1].

From the analysis above, we confirm the expressivity of PG-GNN as follows. The strict proof and more detailed discussion on PG-GNN and 3-WL are provided in Appendix I.

Proposition 5. *PG-GNN is strictly more powerful than the 2-WL test and not less powerful than the 3-WL test.*

4. Experiments

In this section, we evaluate PG-GNN on multiple synthetic and real-world datasets from a wide range of domains. Dataset statistics and details are presented in Appendix M.1. The hyper-parameter search space and final hyper-parameter configurations are provided in Appendix M.2. Computing infrastructures can be found in Appendix M.3. The

---

[1]"A is no/not less powerful than B" means that there exists a pair of non-isomorphic graphs such that A can distinguish but B cannot. The terminology "no/not less powerful" used here follows the standard definition in the literature (Chen et al., 2020; Bodnar et al., 2021b;a; Zhao et al., 2022).

Table 2. Results (measured by MAE) on incidence triangle count.

Model	Erdős-Rényi graph	Random regular graph
GCN	0.599 ± 0.006	0.500 ± 0.012
GraphSAGE	0.118 ± 0.005	0.127 ± 0.011
GIN	0.219 ± 0.016	0.342 ± 0.005
rGIN	0.194 ± 0.009	0.325 ± 0.006
RP	0.058 ± 0.006	0.161 ± 0.003
LRP	0.023 ± 0.011	0.037 ± 0.019
PG-GNN	$\mathbf{0.019 \pm 0.002}$	$\mathbf{0.027 \pm 0.001}$

code is publicly available at https://github.com/zhongyu1998/PG-GNN.

4.1. Counting Substructures in Random Graphs

We conduct synthetic experiments of counting incidence substructures (triangles and 4-cliques) on two types of random graphs: Erdős-Rényi random graphs and random regular graphs (Chen et al., 2020). The incidence substructure counting task is designed on the node level, which is more rigorous than traditional graph-level counting tasks. Table 2 summarizes the results measured by Mean Absolute Error (MAE, lower is better) for incidence triangle counting. We report the average and standard deviation of testing MAEs over 5 runs with 5 different seeds. In addition, the testing MAEs of PG-GNN on ER and random regular graphs are 0.029 ± 0.002 and 0.023 ± 0.001 for incidence 4-clique counting, respectively. Overall, the negligible MAEs of our model support our claim that PG-GNN is powerful enough for counting incidence triangles and 4-cliques.

Another phenomenon is that permutation-sensitive GNNs consistently outperform permutation-invariant GNNs on substructure counting tasks. This indicates that permutation-sensitive GNNs are capable of learning these substructures directly from data, without explicitly assigning them as node features, but the permutation-invariant counterparts like GCN and GIN fail. Therefore, permutation-sensitive GNNs can implicitly leverage the information of characteristic substructures in representation learning and thus benefit real-world tasks in practical scenarios.

4.2. Real-World Benchmarks

Datasets. We evaluate our model on 7 real-world datasets from various domains. PROTEINS and NCI1 are bioinformatics datasets; IMDB-BINARY, IMDB-MULTI, and COLLAB are social network datasets. They are all popular graph classification tasks from the classical TUDataset (Morris et al., 2020). We follow Xu et al. (2019) to create the input features for each node. More specifically, the input node features of bioinformatics graphs are categorical node labels, and the input node features of social networks are

Table 3. Results (measured by accuracy: %) on TUDataset.

Model	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB
WL	75.0 ± 3.1	$\mathbf{86.0 \pm 1.8}$	73.8 ± 3.9	50.9 ± 3.8	78.9 ± 1.9
DGCNN	75.5 ± 0.9	74.4 ± 0.5	70.0 ± 0.9	47.8 ± 0.9	73.8 ± 0.5
IGN	76.6 ± 5.5	74.3 ± 2.7	72.0 ± 5.5	48.7 ± 3.4	78.4 ± 2.5
GIN	76.2 ± 2.8	82.7 ± 1.7	75.1 ± 5.1	52.3 ± 2.8	80.2 ± 1.9
PPGN	$\mathbf{77.2 \pm 4.7}$	83.2 ± 1.1	73.0 ± 5.8	50.5 ± 3.6	80.7 ± 1.7
CLIP	77.1 ± 4.4	N/A	76.0 ± 2.7	52.5 ± 3.0	N/A
NGN	71.7 ± 1.0	82.7 ± 1.4	74.8 ± 2.0	51.3 ± 1.5	N/A
WEGL	76.5 ± 4.2	N/A	75.4 ± 5.0	52.3 ± 2.9	80.6 ± 2.0
SIN	76.5 ± 3.4	82.8 ± 2.2	75.6 ± 3.2	52.5 ± 3.0	N/A
CIN	77.0 ± 4.3	83.6 ± 1.4	75.6 ± 3.7	52.7 ± 3.1	N/A
PG-GNN	76.8 ± 3.8	82.8 ± 1.3	$\mathbf{76.8 \pm 2.6}$	$\mathbf{53.2 \pm 3.6}$	$\mathbf{80.9 \pm 0.8}$

Table 4. Results and running times on MNIST and ZINC.

Model	MNIST		ZINC	
	Accuracy ↑	Time / Epoch	MAE ↓	Time / Epoch
GraphSAGE	97.31 ± 0.10	113.12s	0.468 ± 0.003	3.74s
GatedGCN	97.34 ± 0.14	128.79s	0.435 ± 0.011	5.76s
GIN	96.49 ± 0.25	39.22s	0.387 ± 0.015	2.29s
3-WL-GNN	95.08 ± 0.96	1523.20s	0.407 ± 0.028	286.23s
Ring-GNN	91.86 ± 0.45	2575.99s	0.512 ± 0.023	327.65s
PPGN	N/A	N/A	0.256 ± 0.054	334.69s
Deep-LRP	N/A	N/A	$\mathbf{0.223 \pm 0.008}$	72s
PNA	97.41 ± 0.16	N/A	0.320 ± 0.032	N/A
PG-GNN	$\mathbf{97.51 \pm 0.07}$	82.60s	0.282 ± 0.011	6.92s

node degrees. All the input features are encoded in a one-hot manner. In addition, MNIST is a computer vision dataset for the graph classification task, and ZINC is a chemistry dataset for the graph regression task. They are both modern benchmark datasets, and we obtain the features from the original paper (Dwivedi et al., 2020), but *do not* take edge features into account. We summarize the statistics of all 7 real-world datasets in Table 7, and more details about these datasets can be found in Appendix M.1.

Evaluations. For TUDataset, we follow the same data split and evaluation protocol as Xu et al. (2019). We perform 10-fold cross-validation with random splitting and report our results (the average and standard deviation of testing accuracies) at the epoch with the best average accuracy across the 10 folds. For MNIST and ZINC, we follow the same data splits and evaluation metrics as Dwivedi et al. (2020), please refer to Appendix M.1 for more details. The experiments are performed over 4 runs with 4 different seeds, and we report the average and standard deviation of testing results.

Baselines. We compare our PG-GNN with multiple state-of-the-art baselines: WL (Shervashidze et al., 2011), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GatedGCN (Bresson & Laurent, 2017), DGCNN (Zhang et al., 2018), 3-WL-GNN (Morris et al., 2019), IGN (Maron et al., 2019b), GIN (Xu et al., 2019), RP (Murphy et al., 2019a), PPGN (Maron et al., 2019a), Ring-GNN (Chen et al., 2019), CLIP (Dasoulas et al., 2020), NGN (de Haan et al., 2020), (Deep-)LRP (Chen et al., 2020), PNA (Corso et al., 2020), SMP (Vignac et al., 2020), rGIN (Sato et al., 2021), WEGL (Kolouri et al., 2021), SIN (Bodnar et al., 2021b), and CIN (Bodnar et al., 2021a).

Results and Analysis. Tables 3 and 4 present a summary of the results. The results of baselines in Table 3 are taken from their original papers, except WL taken from Xu et al. (2019), and IGN from Maron et al. (2019a) for preserving the same evaluation protocol. The results of baselines in Table 4 are taken from Dwivedi et al. (2020), except PPGN and Deep-LRP are taken from Chen et al. (2020), and PNA

from Corso et al. (2020). Obviously, our model achieves outstanding performance on most datasets, even outperforming competitive baselines by a considerable margin.

From Tables 3 and 4, we notice that our model significantly outperforms other approaches on all social network datasets, but slightly underperforms main baselines on molecular datasets such as NCI1 and ZINC. Recall that in Section 3.5, we demonstrate that our model is capable of estimating the number of incidence triangles. The capability of counting incidence triangles benefits our model on graphs with many triangular substructures, e.g., social networks. However, triangles rarely exist in chemical compounds (verified in Table 7) due to their instability in the molecular structures. Thus our model achieves sub-optimal performance on molecular datasets. Suppose we extend the 1-hop neighborhoods to 2-hop (even k-hop) in Eq. (3). In that case, our model will exploit more sophisticated substructures such as pentagon (cyclopentadienyl) and hexagon (benzene ring), which will benefit tasks on molecular graphs but increase the complexity. Thus, we leave it to future work.

4.3. Running Time Analysis

As discussed above, compared to other powerful GNNs, one of the most important advantages of PG-GNN is efficiency. To evaluate, we compare the average running times between PG-GNN and baselines on two large-scale benchmarks, MNIST and ZINC. Table 4 also presents the average running times per epoch for various models. As shown in Table 4, PG-GNN is significantly faster than other powerful baselines, even on par with several variants of MPNNs. Thus, we can conclude that our approach outperforms other powerful GNNs in terms of time complexity. We also provide memory cost analysis in Tables 5 and 6, please refer to Appendix L.2 for more details.

5. Conclusion and Future Work

In this work, we devise an efficient permutation-sensitive aggregation mechanism via permutation groups, capturing pairwise correlations between neighboring nodes while ensuring linear sampling complexity. We throw light on the

reasons why permutation-sensitive functions can improve GNNs' expressivity. Moreover, we propose to approximate the property of permutation-invariance to significantly reduce the complexity with a minimal loss of generalization capability. In conclusion, we take an important step forward to better understand the permutation-sensitive GNNs.

However, Eq. (3) only models a small portion of n-ary dependencies while covering all 2-ary dependencies. Although these 2-ary dependencies are invariant to an arbitrary permutation, the invariance to higher-order dependencies may not be guaranteed. It would be interesting to extend the 1-hop neighborhoods to 2-hop (even k-hop) in Eq. (3), thereby completely modeling higher-order dependencies and exploiting more sophisticated substructures, which is left for future work.

Acknowledgements

References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pp. 2112–2118, 2021.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

Al Hasan, M. and Dave, V. S. Triangle counting in large networks: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(2):e1226, 2018.

Artin, M. *Algebra*. Pearson Prentice Hall, 2011.

Azizian, W. and Lelarge, M. Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*, 2021.

Balcilar, M., Héroux, P., Gaüzère, B., Vasseur, P., Adam, S., and Honeine, P. Breaking the limits of message passing graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 599–608, 2021.

Barceló, P., Geerts, F., Reutter, J., and Ryschkov, M. Graph neural networks with local graph parameters. In *Advances in Neural Information Processing Systems*, pp. 25280–25293, 2021.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022.

Birkhoff, G. and Mac Lane, S. *A survey of modern algebra*. CRC Press, 2017.

Bodnar, C., Frasca, F., Otter, N., Wang, Y. G., Liò, P., Montúfar, G. F., and Bronstein, M. Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, pp. 2625–2640, 2021a.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montúfar, G. F., Liò, P., and Bronstein, M. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 1026–1037, 2021b.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Cai, J.-Y., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

Chen, X., Li, Y., Wang, P., and Lui, J. C. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment*, 10(3):253–264, 2016.

Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems*, pp. 15894–15902, 2019.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems*, pp. 10383–10395, 2020.

Chung, K.-M., Lam, H., Liu, Z., and Mitzenmacher, M. Chernoff-Hoeffding bounds for Markov chains: Generalized and simplified. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science*, pp. 124–135, 2012.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, pp. 13260–13271, 2020.

Dasoulas, G., Santos, L. D., Scaman, K., and Virmaux, A. Coloring graph neural networks for node disambiguation. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pp. 2126–2132, 2020.

de Haan, P., Cohen, T., and Welling, M. Natural graph networks. In *Advances in Neural Information Processing Systems*, pp. 3636–3646, 2020.

Deo, N. *Graph theory with applications to engineering and computer science*. Courier Dover Publications, 2017.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Elman, J. L. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Garg, V., Jegelka, S., and Jaakkola, T. Generalization and representational limits of graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3419–3430, 2020.

Geerts, F. On the expressive power of linear algebra on graphs. *Theory of Computing Systems*, 65(1):179–239, 2021.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Grohe, M. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017.

Hagberg, A., Swart, P., and S Chult, D. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pp. 11–15, 2008.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Harary, F. and Manvel, B. On the number of cycles in a graph. *Matematický časopis*, 21(1):55–63, 1971.

Hardiman, S. J. and Katzir, L. Estimating clustering coefficients and size of social networks via random walk. In *Proceedings of the 22nd International Conference on World Wide Web*, pp. 539–550, 2013.

Haykin, S. *Neural networks and learning machines, 3/E*. Pearson Education India, 2010.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. ZINC: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Kolouri, S., Naderializadeh, N., Rohde, G. K., and Hoffmann, H. Wasserstein embedding for graph learning. In *International Conference on Learning Representations*, 2021.

Kondor, R., Son, H. T., Pan, H., Anderson, B., and Trivedi, S. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Advances in Neural Information Processing Systems*, pp. 4465–4478, 2020.

Loukas, A. What graph neural networks cannot learn: Depth vs width. In *International Conference on Learning Representations*, 2020.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pp. 2156–2167, 2019a.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Mitzenmacher, M. and Upfal, E. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 4602–4609, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020. URL www.graphlearning.io.

Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4663–4673, 2019a.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019b.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8026–8037, 2019.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, pp. 4081–4090, 2019.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining*, pp. 333–341, 2021.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

Tahmasebi, B., Lim, D., and Jegelka, S. Counting substructures with higher-order graph neural networks: Possibility and impossibility results. *arXiv preprint arXiv:2012.03174*, 2020.

Thiede, E. H., Zhou, W., and Kondor, R. Autobahn: Automorphism-based graph neural nets. In *Advances in Neural Information Processing Systems*, pp. 29922–29934, 2021.

Vignac, C., Loukas, A., and Frossard, P. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems*, pp. 14143–14155, 2020.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Weisfeiler, B. and Leman, A. A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9): 12–16, 1968.

West, D. B. *Introduction to graph theory*. Prentice Hall, 2001.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

You, J., Gomes-Selman, J., Ying, R., and Leskovec, J. Identity-aware graph neural networks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pp. 10737–10745, 2021.

Zhang, M. and Li, P. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 15734–15747, 2021.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 4438–4445, 2018.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

A. Background on Graph Theory

Given a graph $G = (\mathcal{V}, \mathcal{E})$, a *walk* in G is a finite sequence of alternating vertices and edges such as $v_0, e_1, v_1, e_2, \ldots, e_m, v_m$, where each edge $e_i = (v_{i-1}, v_i)$. A walk may have repeated edges. A *trail* is a walk in which all the edges are distinct. A *path* is a trail in which all vertices (hence all edges) are distinct (except, possibly, $v_0 = v_m$). A trail or path is *closed* if $v_0 = v_m$, and a closed path containing at least one edge is a *cycle* (West, 2001).

A *Hamiltonian path* is a path in a graph that passes through each vertex exactly once. A *Hamiltonian cycle* is a cycle in a graph that passes through each vertex exactly once. A *Hamiltonian graph* is a graph that contains a Hamiltonian cycle.

Let $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}', \mathcal{E}')$ be graphs. If $G' \subseteq G$ and G' contains all the edges $(v_i, v_j) \in \mathcal{E}$ with $v_i, v_j \in \mathcal{V}'$, then G' is an *induced subgraph* of G, and we say that $\mathcal{V}'$ *induces* G' in G.

An *empty graph* is a graph whose edge-set is empty. A *regular graph* is a graph in which each vertex has the same degree. If each vertex has degree r, the graph is *r-regular*. A *strongly regular graph* in the family $\mathrm{SRG}(v, r, \lambda, \mu)$ is an r-regular graph with v vertices, where every two adjacent vertices have λ common neighbors, and every two non-adjacent vertices have μ common neighbors.

A *complete graph* is a simple undirected graph in which every pair of distinct vertices is adjacent. We denote the complete graph on n vertices by K_n. A *tournament* is a directed graph in which each edge of a complete graph is given an orientation. We denote the tournament on n vertices by $\vec{K}_n$. A *clique* of a graph G is a complete induced subgraph of G. A clique of size k is called a *k-clique*.

The *local clustering coefficient* of a vertex quantifies how close its neighbors are to being a clique (complete graph). The local clustering coefficient c_v of a vertex v is given by the proportion of links between the n vertices within its neighborhood $\mathcal{N}(v)$ divided by the number of links that could possibly exist between them, defined as $c_v = \dfrac{2\,|\{e_{ij} : i, j \in \mathcal{N}(v), e_{ij} \in \mathcal{E}\}|}{n(n-1)}$.

This measure is 1 if every neighbor connected to v is also connected to every other vertex within the neighborhood.

Let $G = (\mathcal{V}, \mathcal{E})$ and $G' = (\mathcal{V}', \mathcal{E}')$ be graphs. An *isomorphism* $\vartheta : \mathcal{V} \to \mathcal{V}'$ between G and G' is a bijective map that maps pairs of connected vertices to pairs of connected vertices, and likewise for pairs of non-connected vertices, i.e., $(\vartheta(u), \vartheta(v)) \in \mathcal{E}'$ iff $(u, v) \in \mathcal{E}$ for all u and v in $\mathcal{V}$.

B. Background on Group Theory

Since we deal with finite sets in this paper, all the following definitions are about finite groups.

For an arbitrary element x in a group $\mathfrak{G}$, the *order* of x is the smallest positive integer n such that $x^n = e$, where e is the *identity element*. $H = \{e, x, x^2, \ldots, x^{n-1}\}$ is the *cyclic subgroup* generated by x and is often denoted by $H = \langle x \rangle$. A *cyclic group* is a group that is equal to one of its cyclic subgroups: $\mathfrak{G} = \langle g \rangle$ for some element g, and the element g is called a *generator*. The cyclic group with n elements is denoted by $\mathbb{Z}_n$ (Artin, 2011; Birkhoff & Mac Lane, 2017).

A *permutation* of a finite set S is a bijective map from S to itself. In Cauchy's two-line notation, it denotes such a permutation by listing the "natural" order for all the n elements of S in the first row, and for each one, its image below it in the second row: $\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}$. A *cycle* of length r (or *r-cycle*) is a permutation σ for which there exists an element i_1 in $\{1, 2, \ldots, n\}$ such that $\sigma(i_1) = i_2, \sigma(i_2) = i_3, \cdots, \sigma(i_{r-1}) = i_r, \sigma(i_r) = i_1$ are the only elements moved by σ. In cycle notation, it denotes such a cycle (or r-cycle) by $(i_1\ i_2\ \cdots\ i_r)$.

A *permutation group* is a group whose elements are permutations of a given set S, with the group operation "$\circ$" being the composition of permutations. The permutation group on the set S is denoted by $Perm(S)$. A *symmetric group* is a group whose elements are **all** permutations of a given set S. The symmetric group on the set $S = [n] = \{1, 2, \ldots, n\}$ is denoted by $\mathcal{S}_n$ (Artin, 2011). Every permutation group is a subgroup of a symmetric group.

A *group action* α of a group $\mathfrak{G}$ on a set S is a map $\alpha : \mathfrak{G} \times S \to S$, denoted by $(g, s) \mapsto gs$ (with $\alpha(g, s)$ often shortened to gs or $g \cdot s$) that satisfies the following two axioms:

    a) identity: $e \cdot s = s$, for all $s \in S$, where e is the identity element of $\mathfrak{G}$.

    b) associative law: $(g_1 \circ g_2) \cdot s = g_1 \cdot (g_2 \cdot s)$, for all $g_1, g_2 \in \mathfrak{G}$ and $s \in S$, where $\circ$ denotes the operation or composition in $\mathfrak{G}$.

Let $\mathfrak{G}$ and $\mathfrak{G}'$ be groups. A *homomorphism* $\varphi : \mathfrak{G} \to \mathfrak{G}'$ is a map from $\mathfrak{G}$ to $\mathfrak{G}'$ such that $\varphi(ab) = \varphi(a)\varphi(b)$ for all a and b in $\mathfrak{G}$. An *isomorphism* $\varphi : \mathfrak{G} \to \mathfrak{G}'$ from $\mathfrak{G}$ to $\mathfrak{G}'$ is a bijective group homomorphism - a bijective map such that $\varphi(ab) = \varphi(a)\varphi(b)$ for all a and b in $\mathfrak{G}$ (Artin, 2011). We use the symbol $\cong$ to denote two groups $\mathfrak{G}$ and $\mathfrak{G}'$ are isomorphic, i.e., $\mathfrak{G} \cong \mathfrak{G}'$.

C. Definition of k-WL Test

There are different definitions of the k-dimensional Weisfeiler-Lehman (k-WL) test for $k \geq 2$, while in this work, we follow the definition in Chen et al. (2020). Note that the k-WL test here is equivalent to the k-WL tests in (Grohe, 2017; Morris et al., 2019; Maron et al., 2019a; Azizian & Lelarge, 2021), and the $(k-1)$-WL test in (Cai et al., 1992) (Grohe (2017) calls this version as k-WL'). $(k+1)$-WL test has been proven to be strictly more powerful than k-WL test (Cai et al., 1992).

The k-WL algorithm is a generalization of the 1-WL, it colors tuples from $\mathcal{V}^k$ instead of nodes. For any k-tuple $s = (i_1, \ldots, i_k) \in \mathcal{V}^k$ and each $j \in [k] = \{1, \ldots, k\}$, define the j-th neighborhood

$$\mathcal{N}_j(s) = \{(i_1, \ldots, i_{j-1}, u, i_{j+1}, \ldots, i_k) \mid u \in \mathcal{V}\}$$

That is, the j-th neighborhood $\mathcal{N}_j(s)$ of the k-tuple s is obtained by replacing the j-th component of s with every node from $\mathcal{V}$.

Given a pair of graphs G and G', we use the k-WL algorithm to test them for isomorphism. Suppose that the two graphs have the same number of vertices since otherwise, they can be told apart easily. Without loss of generality, we assume that they share the same set of vertex indices, $\mathcal{V}$ (but may differ in $\mathcal{E}$). The k-WL test follows the following coloring procedure.

1) For each of the graphs, at iteration 0, the test assigns an initial color in the color space Γ to each k-tuple according to its *atomic type*, i.e., two k-tuples s and s' in $\mathcal{V}^k$ get the same color if the subgraphs induced from nodes of s and s' are isomorphic.

2) In each iteration $t > 0$, the test computes a k-tuple coloring $c_k^{(t)} : \mathcal{V}^k \to \Gamma$. More specifically, let $c_k^{(t)}(s)$ denote the color of s in G assigned at the t-th iteration, and let $c'^{(t)}_k(s')$ denote the color assigned for s' in G'. Define

$$C_j^{(t)}(s) = \text{HASH}_1^{(t)}\left(\left\{c_k^{(t-1)}(w) \;\middle|\; w \in \mathcal{N}_j(s)\right\}\right)$$
$$C'^{(t)}_j(s') = \text{HASH}_1^{(t)}\left(\left\{c'^{(t-1)}_k(w') \;\middle|\; w' \in \mathcal{N}_j(s')\right\}\right)$$

where $\text{HASH}_1^{(t)}$ is a hash function that maps injectively from the space of multisets of colors to some intermediate space. Then let

$$c_k^{(t)}(s) = \text{HASH}_2^{(t)}\left(\left(c_k^{(t-1)}(s), \left(C_1^{(t)}(s), \ldots, C_k^{(t)}(s)\right)\right)\right)$$
$$c'^{(t)}_k(s') = \text{HASH}_2^{(t)}\left(\left(c'^{(t-1)}_k(s'), \left(C'^{(t)}_1(s'), \ldots, C'^{(t)}_k(s')\right)\right)\right)$$

where $\text{HASH}_2^{(t)}$ maps injectively from its input space to the color space Γ, $c_k^{(t)}(s)$ and $c'^{(t)}_k(s)$ are updated iteratively in this way.

3) The test will terminate and return the result that the two graphs are not isomorphic if the following two multisets differ at some iteration t:

$$\left\{c_k^{(t)}(s) \;\middle|\; s \in \mathcal{V}^k\right\} \neq \left\{c'^{(t)}_k(s') \;\middle|\; s' \in \mathcal{V}^k\right\}$$

For the detailed difference between k-WL test here and $(k-1)$-WL test in Cai et al. (1992) (k-WL' in Grohe (2017)), see Remark 3.5.9 in Grohe (2017).
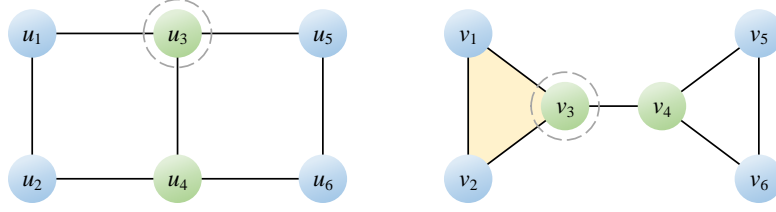
Figure 3. A pair of non-isomorphic graphs that cannot be distinguished by permutation-invariant aggregation functions, but can be easily distinguished by permutation-sensitive aggregation functions.

D. Distinguishing Non-Isomorphic Graph Pairs: Permutation-Sensitive vs. Permutation-Invariant Aggregation Functions

Let f be an arbitrary aggregation function. For a node v, let x_v (b for blue, g for green) denote the initial node feature, h_v denote the feature transformed by f. In the initial stage, we have:

$$x_{u_1} = x_{u_2} = x_{u_5} = x_{u_6} = b, \ x_{u_3} = x_{u_4} = g$$
$$x_{v_1} = x_{v_2} = x_{v_5} = x_{v_6} = b, \ x_{v_3} = x_{v_4} = g$$

Figure 3 illustrates a pair of non-isomorphic graphs that 2-WL test and most permutation-invariant aggregation functions fail to distinguish. Suppose f is permutation-invariant, we take the sum aggregator $\text{SUM}(\cdot)$ as an example to illustrate this process. After the first round of iteration, the transformed feature of each node is:

$$h_{u_1} = h_{u_2} = h_{u_5} = h_{u_6} = b + g, \ h_{u_3} = h_{u_4} = 2b + g$$
$$h_{v_1} = h_{v_2} = h_{v_5} = h_{v_6} = b + g, \ h_{v_3} = h_{v_4} = 2b + g$$

We can find that the distributions of node features of these two graphs are the same. Similarly, after each round of iteration, these two graphs always produce the same distributions of node features. Hence we can conclude that the 2-WL test and the permutation-invariant function $\text{SUM}(\cdot)$ fail to distinguish these two graphs.

In contrast, suppose f is permutation-sensitive, we take a generic permutation-sensitive aggregator $h^{(t)} = k \cdot h^{(t-1)} + x^{(t)}$ as an example to illustrate its process. Here $x^{(t)}$ is the t-th input node feature, $h^{(t)}$ is the corresponding transformed feature with $h^{(0)} = 0$, and the learnable parameter $k > 1$ measures the pairwise correlation between $x^{(t-1)}$ and $x^{(t)}$. For the left graph G_1, we focus on node u_3. Let the input ordering of neighboring nodes be u_1, u_4, u_5, i.e., $x_{u_3}^{(1)} \to x_{u_3}^{(2)} \to x_{u_3}^{(3)} = b \to g \to b$, then f only encodes the pairwise correlation between b and g. Thus, we have

$$h_{u_3}^{(1)} = k \cdot 0 + b = b$$
$$h_{u_3}^{(2)} = k \cdot b + g = kb + g$$
$$h_{u_3}^{(3)} = k \cdot (kb + g) + b = (k^2 + 1)b + kg$$

For the right graph G_2, we focus on node v_3. Let the input ordering of neighboring nodes be v_1, v_2, v_4, i.e., $x_{v_3}^{(1)} \to x_{v_3}^{(2)} \to x_{v_3}^{(3)} = b \to b \to g$, then f also encodes the pairwise correlation between b and b. Thus, we have

$$h_{v_3}^{(1)} = k \cdot 0 + b = b$$
$$h_{v_3}^{(2)} = k \cdot b + b = kb + b$$
$$h_{v_3}^{(3)} = k \cdot (kb + b) + g = (k^2 + k)b + g$$

After the first round of iteration, the node feature $h_{u_3}^{(3)}$ of u_3 differs from the $h_{v_3}^{(3)}$ of v_3. Hence we can conclude that the permutation-sensitive aggregation function f can distinguish these two graphs. Moreover, the weight ratio of b and g in $h_{u_3}^{(3)}$ is $(k^2 + 1) : k$, which is smaller than that in $h_{v_3}^{(3)}$, i.e., $(k^2 + k) : 1$. This fact indicates that, in G_1, f focuses more on encoding the pairwise correlation between b and g. In contrast, in G_2, f focuses more on encoding the pairwise correlation between b and b, thereby exploiting the triangular substructure such as $\triangle v_1 v_3 v_2$. It is worth noting that when $k = 1$, the function f is $h^{(t)} = h^{(t-1)} + x^{(t)}$ and degenerates to the permutation-invariant function $\text{SUM}(\cdot)$, resulting in $h_{u_3}^{(3)} = h_{v_3}^{(3)} = 2b + g$.
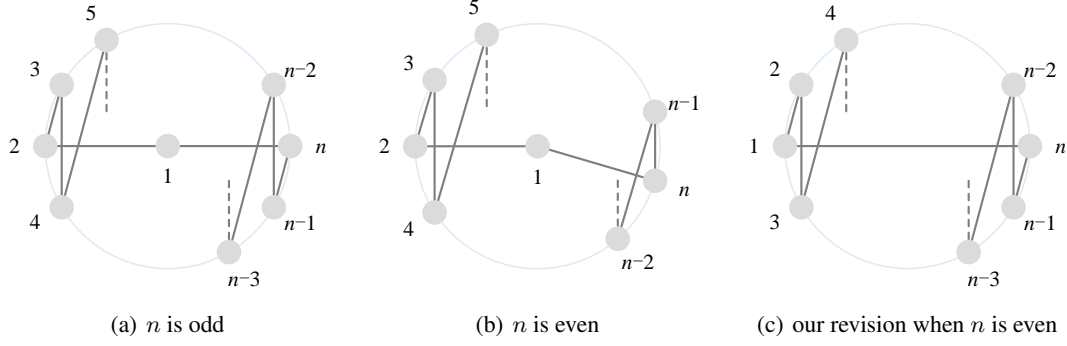
(a) n is odd   (b) n is even   (c) our revision when n is even

Figure 4. The initial arrangements (following the gray solid lines).

E. Proof of Theorem 1

Theorem 1. *Let $n(n \geq 4)$ denote the number of 1-hop neighboring nodes around the central node v. There are $\lfloor (n-1)/2 \rfloor$ kinds of arrangements in total, satisfying that their corresponding 2-ary dependencies are disjoint. Meanwhile, after at least $\lfloor n/2 \rfloor$ arrangements (including the initial one), all 2-ary dependencies have been covered at least once.*

Proof. Construct a simple undirected graph $G' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}'$ denotes the n neighboring nodes (abbreviated as nodes in the following) around the central node v, and $\mathcal{E}'$ represents an edge set in which each edge indicates the corresponding 2-ary dependency has been covered in some arrangements. Thus, each arrangement corresponds to a Hamiltonian cycle in graph G'. For any two arrangements, detecting whether their corresponding 2-ary dependencies are disjoint can be analogous to finding two edge-disjoint Hamiltonian cycles. Since every pair of nodes can form a 2-ary dependency, the first problem can be translated into finding the maximum number of edge-disjoint Hamiltonian cycles in a complete graph K_n, and the second problem can be translated into finding the minimum number of Hamiltonian cycles to cover a complete graph K_n.

Since a K_n has $\frac{n(n-1)}{2}$ edges and each Hamiltonian cycle has n edges, there are at most $\lfloor \frac{n(n-1)}{2}/n \rfloor = \lfloor \frac{n-1}{2} \rfloor$ edge-disjoint Hamiltonian cycles in a K_n. In addition, we can specifically construct $\lfloor \frac{n-1}{2} \rfloor$ edge-disjoint Hamiltonian cycles as follows. If n is odd, keep the nodes fixed on a circle with node 1 at the center, rotate the node numbers on the circle clockwise by $\frac{360°}{n-1}, 2 \times \frac{360°}{n-1}, \ldots, \frac{n-3}{2} \times \frac{360°}{n-1}$, while the graph structure always remains unchanged as the initial arrangement shown in Figure 4(a). Each rotation can be formulated as the following permutation σ':

$$
\sigma' = \begin{cases} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & n-1 & n \\ 1 & 4 & 2 & 6 & 3 & \cdots & n & n-2 \end{pmatrix} = (2\,4\,6\,\cdots\,n-1\,n\,n-2\,\cdots\,7\,5\,3), & \text{if } n \text{ is odd}, \\[2mm] \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & n-1 & n \\ 1 & 4 & 2 & 6 & 3 & \cdots & n-3 & n-1 \end{pmatrix} = (2\,4\,6\,\cdots\,n-2\,n\,n-1\,\cdots\,7\,5\,3), & \text{if } n \text{ is even}. \end{cases}
$$

Observe that each rotation generates a new Hamiltonian cycle containing completely different edges from before. Thus we have $\frac{n-3}{2} = \lfloor \frac{n-1}{2} \rfloor - 1$ new Hamiltonian cycles with all edges disjoint from the ones in Figure 4(a) and among themselves (Deo, 2017). If n is even, the node arrangement can be initialized as shown in Figure 4(b), and $\frac{n-4}{2} = \lfloor \frac{n-1}{2} \rfloor - 1$ new Hamiltonian cycles can be constructed successively in a similar way. We thus conclude that there are $\lfloor \frac{n-1}{2} \rfloor$ kinds of arrangements in total, satisfying that their corresponding 2-ary dependencies are disjoint.

Furthermore, if n is odd, K_n has $\frac{n(n-1)}{2}$ edges *divisible* by the length n of each Hamiltonian cycle. Therefore, we can exactly cover all edges by the above $\lfloor \frac{n-1}{2} \rfloor = \frac{n-1}{2} = \lfloor \frac{n}{2} \rfloor$ kinds of arrangements. On the contrary, if n is even, K_n has $\frac{n(n-1)}{2}$ edges *indivisible* by the length n of each Hamiltonian cycle, remaining $\frac{n}{2}$ edges uncovered by the above $\lfloor \frac{n-1}{2} \rfloor = \frac{n-2}{2}$ kinds of arrangements. Thus we continue to perform the permutation σ' once, i.e., $\lfloor \frac{n-1}{2} \rfloor + 1 = \frac{n}{2} = \lfloor \frac{n}{2} \rfloor$ kinds of arrangements in total, to cover all edges but result in $\frac{n}{2}$ edges duplicated twice.

As discussed in the main body, these $\lfloor \frac{n}{2} \rfloor$ arrangements and the corresponding $\lfloor \frac{n}{2} \rfloor$ Hamiltonian cycles are modeled by the permutation-sensitive function in a directed manner. In addition, we also expect to reverse these $\lfloor \frac{n}{2} \rfloor$ directed Hamiltonian cycles by performing the permutation σ' successively, thereby transforming them into an undirected manner. However, σ'

cannot satisfy this requirement if n is even. Thus, we propose to revise the permutation σ' into the following one:

$$\sigma = \begin{cases} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & n-1 & n \\ 1 & 4 & 2 & 6 & 3 & \cdots & n & n-2 \end{pmatrix} = (2\ 4\ 6\ \cdots\ n-1\ n\ n-2\ \cdots\ 7\ 5\ 3)\,, \text{if } n \text{ is odd,} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & n-1 & n \\ 3 & 1 & 5 & 2 & \cdots & n & n-2 \end{pmatrix} = (1\ 3\ 5\ \cdots\ n-1\ n\ n-2\ \cdots\ 6\ 4\ 2)\,, \text{if } n \text{ is even.} \end{cases}$$

where σ is the same as σ' when n is odd, but a little different when n is even. If n is even, σ is an n-cycle, but σ' is an $(n-1)$-cycle. The corresponding initial node arrangement after revision is shown in Figure 4(c). After adding a virtual node 0 at the center in Figure 4(c), σ becomes the same as σ' with $n+1$ in Figure 4(a), which can cover all edges with $\lfloor \frac{(n+1)-1}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$ kinds of arrangements. Moreover, after performing σ for n times in succession, it can cover a complete graph bi-directionally but σ' fails.

In conclusion, after performing σ or σ' for $\lfloor \frac{n}{2} \rfloor - 1$ times in succession (excluding the initial one), all 2-ary dependencies have been covered at least once. □

F. Proof of Lemma 2

Theorem F.1. *The order of any permutation is the least common multiple of the lengths of its disjoint cycles (Birkhoff & Mac Lane, 2017).*

Proposition F.2. *The order of a cyclic group is equal to the order of its generator (Artin, 2011).*

Using Theorem F.1 and Proposition F.2, we prove Lemma 2 as follows.

Lemma 2. *For the permutation σ of n indices, $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-2}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_{n-1}$ if n is odd. And $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-1}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_n$ if n is even.*

Proof. If n is odd, we find the order of permutation σ first. Since

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & \cdots & n-1 & n \\ 1 & 4 & 2 & 6 & 3 & \cdots & n & n-2 \end{pmatrix} = (1)\,(2\ 4\ 6\ \cdots\ n-1\ n\ n-2\ \cdots\ 7\ 5\ 3)$$

Let $\pi_1 = (1)$, $\pi_2 = (2\ 4\ 6\ \cdots\ n-1\ n\ n-2\ \cdots\ 7\ 5\ 3)$, then the permutation σ can be represented as the product of these two disjoint cycles, i.e., $\sigma = \pi_1 \pi_2$. Here π_1 is a 1-cycle of length 1, π_2 is an $(n-1)$-cycle of length $n-1$. Using Theorem F.1, the order of permutation σ is the least common multiple of 1 and $n-1$: $\text{lcm}(1, n-1) = n-1$, which indicates that $\sigma^{n-1} = e$. Therefore, $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-2}\}$ is a permutation group generated by σ, i.e., $\mathfrak{G} = \langle \sigma \rangle$. According to the definition of the cyclic group (see Appendix B), $\mathfrak{G}$ is isomorphic to a cyclic group. By Proposition F.2, the order of group $\mathfrak{G} = \langle \sigma \rangle$ is equal to the order of its generator σ, i.e., $n-1$. Thus, $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-2}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_{n-1}$.

Similarly, we can prove that $\mathfrak{G} = \{e, \sigma, \sigma^2, \ldots, \sigma^{n-1}\}$ is a permutation group isomorphic to the cyclic group $\mathbb{Z}_n$ if n is even. □

Theorem F.3 (Cayley's Theorem). *Every finite group is isomorphic to a permutation group (Artin, 2011).*

The conclusion of Lemma 2 also obeys the most fundamental Cayley's Theorem in group theory.

G. Proof of Corollary 3 and the Diagram of Group Action

Corollary 3. *The map $\alpha : \mathfrak{G} \times S \to S$ denoted by $(g, s) \mapsto gs$ is a group action of $\mathfrak{G}$ on S.*

Proof. Let e be the identity element of $\mathfrak{G}$ and id_σ be the identity permutation. And let $\circ$ denote the composition in $\mathfrak{G}$. For all $\sigma^i, \sigma^j \in \mathfrak{G}$ and $s \in S$, we have

$$\alpha(e, s) = e \cdot s = id_\sigma \cdot s = s$$
$$\alpha(\sigma^i \sigma^j, s) = (\sigma^i \circ \sigma^j) \cdot s = \sigma^i \cdot (\sigma^j \cdot s) = \alpha(\sigma^i, \alpha(\sigma^j, s))$$

Thus, the map α defines a group action of the permutation group $\mathfrak{G}$ on the set S. □

(a) $\mathfrak{G}_1 = \{e, \sigma, \sigma^2, \ldots, \sigma^5\} \cong \mathbb{Z}_6$

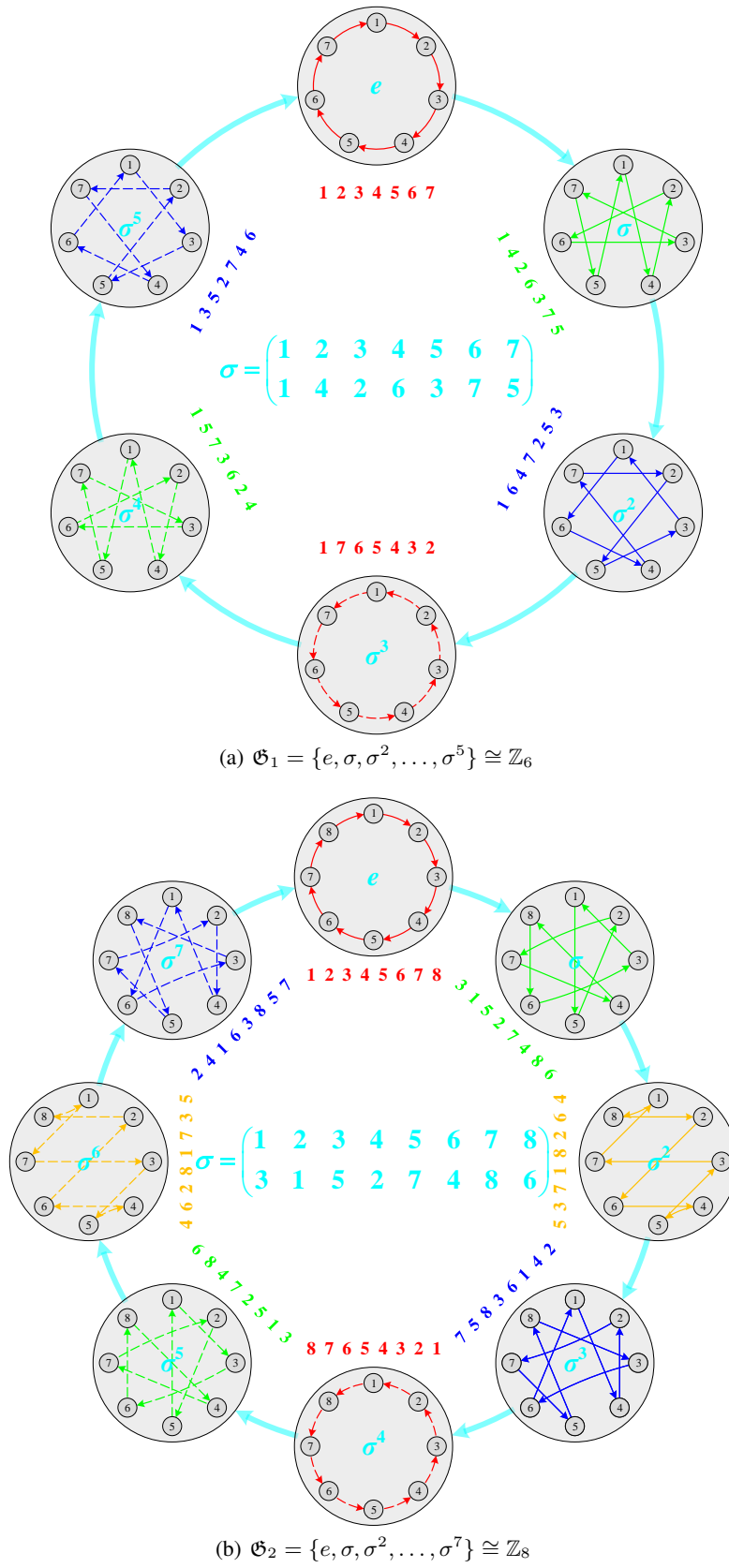(b) $\mathfrak{G}_2 = \{e, \sigma, \sigma^2, \ldots, \sigma^7\} \cong \mathbb{Z}_8$

Figure 5. The group structure of the permutation group $\mathfrak{G}$ and the results of its actions on the set S.

To better understand Lemma 2 and Corollary 3, we provide diagrams to illustrate the group actions of the permutation groups $\mathfrak{G}_1 = \{e, \sigma, \sigma^2, \ldots, \sigma^5\}$ and $\mathfrak{G}_2 = \{e, \sigma, \sigma^2, \ldots, \sigma^7\}$ when $n = 7$ and $n = 8$, respectively. As shown in Figures 5(a) and 5(b), the overall frameworks with big light-gray circles and cyan arrows represent the Cayley diagrams of the permutation groups $\mathfrak{G}_1 \cong \mathbb{Z}_6$ and $\mathfrak{G}_2 \cong \mathbb{Z}_8$ constructed by Lemma 2, respectively. The center of each subfigure presents the corresponding generator σ. Each big light-gray circle represents an element g (i.e., a permutation) of group $\mathfrak{G}$, marked at the center of the circle. And each cyan arrow $g_i \rightarrow g_j$ indicates the relationship $g_j = g_i \circ \sigma$ exists between two group elements $g_i, g_j \in \mathfrak{G}$. After g acts on the elements $1, \ldots, n$ of the set S, the corresponding images are presented as the colored numbers next to the big light-gray circle. Finally, the 2-ary dependencies (colored arrows) between neighboring nodes (small dark-gray circles) are modeled according to the action results of g, shown in each big light-gray circle.

H. Proofs About Incidence Triangles

H.1. Proof of Eq. (6)

$$\boldsymbol{\tau} = \frac{1}{2}\boldsymbol{A}^2 \odot \boldsymbol{A} \cdot \boldsymbol{1}_N, \quad \boldsymbol{1}_N^\top \boldsymbol{\tau} = \frac{1}{2}\mathrm{tr}(\boldsymbol{A}^3)$$

Proof. Let $\boldsymbol{A} = (a_{ij})_{N \times N}$, $\boldsymbol{B} = \boldsymbol{A}^2 = (b_{ij})_{N \times N}$, where a_{ij} and b_{ij} denote the (i, j) element of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Since a_{ij} equals 1 iff nodes v_i and v_j are adjacent in G, b_{ij} equals the number of walks of length 2 from nodes v_i to v_j in G. In addition, a walk of length 2 from v_i to v_j and an edge from v_j to v_i form a triangle containing both v_i and v_j. Therefore, the (i, j) element of $\boldsymbol{A}^2 \odot \boldsymbol{A}$ equals $b_{ij}a_{ij} = b_{ij}a_{ji}$, which indicates how many triangles contain both v_i and v_j. We can use a sum vector $\boldsymbol{1}_N = (1, 1, \cdots, 1)^\top \in \mathbb{R}^N$ to sum up each row of $\boldsymbol{A}^2 \odot \boldsymbol{A}$ and get a result vector, whose i-th element gives **twice** the number of incidence triangles of node v_i. Here the "twice" comes from the fact that each incidence triangle $\triangle v_j v_i v_k$ over node v_i has two walks of length 2 starting from node v_i, that is, $v_i \rightarrow v_j \rightarrow v_k$ and $v_i \rightarrow v_k \rightarrow v_j$. Hence after dividing each element of the result vector by 2, we finally obtain $\boldsymbol{\tau} = \frac{1}{2}\boldsymbol{A}^2 \odot \boldsymbol{A} \cdot \boldsymbol{1}_N$.

For the second equation, we have

$$\boldsymbol{1}_N^\top \boldsymbol{\tau} = \frac{1}{2}\boldsymbol{1}_N^\top \cdot (\boldsymbol{A}^2 \odot \boldsymbol{A}) \cdot \boldsymbol{1}_N = \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N b_{ij}a_{ij} = \frac{1}{2}\sum_{i=1}^N (\boldsymbol{A}^2 \cdot \boldsymbol{A}^\top)_{ii} = \frac{1}{2}\mathrm{tr}(\boldsymbol{A}^2 \cdot \boldsymbol{A}^\top) = \frac{1}{2}\mathrm{tr}(\boldsymbol{A}^3) \qquad \square$$

Remark. The i-th diagonal entry of $\boldsymbol{A}^3$ is equal to twice the number of triangles in which the i-th node is contained (Mitzenmacher & Upfal, 2017). In addition, each triangle has three vertices. Hence we can divide the sum of the diagonal entries by 6 to obtain the total number of triangles in graph G, i.e., $\frac{1}{6}\mathrm{tr}(\boldsymbol{A}^3)$ (Harary & Manvel, 1971).

For directed graphs, we also have similar results:

$$\vec{\boldsymbol{\tau}} = \boldsymbol{A}^2 \odot \boldsymbol{A}^\top \cdot \boldsymbol{1}_N, \quad \boldsymbol{1}_N^\top \vec{\boldsymbol{\tau}} = \mathrm{tr}(\boldsymbol{A}^3)$$

where $\vec{\boldsymbol{\tau}} \in \mathbb{R}^N$ and its i-th element $\vec{\tau}_i$ represents the number of *directed* incidence triangles over node i.

H.2. Proof of Theorem 4

Theorem H.1 (Chernoff-Hoeffding Bound for Discrete Time Markov Chain (Chung et al., 2012)). *Let $\mathcal{M}$ be an ergodic Markov chain with state space $[n] = \{1, 2, \ldots, n\}$ and stationary distribution $\boldsymbol{\pi}$. Let $\mathfrak{t} = \mathfrak{t}(\epsilon)$ be its ϵ-mixing time for $\epsilon \leq 1/8$. Let $(X_1, X_2, \ldots, X_r)$ denote an r-step random walk on $\mathcal{M}$ starting from an initial distribution φ on $[n]$, i.e., $X_1 \leftarrow \varphi$. Define $\|\varphi\|_{\boldsymbol{\pi}} = \sum_{i=1}^n \frac{\varphi_i^2}{\pi_i}$. For every step $k \in [r]$, let $f^{(k)} : [n] \rightarrow [0, 1]$ be a weight function such that the expectation $\mathbb{E}_{X_k \leftarrow \boldsymbol{\pi}}[f^{(k)}(X_k)] = \mu$ for all k. Define the total weight of the walk $(X_1, X_2, \ldots, X_r)$ by $Z \triangleq \sum_{k=1}^r f(X_k)$. There exists some constant c (which is independent of μ, δ and ϵ) such that for $0 < \delta < 1$*

$$\Pr\left(|Z - \mu r| > \epsilon \mu r\right) \leq c\|\varphi\|_{\boldsymbol{\pi}} \exp\left(-\frac{\epsilon^2 \mu r}{72\mathfrak{t}}\right)$$

or equivalently

$$\Pr\left(\left|\frac{Z}{r} - \mu\right| > \epsilon \mu\right) \leq c\|\varphi\|_{\boldsymbol{\pi}} \exp\left(-\frac{\epsilon^2 \mu r}{72\mathfrak{t}}\right).$$

Theorem H.2. *Any nonlinear dynamic system may be approximated by a recurrent neural network to any desired degree of accuracy and with no restrictions imposed on the compactness of the state space, provided that the network is equipped with an adequate number of hidden neurons (Haykin, 2010).*

Using Theorem H.1 and Theorem H.2, we prove Theorem 4 as follows.

Theorem 4. *Let $x_v, \forall v \in \mathcal{V}$ denote the feature inputs on graph $G = (\mathcal{V}, \mathcal{E})$, and $\mathbb{M}$ be a general GNN model with RNN aggregators. Suppose that x_v is initialized as the degree d_v of node v, and each node is distinguishable. For any $0 < \epsilon \leq 1/8$ and $0 < \delta < 1$, there exists a parameter setting Θ for $\mathbb{M}$ so that after $\mathcal{O}\left(\frac{d_v(2d_v + \tau_v)\mathsf{t}}{d_v + \tau_v}\right)$ samples,*

$$\Pr\left(\left|\frac{z_v}{\tau_v} - 1\right| \leq \epsilon\right) \geq 1 - \delta, \forall v \in \mathcal{V},$$

where $z_v \in \mathbb{R}$ is the final output value generated by $\mathbb{M}$ and τ_v is the number of incidence triangles.

Proof. Without loss of generality, we discuss how to estimate the number of incidence triangles τ_0 for an arbitrary node v_0 based on its n neighbors $v_1', v_2', \ldots, v_n'$. Let $v_0' = v_0$, and let $G' = (\mathcal{V}', \mathcal{E}')$ denote the subgraph induced by $\mathcal{V}' = \{v_0', v_1', v_2', \ldots, v_n'\}$, with an adjacency matrix $\boldsymbol{A}' \in \mathbb{R}^{(n+1)\times(n+1)}$. We add a symbol "*'*" to all notations of the induced subgraph G' to distinguish them from those of graph G. For each node $v_i' \in \mathcal{V}'$, d_i' denotes the degree of v_i' in graph G', τ_i' denotes the number of incidence triangles of v_i' in graph G'. In particular, $d_0' = d_0 = n$, $\tau_0' = \tau_0$. Our goal is to estimate τ_0 for an arbitrary node v_0 in graph G, which is equal to τ_0' in graph G'.

A *simple random walk* (SRW) with r steps on graph G', denoted by $R = (X_1, X_2, \ldots, X_r)$, is defined as follows: start from an initial node in G', then move to one of its neighboring nodes chosen uniformly at random, and repeat this process $(r-1)$ times. This random walk on graph G' can be viewed as a finite Markov chain $\mathcal{M}$ with the state space $\mathcal{V}'$, and the transition probability matrix $\boldsymbol{P}$ of this Markov chain is defined as

$$\boldsymbol{P}(i,j) = \begin{cases} \dfrac{1}{d_i'}, & \text{if } (v_i', v_j') \in \mathcal{E}', \\ 0, & \text{otherwise.} \end{cases}$$

Let $D' = \sum_{i=0}^{n} d_i' = 2|\mathcal{E}'|$ denote the sum of degrees in graph G'. After many random walk steps, the probability $\Pr(X_r = v_i')$ converges to $p_i \triangleq d_i'/D'$, and the vector $\boldsymbol{\pi} = (p_0, p_1, \ldots, p_n)$ is called the *stationary distribution* of this random walk.

The *mixing time* of a Markov chain is the number of steps it takes for a random walk to approach its stationary distribution. We adopt the definition in (Chung et al., 2012; Chen et al., 2016; Mitzenmacher & Upfal, 2017) and define the mixing time $\mathsf{t}(\epsilon)$ as follows:

$$\mathsf{t}(\epsilon) = \max_{X_i \in \mathcal{V}'} \min\left\{t : \left|\boldsymbol{\pi} - \boldsymbol{\pi}^{(i)}\boldsymbol{P}^t\right| < \epsilon\right\}$$

where $\boldsymbol{\pi}$ is the stationary distribution of the Markov chain defined above, $\boldsymbol{\pi}^{(i)}$ is the initial distribution when starting from state $X_i \in \mathcal{V}'$, $\boldsymbol{P}^t$ is the transition matrix after t steps, and $|\cdot|$ is the variation distance between two distributions.

Later on, we will exploit node samples taken from a random walk to construct an estimator z_0, then use the mixing time based Chernoff-Hoeffding bound (Chung et al., 2012) to compute the number of steps/samples needed, thereby guaranteeing that our estimator z_0 is within $(1 \pm \epsilon)$ of the true value τ_0 with the probability of at least $1 - \delta$.

Given a random walk $(X_1, X_2, \ldots, X_r)$ on graph G', we define a new variable $a_k = \boldsymbol{A}'_{X_{k-1}, X_{k+1}}$ for every $2 \leq k \leq r-1$, then we have

$$\begin{aligned} \mathbb{E}\left[a_k d'_{X_k}\right] &= \sum_{i=0}^{n} p_i \mathbb{E}\left[a_k d'_{X_k} \mid X_k = v_i'\right] \\ &= \sum_{i=0}^{n} \frac{d_i'}{D'} \frac{2\tau_i'}{d_i'^2} d_i' \\ &= \frac{2}{D'} \sum_{i=0}^{n} \tau_i' \end{aligned} \tag{7}$$

The second equality holds because there are $d_i'^2$ equal probability combinations of (X_{k-1}, v_i', X_{k+1}), out of which only $2\tau_i'$ combinations form a triangle (u', v_i', w') or its reverse (w', v_i', u'), where u' is connected to w', i.e., $a_k = \boldsymbol{A}'_{X_{k-1}, X_{k+1}} = \boldsymbol{A}'_{u', w'} = 1$.

To estimate τ_0, we introduce two variables Y_1 and Y_2, defined as follows:

$$Y_1 \triangleq \frac{1}{r-2} \sum_{k=2}^{r-1} a_k d'_{X_k}, \quad Y_2 \triangleq \frac{1}{r} \sum_{k=1}^{r} \frac{1}{d'_{X_k}}$$

Using the linearity of expectation and Eq. (7), we obtain

$$\mathbb{E}[Y_1] = \frac{1}{r-2} \sum_{k=2}^{r-1} \mathbb{E}\left[a_k d'_{X_k}\right] = \frac{2}{D'} \sum_{i=0}^{n} \tau_i' \tag{8}$$

Similarly, we have

$$\mathbb{E}[Y_2] = \frac{1}{r} \sum_{k=1}^{r} \mathbb{E}\left[\frac{1}{d'_{X_k}}\right] = \frac{1}{r} \sum_{k=1}^{r} \left(\sum_{i=0}^{n} \frac{d_i'}{D'} \frac{1}{d_i'}\right) = \frac{n+1}{D'} \tag{9}$$

Recall that G' is a subgraph induced by $\mathcal{V}' = \{v_0', v_1', v_2', \ldots, v_n'\}$, where $v_1', v_2', \ldots, v_n'$ are n neighbors of an arbitrary node $v_0' = v_0$. Therefore, the maximum degree of graph G' is $\Delta' = n$, which is equal to $d_0' = d_0$. In addition, we have $\sum_{i=0}^{n} \tau_i' = 3\tau_0' = 3\tau_0$, and $D' = 2|\mathcal{E}'| = 2(d_0' + \tau_0') = 2(d_0 + \tau_0)$. Substituting them in Eq. (8) and Eq. (9), we get

$$\mathbb{E}[Y_1] = \frac{3\tau_0}{d_0 + \tau_0} \tag{10}$$

and

$$\mathbb{E}[Y_2] = \frac{d_0 + 1}{2(d_0 + \tau_0)} \tag{11}$$

From Eq. (10) and Eq. (11) we can isolate τ_0 and get

$$\tau_0 = \frac{d_0 + 1}{6} \cdot \frac{\mathbb{E}[Y_1]}{\mathbb{E}[Y_2]} \tag{12}$$

Since d_0 is the feature input, the coefficient $\frac{d_0 + 1}{6}$ can be considered as a constant factor here. Intuitively, both Y_1 and Y_2 converge to their expected values, and thus the estimator $z_0 \triangleq \frac{d_0 + 1}{6} \cdot \frac{Y_1}{Y_2}$ converges to τ_0 as well. Next, we will find the number of steps/samples r for convergence.

Since $a_k d'_{X_k} = \boldsymbol{A}'_{X_{k-1}, X_{k+1}} d'_{X_k}$ in Y_1 only depends on a 3-nodes history, we observe a related Markov chain $\tilde{\mathcal{M}}$ that remembers the three latest visited nodes. Accordingly, $\tilde{\mathcal{M}}$ has $(n+1) \times (n+1) \times (n+1)$ states, and $(X_{k-1}, X_k, X_{k+1}) \to (X_k, X_{k+1}, X_{k+2})$ has the same transition probability as $X_{k+1} \to X_{k+2}$ in $\mathcal{M}$. Define each state $\tilde{X}_k = (X_{k-1}, X_k, X_{k+1})$ for $2 \le k \le r-1$. Let $\tilde{f}_1^{(k)}(\tilde{X}_k) = f_1^{(k)}(X_k) = \frac{a_k d'_{X_k}}{\Delta'} = \frac{a_k d'_{X_k}}{d_0}$ such that all values of $\tilde{f}_1^{(k)}(\tilde{X}_k)$ are in $[0,1]$. By Eq. (7), Eq. (8), and Eq. (10), we have $\mu_1 = \mathbb{E}_{\tilde{X}_k \leftarrow \boldsymbol{\pi}}(\tilde{f}_1^{(k)}(\tilde{X}_k)) = \frac{3\tau_0}{d_0(d_0 + \tau_0)}$. Define $Z_1 \triangleq \sum_{k=2}^{r-1} \tilde{f}_1^{(k)}(\tilde{X}_k) = \frac{r-2}{d_0} Y_1$, assume that $\varphi \approx \boldsymbol{\pi}$ thus $\|\varphi\|_{\boldsymbol{\pi}} = 1$. By Theorem H.1 and Eq. (10), we have

$$\Pr\left(|Y_1 - \mathbb{E}[Y_1]| > \frac{\epsilon}{3}\mathbb{E}[Y_1]\right) \le c_1 \exp\left(-\frac{3 \cdot \epsilon^2 \tau_0 (r-2)}{9 \cdot 72 \cdot \tilde{t} d_0 (d_0 + \tau_0)}\right) \tag{13}$$

Extracting r_{Y_1} from $\frac{\delta}{2} = c_1 \exp\left(-\frac{\epsilon^2 \tau_0 (r-2)}{216 \cdot \tilde{t} d_0 (d_0 + \tau_0)}\right)$, we obtain $r_{Y_1} = 2 - 216\frac{\ln(\delta/2c_1)}{\epsilon^2} \cdot \frac{d_0(d_0 + \tau_0)\tilde{t}}{\tau_0} = \mathcal{O}\left(\frac{d_0(d_0 + \tau_0)\tilde{t}}{\tau_0}\right)$, where c_1, ϵ and δ are all constants.

(a) Star graph, without any triangles   (b) General case, with some triangles
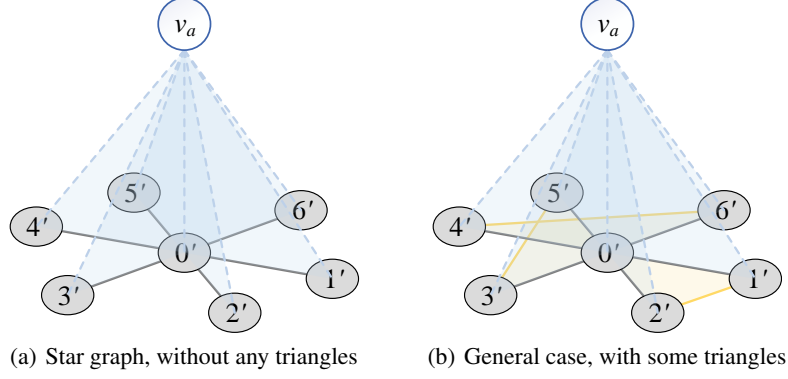
Figure 6. Add an artificial node v_a and connect it to all nodes in G'.

Let $f_2^{(k)}(X_k) = \dfrac{1}{d'_{X_k}}$, by Eq. 9 and Eq. 11 we have $\mu_2 = \mathbb{E}_{X_k \leftarrow \pi}(f_2^{(k)}(X_k)) = \dfrac{d_0+1}{2(d_0+\tau_0)}$. Define $Z_2 \triangleq \sum\limits_{k=1}^{r} f_2^{(k)}(X_k) = rY_2$, assume that $\varphi \approx \pi$ thus $\|\varphi\|_{\pi} = 1$. By Theorem H.1 and Eq. (11), we have

$$\Pr\left(|Y_2 - \mathbb{E}[Y_2]| > \frac{\epsilon}{3}\mathbb{E}[Y_2]\right) \le c_2 \exp\left(-\frac{\epsilon^2(d_0+1)r}{2 \cdot 9 \cdot 72 \cdot \mathfrak{t}(d_0+\tau_0)}\right) \tag{14}$$

Extracting r_{Y_2} from $\dfrac{\delta}{2} = c_2 \exp\left(-\dfrac{\epsilon^2(d_0+1)r}{1296 \cdot \mathfrak{t}(d_0+\tau_0)}\right)$, we obtain $r_{Y_2} = -1296\dfrac{\ln(\delta/2c_2)}{\epsilon^2} \cdot \dfrac{(d_0+\tau_0)\mathfrak{t}}{d_0+1} = \mathcal{O}\left(\dfrac{(d_0+\tau_0)\mathfrak{t}}{d_0+1}\right)$, where c_2, ϵ and δ are all constants.

Since $\mathfrak{t} \ge \tilde{\mathfrak{t}}$ (see Appendix A in (Hardiman & Katzir, 2013) for details), choose $r \ge \mathcal{O}\left(\dfrac{d_0(d_0+\tau_0)\mathfrak{t}}{\tau_0}\right) \ge \max\{r_{Y_1}, r_{Y_2}\}$. Eq. (13) and Eq. (14) find the number of steps/samples r, which guarantees both Y_1 and Y_2 are within $(1 \pm \epsilon/3)$ of their expected values with the probability of at least $1 - \delta/2$. Since the probability of Y_1 or Y_2 deviating from their expected value is at most $\delta/2$, the probability of either Y_1 or Y_2 deviating is at most δ:

$$\Pr\left(|Y - \mathbb{E}[Y]| > \frac{\epsilon}{3}\mathbb{E}[Y]\right) \le \frac{\delta}{2}, \quad Y = Y_1, Y_2$$

$$\Rightarrow \Pr\left(\left(1 - \frac{\epsilon}{3}\right)\mathbb{E}[Y] \le Y \le \left(1 + \frac{\epsilon}{3}\right)\mathbb{E}[Y]\right) \ge 1 - \frac{\delta}{2}, \quad Y = Y_1, Y_2$$

$$\Rightarrow \Pr\left(\underbrace{(1-\epsilon)\tau_0 \le \frac{d_0+1}{6}\frac{1-\frac{\epsilon}{3}}{1+\frac{\epsilon}{3}}\frac{\mathbb{E}[Y_1]}{\mathbb{E}[Y_2]}}_{\star} \le \underbrace{\frac{d_0+1}{6}\frac{Y_1}{Y_2}}_{\text{estimator } z_0} \le \underbrace{\frac{d_0+1}{6}\frac{1+\frac{\epsilon}{3}}{1-\frac{\epsilon}{3}}\frac{\mathbb{E}[Y_1]}{\mathbb{E}[Y_2]} \le (1+\epsilon)\tau_0}_{\star}\right) \ge 1 - \delta$$

The first line is a summary of Eq. (13) & Eq. (14). The inequalities "$\star$" hold due to Eq. (12), and the fact of $1 - \epsilon \le \dfrac{1-\frac{\epsilon}{3}}{1+\frac{\epsilon}{3}}$ and $1 + \epsilon \ge \dfrac{1+\frac{\epsilon}{3}}{1-\frac{\epsilon}{3}}$ when $0 < \epsilon \le 1/8$. We thus conclude that after $\mathcal{O}\left(\dfrac{d_0(d_0+\tau_0)\mathfrak{t}}{\tau_0}\right)$ samples, $\Pr\left(\left|\dfrac{z_0}{\tau_0} - 1\right| \le \epsilon\right) \ge 1 - \delta$.

However, if $\tau_0 = 0$ and G' is a star graph, the number of samples $r \ge \mathcal{O}\left(\dfrac{d_0(d_0+\tau_0)\mathfrak{t}}{\tau_0}\right) \to \infty$. To avoid that, we add an artificial node v_a and connect it to all nodes in G', as illustrated in Figure 6. Since $d_0^{(a)} = d_0 + 1$, $\tau_0^{(a)} = d_0 + \tau_0$, we only need to minus a d_0 for the estimated result $\tau_0^{(a)}$, and the number of samples can then be reduced to $\mathcal{O}\left(\dfrac{d_0^{(a)}(d_0^{(a)}+\tau_0^{(a)})\mathfrak{t}^{(a)}}{\tau_0^{(a)}}\right) \approx \mathcal{O}\left(\dfrac{d_0(2d_0+\tau_0)\mathfrak{t}}{d_0+\tau_0}\right)$.

We have proved that we can estimate the number of incidence triangles τ_0 for an arbitrary node v_0 based on its n neighbors by a random walk. Consider the random walk as a nonlinear dynamic system, according to the RNNs' universal approximation ability (Theorem H.2), this random walk can be approximated by an RNN to any desired degree of accuracy. Therefore, let the input sequence of RNN follow the random walk above, then the RNN aggregator can mimic this random walk on the subgraph induced by v_0 and its 1-hop neighbors when aggregating, finally outputs $z_0 \approx \tau_0$. This completes the proof. □

Note: This proof is inspired by Hardiman & Katzir (2013) and Chen et al. (2016).

H.3. Analysis of GraphSAGE

Theorem H.3. *Let $x_v \in U, \forall v \in \mathcal{V}$ denote the input features for Algorithm 1 (proposed in GraphSAGE) on graph $G = (\mathcal{V}, \mathcal{E})$, where U is any compact subset of $\mathbb{R}^d$. Suppose that there exists a fixed positive constant $C \in \mathbb{R}^+$ such that $\|x_v - x_{v'}\|_2 > C$ for all pairs of nodes. Then we have that $\forall \epsilon > 0$ there exists a parameter setting Θ^* for Algorithm 1 such that after $K = 4$ iterations*

$$|z_v - c_v| < \epsilon, \forall v \in \mathcal{V},$$

where $z_v \in \mathbb{R}$ are final output values generated by Algorithm 1 and c_v are node clustering coefficients (Hamilton et al., 2017).

According to Theorem H.3, GraphSAGE can approximate the clustering coefficients in a graph to arbitrary precision. In addition, since GraphSAGE with LSTM aggregators is a special case of our proposed Theorem 4, it can also approximate the number of incidence triangles to arbitrary precision. In fact, the number of incidence triangles τ_v is related to the local clustering coefficient c_v. More specifically, $\tau_v = c_v \cdot d_v(d_v - 1)/2$. Therefore, the conclusion of Theorem 4 is consistent with that of Theorem H.3. However, Theorem 4 reveals that the required samples $\mathcal{O}\left(\frac{d_v(2d_v + \tau_v)\mathfrak{t}}{d_v + \tau_v}\right)$ are proportional to the mixing time $\mathfrak{t}$ and related to τ_v, leading to a practically prohibitive aggregation complexity.

To overcome this problem and improve the efficiency, GraphSAGE performs neighborhood sampling and suggests sampling 2-hop neighborhoods for each node. Suppose the neighborhood sample sizes of 1-hop and 2-hop are S_1 and S_2, then the sampling complexity is $\Theta(NS_1S_2)$. Accordingly, the memory and time complexity of GraphSAGE with LSTM are $\Theta(Nc + NS_1S_2)$ and $\Theta(NS_1S_2c^2 + NS_1S_2)$.

I. Proof of Proposition 5

Theorem I.1. *2-WL and MPNNs cannot induced-subgraph-count any connected pattern with 3 or more nodes (Chen et al., 2020).*

Lemma I.2. *No pair of strongly regular graphs in family $SRG(v, r, \lambda, \mu)$ can be distinguished by the 2-FWL test (Bodnar et al., 2021b; Bouritsas et al., 2022).*

Using Theorem I.1 and Lemma I.2, we prove Proposition 5 as follows.

Proposition 5. *PG-GNN is strictly more powerful than the 2-WL test and not less powerful than the 3-WL test.*

Proof. We first verify that the GIN (with the equivalent expressive power as the 2-WL test) (Xu et al., 2019) can be instantiated by a GNN model with RNN aggregators (including our proposed PG-GNN). Consider a single layer of GIN:

$$h_v^{(k)} = \text{MLP}^{(k)}\left(h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right) \tag{15}$$

where $\text{MLP}^{(k)}$ has a linear mapping $W_{\text{GIN}}^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ and a bias term $b_{\text{GIN}}^{(k)} \in \mathbb{R}^{d_k}$. Without loss of generality, we take the Simple Recurrent Network (SRN) (Elman, 1990) as the RNN aggregator in Eq. (3), formulated as follows:

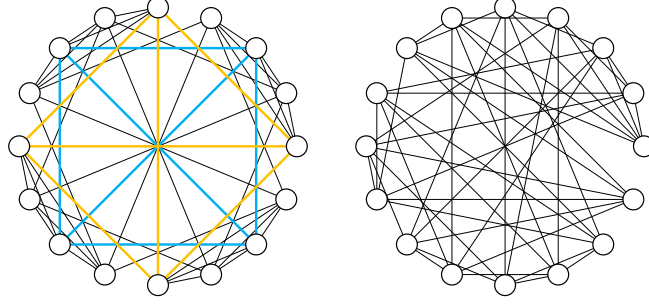$$z_t^{(k)} = U y_{t-1}^{(k)} + W h_t^{(k-1)} + b$$
$$y_t^{(k)} = a(z_t^{(k)})$$

Figure 7. A pair of non-isomorphic strongly regular graphs in the family SRG(16,6,2,2): 4×4 Rook's graph and the Shrikhande graph.

Let $\boldsymbol{W} = \boldsymbol{W}_{\text{GIN}}^{(k)}$, $\boldsymbol{U} = \boldsymbol{I}_{d_k}$, $\boldsymbol{b} = \boldsymbol{b}_{\text{GIN}}^{(k)}$, the initial state $\boldsymbol{y}_0^{(k)} = \boldsymbol{0}$, the activation function $a(\cdot)$ be an identity function. And let the input sequence of the RNN aggregator be an arbitrarily ordered sequence of the set $\{\boldsymbol{h}_u^{(k-1)}\}_{u \in \mathcal{N}(v) \cup v}$. Then any GIN with Eq. (15) can be instantiated by a GNN model with RNN aggregators (in particular, a PG-GNN with Eq. (3)), which implies that the permutation-sensitive GNNs can be at least as powerful as the 2-WL test.

Next, we prove that PG-GNN is strictly more powerful than MPNNs and 2-WL test from the perspective of substructure counting. Without loss of generality, we take an arbitrary node v into consideration. According to the definition of incidence triangles and the fact that they always appear in the 1-hop neighborhood of the central node, the number of connections between neighboring nodes of the central node v is equivalent to the number of incidence triangles over v. Theorem 1 ensures that all the 2-ary dependencies can be modeled by Eq. (3). Suppose we are aiming to capture the connections between two arbitrary neighbors of the central node, we can use an MSE loss to measure the mean squared error between the predicted and ground-truth counting values and guide our model to learn the correct 2-ary dependencies, thereby capturing the correct connections and counting the number of connections between neighboring nodes. And if we mainly focus on specific downstream tasks (e.g., graph classification), these 2-ary dependencies will be learned adaptively with the guidance of a specific loss function (e.g., cross-entropy loss). Thus PG-GNN is capable of counting incidence triangles[2]. Moreover, since the incidence 4-cliques always appear in the 1-hop neighborhood of the central node and every 4-clique is entirely composed of triangles, PG-GNN can also leverage 2-ary dependencies to count incidence 4-cliques, similar to counting incidence triangles. Thus PG-GNN can count all 3-node graphlets (⌴•, ◣), even 4-cliques (▧) incident to node v.

In addition, Chen et al. (2020) proposed Theorem I.1, which implies that 2-WL and MPNNs cannot count any connected induced subgraph with 3 or more nodes. Since the incidence wedges, triangles, and 4-cliques are all connected induced subgraphs with ≥ 3 nodes, the above arguments demonstrate that the expressivity of PG-GNN goes beyond the 2-WL test and MPNNs.

To round off the proof, we finally prove that PG-GNN is not less powerful than the 3-WL test. Consider a pair of strongly regular graphs in the family SRG(16,6,2,2): 4×4 Rook's graph and the Shrikhande graph. As illustrated in Figure 7, only Rook's graph (left) possesses 4-cliques (some are emphasized by colors), but the Shrikhande graph (right) possesses no 4-cliques. Since PG-GNN is capable of counting incidence 4-cliques, our approach can distinguish this pair of strongly regular graphs. However, in virtue of Lemma I.2 and the fact that 2-FWL is equivalent to 3-WL (Maron et al., 2019a), the 3-WL test fails to distinguish them. Thus PG-GNN is not less powerful than the 3-WL test[3].

In conclusion, our proposed PG-GNN is strictly more powerful than the 2-WL test and not less powerful than the 3-WL test. □

---

[2]In fact, since PG-GNN can count incidence triangles, it is also capable of counting all incidence 3-node graphlets. There are only two types of 3-node graphlets, i.e., wedges (⌴•) and triangles (◣). Let τ_v be the number of incidence triangles over v and n be the number of 1-hop neighbors, then we have $\binom{n}{2} - \tau_v$ incidence wedges.

[3]More accurately, PG-GNN is outside the WL hierarchy, and thus it is not easy to fairly compare it with 3-WL. On the one hand, PG-GNN can distinguish some strongly regular graphs but 3-WL fails. On the other hand, 3-WL considers all the 3-tuples $(i_1, i_2, i_3) \in \mathcal{V}^3$, which form a superset of (induced) subgraphs, but PG-GNN only considers the induced subgraphs and thus cannot completely achieve 3-WL. In summary, 3-WL and PG-GNN have their own unique merits. However, since 3-WL needs to consider all $\binom{N}{3} = \Theta(N^3)$ 3-tuples, the problem of complexity is inevitable. In contrast, PG-GNN breaks from the WL hierarchy to make a trade-off between expressive power and computational efficiency.

J. Details of the Proposed Model

In this section, we discuss the proposed model in detail. The notations follow the definitions in Section 3.1, i.e., let n denote the number of 1-hop neighbors of the central node v. Suppose these n neighbors are randomly numbered as $u_1, \ldots, u_n$ (also abbreviated as $1, \ldots, n$ for simplicity), the set of neighboring nodes is represented as $\mathcal{N}(v)$ (or $S = [n] = \{1, \ldots, n\}$).

J.1. Illustration of the Proposed Model

Figure 8 presents a further explanation of Figure 2 and the relationships among Theorem 1, Lemma 2, Corollary 3, Figure 2, and Eq. (3). In this figure, we ignore the central node v for clarity and illustrate for $n = 5$ and $n = 6$. Here we take $n = 5$ as an example to explain Figure 8(a).

The very left column shows the components of Figure 2 and Eq. (3), and the right four columns provide the decoupled illustrations of Figure 2 and Eq. (3). The first row of the right four columns lists the group action gu_i (g acts on u_i) defined by Corollary 3, where u_i ranges from u_1 to u_5, $g \in \mathfrak{G} = \{e, \sigma, \sigma^2, \sigma^3\}$ and $\mathfrak{G}$ is defined by Lemma 2. For readers unfamiliar with group theory, the third row of the right four columns explicitly provides the corresponding action results of gu_i, such as $\sigma^2 u_1 = u_1, \sigma^2 u_2 = u_5, \sigma^2 u_3 = u_4, \sigma^2 u_4 = u_3, \sigma^2 u_5 = u_2$ in the third column. In addition, these four columns are associated with each other by the generator σ. For example, in the third row, after σ acts on the action results in the first column, they are transformed into the action results in the second column according to the permutation diagram, i.e., $\sigma u_1 = u_1, \sigma u_2 = u_4, \sigma u_3 = u_2, \sigma u_4 = u_5, \sigma u_5 = u_3$. Action results in other columns are transformed in a similar manner and form a cyclic structure. The second row of the right four columns illustrates this process.

In each column, after obtaining the action results of $gu_1, \ldots, gu_n$, we arrange these $n = 5$ neighbors (action results) as an *undirected* ring. The first $\lfloor n/2 \rfloor = 2$ arrangements (columns in Figure 8(a), marked by solid lines) are constructed according to Theorem 1, and the last $\lfloor n/2 \rfloor = 2$ arrangements (columns in Figure 8(a), marked by dashed lines) reverse the former. Either the first or the last $\lfloor n/2 \rfloor = 2$ arrangements cover all *undirected* 2-ary dependencies. Then, we use permutation-sensitive RNNs to model the 2-ary dependencies in a *directed* manner (since permutation-sensitive RNNs serve $a \rightarrow b$ and $b \rightarrow a$ as two different pairs) and construct the corresponding Hamiltonian cycles. As a result, the Hamiltonian cycles are modeled bi-directionally, and edges in Hamiltonian cycles are transformed into an undirected manner. The arrangement generation and Hamiltonian cycle construction are detailed in Section 3.3.

Figure 8(b) presents in a similar way as Figure 8(a) does. However, we do not show all six columns due to the limited space. Here we omit the 5th and the 6th columns, which illustrate the modeling processes based on group elements σ^4 and σ^5. Please refer to https://github.com/zhongyu1998/PG-GNN/tree/main/figures for a complete illustration of Figure 8(b).

J.2. Discussion on Groups

Since the (permutation) group in Eq. (3) plays a pivotal role in our model, it is necessary to discuss the motivation for using groups and why we select the specific group. In fact, the group is used to effectively model all 2-ary dependencies (pairwise correlations). We first summarize why the modeling of all 2-ary dependencies is indispensable:

- *Expressive power.* Modeling all 2-ary dependencies can capture whether any two neighboring nodes are connected, helping our model count incidence triangles and 4-cliques hence improving its expressive power.

- *Generalization capability and computational complexity.* Modeling all 2-ary dependencies can make these dependencies invariant to arbitrary permutations of the nodes. Such an invariance to 2-ary dependencies is an approximation of the permutation-invariance and helps to guarantee the generalization capability. Moreover, it also avoids considering all $n!$ permutations to strictly ensure the permutation-invariance, thereby significantly reducing the computational complexity.

- *Robustness.* Modeling all 2-ary dependencies makes our model insensitive to a specific 2-ary dependency and robust to potential data noise and adversarial perturbations.

In order to effectively cover all 2-ary dependencies *with the lowest complexity*, we try to design a special group to accomplish this goal. According to Cayley's Theorem (Theorem F.3) that "Every finite group is isomorphic to a permutation group", we focus on finding permutation groups instead of all finite groups. Hence the problem is converted to finding the basic element of the permutation group, i.e., the permutation. Lemma 2 defines a permutation group $\mathfrak{G}$ and constructs its permutation σ (Eq. (2)) based on Theorem 1, which has been proven to reach the theoretical lower bound of the sampling complexity

(a) $n = 5$
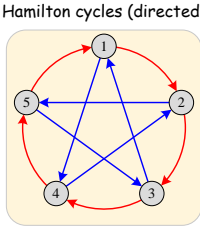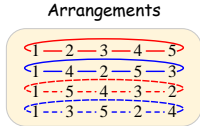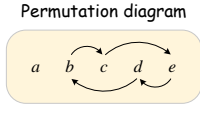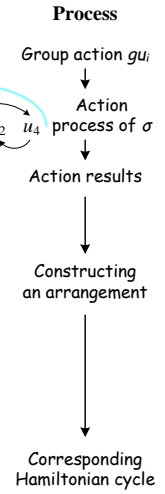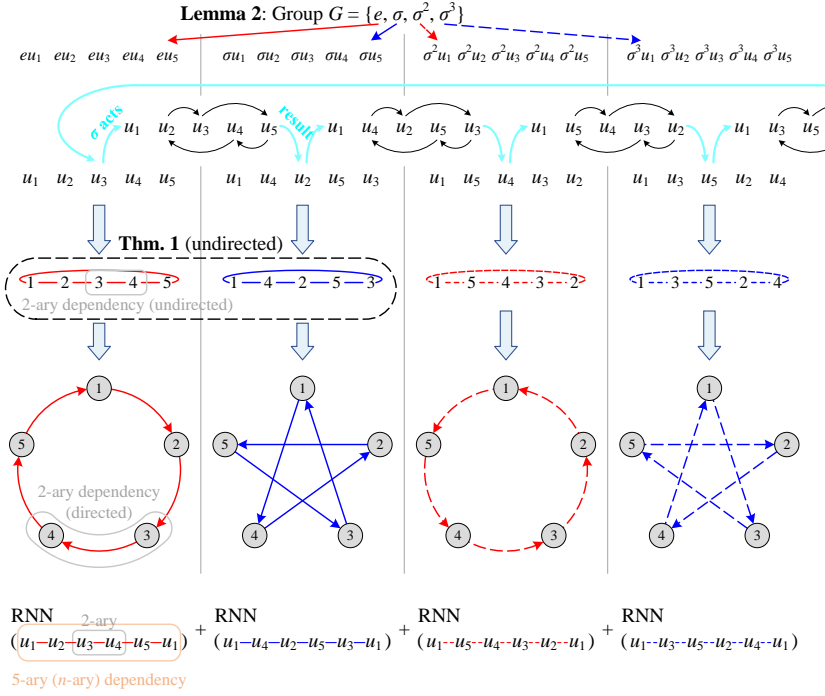
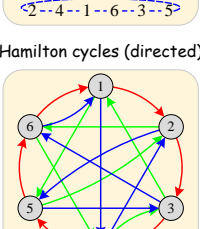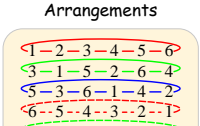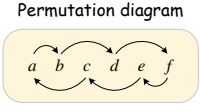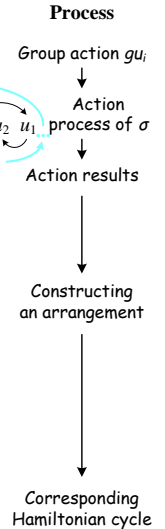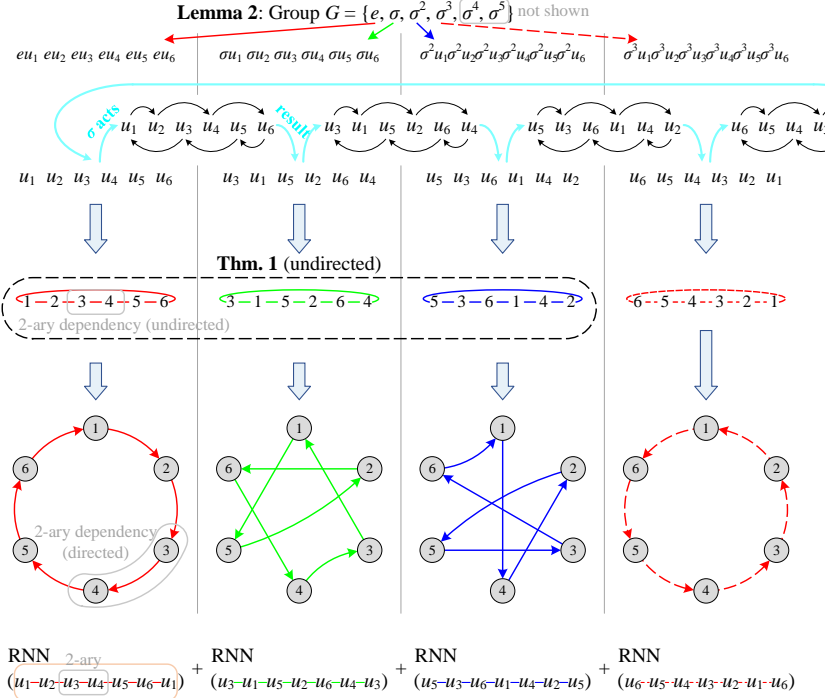(b) $n = 6$

Figure 8. Illustration of the proposed PG-GNN model.

when sampling permutations to cover all 2-ary dependencies. This permutation group is isomorphic to the cyclic group, the simplest group to achieve linear sampling complexity. On the contrary, other groups, such as dihedral group $\mathcal{D}_n$, alternating group $\mathcal{A}_n$, symmetric group $\mathcal{S}_n$, etc., will lead to higher nonlinear complexity hence sacrificing efficiency. Thus, in terms of computational efficiency, group $\mathfrak{G}$ defined in Lemma 2 is the best choice, which drives us to apply it to our model design (Eq. (3)).

J.3. Discussion on Model Variants

Since our proposed model mainly focuses on modeling all 2-ary dependencies, the most intuitive way is to enumerate all $n(n-1)$ *bi-directional* 2-ary dependencies between the n neighbors of the central node v and then sum them up, which can be formulated as follows:

$$h_v^{(k)} = \sum_{\substack{u_i, u_j \in \mathcal{N}(v) \\ u_i \neq u_j}} \text{RNN}\left(h_{u_i}^{(k-1)}, h_{u_j}^{(k-1)}\right) + W_{\text{self}}^{(k-1)} h_v^{(k-1)} \tag{16}$$

Besides, we can also merge the central node v into RNN to form $n(n-1)$ triplets:

$$h_v^{(k)} = \sum_{\substack{u_i, u_j \in \mathcal{N}(v) \\ u_i \neq u_j}} \text{RNN}\left(h_{u_i}^{(k-1)}, h_{u_j}^{(k-1)}, h_v^{(k-1)}\right) \tag{17}$$

In fact, both these two naive variants and our proposed Eq. (3) can model all 2-ary dependencies. However, each term $\left(h_{u_i}^{(k-1)}, h_{u_j}^{(k-1)}\right)$ in Eq. (16) can only capture a 2-ary dependency, and each term $\left(h_{u_i}^{(k-1)}, h_{u_j}^{(k-1)}, h_v^{(k-1)}\right)$ in Eq. (17) can only capture a triplet (3-ary dependency). Contrary to these two naive variants, each term $\left(h_{gu_1}^{(k-1)}, \cdots, h_{gu_n}^{(k-1)}, h_{gu_1}^{(k-1)}\right)$ in Eq. (3) encodes all neighbors as a higher-order n-ary dependency, which contains more information and is more powerful than 2-ary or 3-ary dependency.

On the other hand, we can also integrate all terms of Eq. (3) into only one term, and use a single RNN to model it as follows:

$$h_v^{(k)} = \text{RNN}\left(\Big\|_{g \in \mathfrak{G}} \left(h_{gu_1}^{(k-1)}, h_{gu_2}^{(k-1)}, \cdots, h_{gu_n}^{(k-1)}, h_{gu_1}^{(k-1)}\right)\right) + W_{\text{self}}^{(k-1)} h_v^{(k-1)}, \; u_{1:n} \in \mathcal{N}(v) \tag{18}$$

where $\|$ is the concatenation operation. For example, if n is even, it concatenates $g \in \mathfrak{G}$ as:

$$\Big\|_{g \in \mathfrak{G}} \left(h_{gu_1}^{(k-1)}, h_{gu_2}^{(k-1)}, \cdots, h_{gu_n}^{(k-1)}, h_{gu_1}^{(k-1)}\right) = h_{eu_1}^{(k-1)}, h_{eu_2}^{(k-1)}, \cdots, h_{eu_n}^{(k-1)}, h_{eu_1}^{(k-1)},$$

$$h_{\sigma u_1}^{(k-1)}, h_{\sigma u_2}^{(k-1)}, \cdots, h_{\sigma u_n}^{(k-1)}, h_{\sigma u_1}^{(k-1)},$$

$$\cdots,$$

$$h_{\sigma^{n-1} u_1}^{(k-1)}, h_{\sigma^{n-1} u_2}^{(k-1)}, \cdots, h_{\sigma^{n-1} u_n}^{(k-1)}, h_{\sigma^{n-1} u_1}^{(k-1)}$$

Although this variant can model all $n(n-1)$ 2-ary dependencies in a single term, the time complexity is problematic. Since the concatenation operation orders these representations $h_*^{(k-1)}$, Eq. (18) can only be processed serially with the time complexity of $\Theta(N\Delta^2 c^2)$. This drawback hinders us from effectively balancing the expressive power and computational cost. In contrast, as explained in Section 3.4, our proposed Eq. (3) can be computed in parallel with lower time complexity of $\Theta(N\Delta c^2)$, making it more efficient in practice.

K. Analysis of Sampling Complexity

In this section, we first consider a variant of the coupon collector's problem (Problem K.2) and find the analytical solution to it. Then, we use the solution of Problem K.2 to estimate the sampling complexity of π-SGD optimization (proposed by Janossy Pooling (Murphy et al., 2019b) and Relational Pooling (Murphy et al., 2019a)). Finally, we conduct numerical experiments to verify the rationality of our estimation.

K.1. A Variant of Coupon Collector's Problem

The coupon collector's problem is a famous probabilistic paradigm arising from the following scenario.

Problem K.1 (Coupon Collector's Problem). *Suppose there are m different types of coupons, and each time one chooses a coupon independently and uniformly at random from the m types. One needs to collect $mH(m) = m \ln m + \mathcal{O}(m)$ coupons on average before obtaining at least one of every type of coupon, here $H(m) = \sum_{i=1}^{m} \frac{1}{i}$ is the m-th harmonic number (*Mitzenmacher & Upfal, 2017*).*

In order to estimate the sampling complexity of π-SGD optimization, we need a more sophisticated analysis of the coupon collector's problem. The following problem is the generalization of Problem K.1 from one coupon to $k(k \geq 1)$ coupons at each time, providing a theoretical foundation for our discussion in Section K.2.

Problem K.2 (k-Coupon Collector's Problem). *Suppose there are m different types of coupons, and each time one chooses k coupons ($k \geq 1$, without repetition) independently and uniformly at random from the m types. One needs to collect $\sum_{i=1}^{m} (-1)^{i+1} \frac{\binom{m}{i}}{1 - \binom{m-i}{k}/\binom{m}{k}}$ times on average before obtaining at least one of every type of coupon.*

Proof. Let X be the collecting times until at least one of every type of coupon is obtained. We start by considering the probability that X is greater than s when s is fixed. For $j = 1, \ldots, m$, let A_j denote the event that no type j coupon is collected in the first s times. By the inclusion-exclusion principle,

$$
\begin{aligned}
\Pr(X > s) &= \Pr\left(\bigcup_{j=1}^{m} A_j\right) \\
&= \sum_{1 \leq j_1 \leq m} \Pr(A_{j_1}) - \sum_{1 \leq j_1 < j_2 \leq m} \Pr(A_{j_1} \cap A_{j_2}) + \sum_{1 \leq j_1 < j_2 < j_3 \leq m} \Pr(A_{j_1} \cap A_{j_2} \cap A_{j_3}) \\
&\quad - \cdots + (-1)^{m+1} \Pr(A_1 \cap \cdots \cap A_m) \\
&= \sum_{i=1}^{m} (-1)^{i+1} \sum_{1 \leq j_1 < \cdots < j_i \leq m} \Pr(A_{j_1} \cap \cdots \cap A_{j_i})
\end{aligned}
$$

where $\Pr(A_{j_1} \cap \cdots \cap A_{j_i}) = \left[\dfrac{\binom{m-i}{k}}{\binom{m}{k}}\right]^s$, and for $1 \leq j_1 < \cdots < j_i \leq m$ there are $\binom{m}{i}$ choices. Thus, we have

$$
\Pr(X > s) = \sum_{i=1}^{m} (-1)^{i+1} \binom{m}{i} \left[\frac{\binom{m-i}{k}}{\binom{m}{k}}\right]^s \tag{19}
$$

Since X takes only positive integer values, we can compute its expectation by

$$
\mathbb{E}[X] = \sum_{s=1}^{\infty} s \cdot \Pr(X = s) = \sum_{s=0}^{\infty} \Pr(X > s) \tag{20}
$$

Using Eq. (19) in Eq. (20), we obtain

$$
\begin{aligned}
\mathbb{E}[X] &= \sum_{s=0}^{\infty} \sum_{i=1}^{m} (-1)^{i+1} \binom{m}{i} \left[\frac{\binom{m-i}{k}}{\binom{m}{k}}\right]^s \\
&= \sum_{i=1}^{m} (-1)^{i+1} \binom{m}{i} \sum_{s=0}^{\infty} \left[\frac{\binom{m-i}{k}}{\binom{m}{k}}\right]^s
\end{aligned}
$$

$$= \sum_{i=1}^{m} (-1)^{i+1} \frac{\binom{m}{i}}{1 - \binom{m-i}{k} / \binom{m}{k}} \qquad \square$$

K.2. Sampling Complexity Analysis of π-SGD Optimization

Suppose there are n neighboring nodes around the central node v. π-SGD optimization samples a permutation of these n nodes randomly at each time and models their dependencies based on the sampled permutation. As mentioned in the main body, we are interested in the average times of modeling all the pairwise correlations between these n nodes. This problem can be equivalently formulated in graph-theoretic language as follows:

Problem K.3 (Complete Graph Covering Problem). *Let G' be an empty graph with n nodes. Each time we generate a Hamiltonian path at random and add the corresponding $n-1$ edges to G' (edges can be generated repeatedly at different times). How many times does it take on average before graph G' covers a complete graph K_n?*

It is difficult to give an analytical solution to this problem, so we try to find an approximate solution. In fact, the complete graph covering problem (Problem K.3) has an interesting connection with the k-coupon collector's problem (Problem K.2) discussed above. The generation of a Hamiltonian path among n nodes at each time is equivalent to the drawing of $n-1$ *interrelated* edges[4] from all possible $\frac{n(n-1)}{2}$ edges. Suppose we ignore the interrelations between these $n-1$ edges and each time choose $n-1$ edges *independently*[5] and randomly without repetition. In that case, Problem K.3 will degenerate into a special case of Problem K.2. Thus, we have the following conjecture:

Conjecture K.4. *Suppose there are n neighboring nodes around the central node v, and each time we sample a permutation of these n nodes at random. How many times does it take on average before any two nodes have become neighbors at least once? This problem is equivalent to the complete graph covering problem, which shares a similar result to the k-coupon collector's problem: Suppose there are $m = \frac{n(n-1)}{2}$ different types of coupons, and each time one chooses $k = n-1$ coupons (without repetition) independently and uniformly at random from the m types. How many times does it take on average before obtaining at least one of every type of coupon?*

Since the analytical solution to the k-coupon collector's problem has been given by Problem K.2 in Section K.1, we can use it to approximate the result of Problem K.3 and estimate the sampling complexity of π-SGD optimization. We also conduct extensive numerical experiments to compare the results of Problem K.3 with those of Problem K.2 when n ranges from 1 to 1,000. We consider both undirected and directed cases for Problem K.3, there are $\frac{n(n-1)}{2}$ undirected and $n(n-1)$ bi-directional edges, respectively. Correspondingly, Problem K.2 takes $m = \frac{n(n-1)}{2}$ and $m = n(n-1)$ coupons. For each n, we conduct experiments for 10,000 runs and report the average times of covering these edges/coupons. As shown in Figure 9, Problem K.3 (π-SGD) gives almost the same numerical results as Problem K.2 (the closed-form expression), verifying the rationality of Conjecture K.4. Hence, we conclude the following observation:

Observation K.5. *Suppose that there are n neighboring nodes around the central node v, and each time we sample a permutation of these n nodes at random. Any two nodes have been neighbors at least once after*
$$\sum_{i=1}^{\frac{n(n-1)}{2}} (-1)^{i+1} \frac{\binom{\frac{n(n-1)}{2}}{i}}{1 - \binom{\frac{n(n-1)}{2}-i}{n-1} / \binom{\frac{n(n-1)}{2}}{n-1}} \quad \text{approximately } \mathcal{O}(n \ln n) \text{ times on average.}$$

It is worth noting that our approach only needs $\frac{n}{2}$ times in the undirected cases and n times in the directed cases. According to the conclusion of Observation K.5, if the degree of the central node v is $n = 100$, our approach saves $\frac{100 \ln 100}{100/2} \approx 9$ times compared to π-SGD optimization. If $n = 1000$, e.g., a hub node in the large-scale network dataset, our approach saves $\frac{1000 \ln 1000}{1000/2} \approx 14$ times.

---

[4]They *have to* be in an end-to-end manner, e.g., 1-2, 2-3, 3-4.
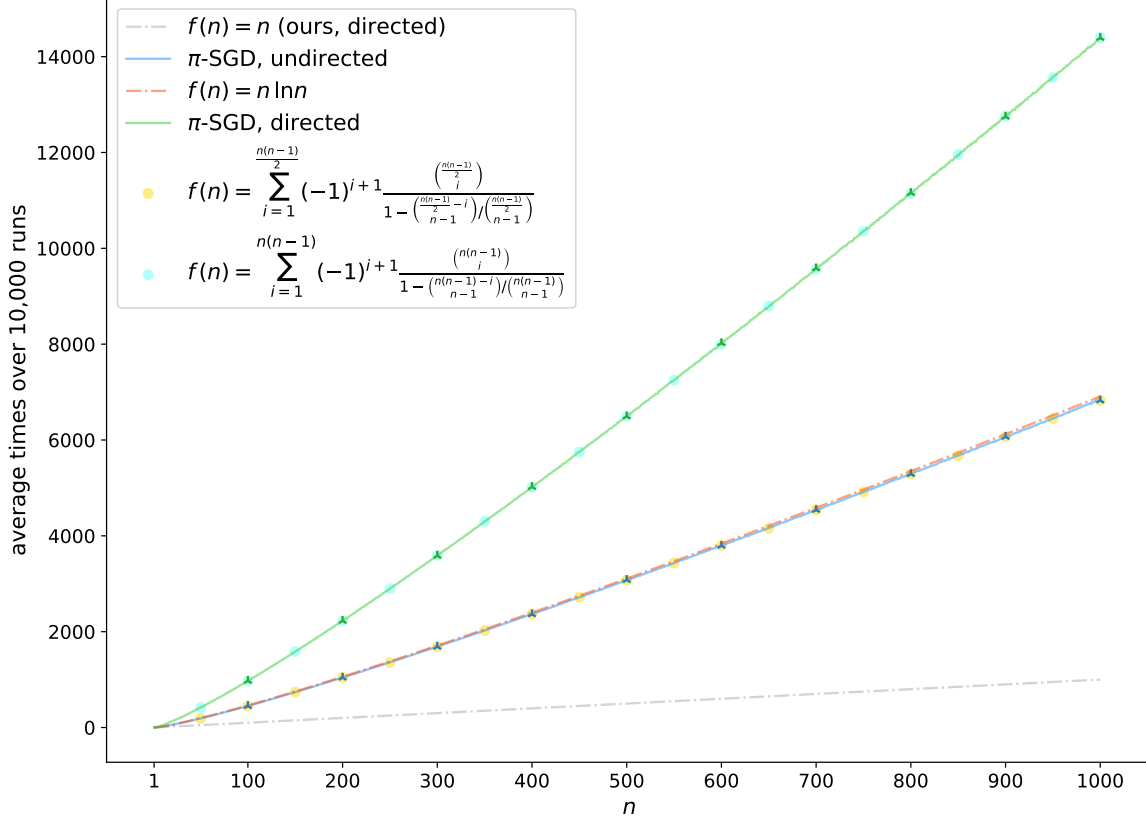[5]They *do not* have to be in an end-to-end manner, e.g., 1-2, 1-3, 1-4.

Figure 9. The experimental results of numerical simulation. Since the results of the complete graph covering problem are equal to the sampling times of π-SGD optimization, we label them as "π-SGD". The blue — and green — lines represent undirected (with $\frac{n(n-1)}{2}$ undirected edges) and directed (with $n(n-1)$ bi-directional edges) cases, respectively. In addition, since the k-coupon collector's problem gives almost the same results as the complete graph covering problem, we only show 20 points (•,•) uniformly for the numerical results of the closed-form expression of the k-coupon collector's problem for clarity. The light yellow • and light blue • points represent undirected ($m = \frac{n(n-1)}{2}$, $k = n - 1$) and directed ($m = n(n-1)$, $k = n - 1$) cases, respectively. We highlight the results of π-SGD at the points that n are multiples of 100 (marked by blue ⋏ and green ⋏ triangular stars) for comparison with those of the k-coupon collector's problem (marked by light yellow • and light blue • points).

L. Analysis of Computational Complexity

In this section, we first provide the time and space complexity analysis of the recent related works MPSN (Bodnar et al., 2021b) and CWN (Bodnar et al., 2021a), then conduct comprehensive experiments about memory consumption to validate the efficiency of our proposed PG-GNN.

L.1. Computational Complexity Analysis of MPSN and CWN

Recently, a batch of works (Bodnar et al., 2021b;a) exploited local high-order interactions to effectively improve the expressive power of GNNs. MPSN (Bodnar et al., 2021b) focuses on simplicial complexes, which are composed of simplices that generalize the 2-dimensional triangle to arbitrary k dimensions. CWN (Bodnar et al., 2021a) further generalizes simplicial complexes of MPSN to cell complexes. Hence these works are particularly relevant to our approach. The core idea of these works is to use the *lifting transformation*, which maps graphs to more distinguishable simplicial complexes (Bodnar et al., 2021b) or cell complexes (Bodnar et al., 2021a) by adding additional structures, e.g., attaching k-simplices ($k \geq 2$) to $(k + 1)$-cliques (Bodnar et al., 2021b) and 2-cells to induced cycles (Bodnar et al., 2021a). Here we mainly discuss the CWN since it is more powerful yet efficient than MPSN.

Let N and M denote the number of nodes and edges, respectively. Let X be a d-dimensional regular cell complex, B_p be the maximum boundary size of a p-cell in X, and S_p be the number of p-cells. For CWN, the time complexity is

Table 5. CPU RAM consumption (MiB) on real-world datasets. The gray font denotes the consumption of the pre-processing stage (i.e., lifting transformation) of CIN.

Model	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB	MNIST	ZINC
GIN	2,338	2,460	2,337	2,343	11,351	24,946	3,017
CIN, pre-proc.	561	627	769	745	N/A	N/A	1,558
CIN, training	2,689	2,749	2,953	3,001	N/A	N/A	2,993
PG-GNN (Ours)	2,343	2,466	2,351	2,349	11,298	24,955	3,020

Table 6. GPU memory consumption (MiB) on real-world datasets. "OOM" means out of memory ($> 24{,}220$MiB).

Model	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB	MNIST	ZINC
GIN	887	889	881	877	1,125	981	901
CIN	2,039	1,033	3,891	13,361	OOM	N/A	1,371
PG-GNN (Ours)	980	1,142	1,202	1,036	21,485	4,127	1,367

$\mathcal{O}\left(\sum_{p=1}^{d}\left(B_p S_p + 2 \cdot \binom{B_p}{2}S_p\right)\right)$, and the space complexity is $\mathcal{O}\left(N + \sum_{p=1}^{d}\left(S_p + B_p S_p + 2 \cdot \binom{B_p}{2}S_p\right)\right)$. Next, we analyze the time complexity for a generic lifting transformation that maps a graph to a 2-dimensional cell complex and attaches 2-cells to all the induced cycles in the graph. Since 0-cells, 1-cells, and 2-cells represent vertices, edges, and induced cycles, respectively, we have $d = 2$, $S_1 = M$, $B_1 = 2$, and B_2 equals the size of the maximum induced cycle considered. In the case of molecular graphs, the number of induced cycles (chemical rings), S_2, is usually upper-bounded by a small constant. Accordingly, CWN achieves outstanding empirical performance and efficiency on molecular tasks. However, in the case of social networks, there are usually $\Omega(N^2)$ triangles (see IMDB-B, IMDB-M, and COLLAB in Table 7), even without mentioning other types of induced cycles. Thus we have $S_2 = \Omega(N^2)$, and the time complexity is $\mathcal{O}\left(4M + B_2 S_2 + 2 \cdot \binom{B_2}{2}S_2\right) \geq \mathcal{O}(N^2)$. To make matters worse, for general graph distributions, S_2 may grow exponentially with the number of nodes (Bodnar et al., 2021a), and the computation of the pre-processing step (lifting transformation) may also be intractable. In a nutshell, the computational complexity may hinder the application of CWN outside of the molecular domain, where the importance of specific substructures is not well understood and their number may grow rapidly.

L.2. Memory Cost Analysis

According to Bodnar et al. (2021a), in all experiments, they employ a model which stacks CWN layers with local aggregators as in GIN, thus naming their architecture "Cell Isomorphism Network" (CIN). Here we use GIN and CIN as our baselines to compare the memory consumption of different models. We use the codes released by the authors of GIN[6] and CIN[7], and run experiments with the (optimal) hyper-parameter configurations reported in their original papers to keep the comparison as fair as possible. Tables 5 and 6 summarize the CPU RAM and GPU memory consumption for various models, respectively. Note that the total CPU RAM consumption of CIN should be computed as the consumption of "pre-processing" + "training", while other models do not require the extra pre-processing steps. As shown in the tables, the memory cost of CIN grows rapidly outside of the molecular domain, such as on social networks and MNIST, consistent with our analysis above. In contrast, our PG-GNN is memory-efficient and outperforms CIN in terms of memory cost on almost all datasets, even performing on par with GIN on most datasets.

M. Details of the Experiments

M.1. Details of Datasets

In this subsection, we provide detailed descriptions of datasets used in our experiments. The statistics of real-world datasets are summarized in Table 7.

---

[6]https://github.com/weihua916/powerful-gnns
[7]https://github.com/twitter-research/cwn

Table 7. Statistics of real-world datasets. The degree denotes in- / out-degree for MNIST containing directed graphs.

Property	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB	MNIST	ZINC
Graphs	1,113	4,110	1,000	1,500	5,000	70,000	12,000
Classes	2	2	2	3	3	10	N/A
Nodes (avg)	39.06	29.87	19.77	13.00	74.49	70.57	23.16
Nodes (max)	620	111	136	89	492	75	37
Degree (avg)	3.73	2.16	9.76	10.14	65.97	8.00 / 8.00	2.15
Degree (max)	25	4	135	88	491	18 / 8	4
Triangles (avg)	27.40	0.05	391.99	305.90	124551.40	626.07	0.06
Triangles (max)	534	3	6,985	14,089	2,574,680	702	2

M.1.1. SYNTHETIC DATASETS

We conduct synthetic experiments of counting incidence substructures on two types of random graphs: Erdős-Rényi random graphs and random regular graphs, created by Chen et al. (2020). The first one consists of 5,000 Erdős-Rényi random graphs with 10 nodes in each graph, and each edge exists with a probability of 0.3. The second one consists of 5,000 random regular graphs with n nodes in each graph and the degree of d, where (n, d) is uniformly sampled from $\{(10, 6), (15, 6), (20, 5), (30, 5)\}$. Both datasets are randomly split into 30%, 20%, and 50% for training, validation, and testing.

For the incidence triangle counting task, all nodes are labeled with Eq. (6). For the incidence 4-clique counting task, it is hard to derive such a closed-form expression as Eq. (6), so we manually label each central node by counting how many groups of three neighboring nodes are fully connected. The evaluation metric of the incidence substructure counting task is the mean absolute error (MAE) between the predicted and true number of incidence substructures for each node.

M.1.2. TUDATASET

Bioinformatics. PROTEINS is a dataset in which each graph represents a protein, and nodes represent secondary structure elements (SSEs) within the protein structure, i.e., *helices*, *sheets*, and *turns*. An edge connects two nodes if they are neighbors in the amino-acid sequence or 3D space. The task is to classify the proteins into enzymes and non-enzymes. NCI1 is a publicly available dataset collected by the National Cancer Institute (NCI). Each graph represents a chemical compound, in which nodes and edges represent atoms and chemical bonds. This dataset is related to anti-cancer screening, and the task is to predict whether the chemical compounds are positive or negative for cell lung cancer.

Social Networks. IMDB-BINARY is a movie-collaboration dataset containing the actor/actress and genre information of different movies on IMDB. Each graph corresponds to an actor/actress's ego network, in which nodes correspond to actors/actresses, and an edge indicates two actors/actresses appear in the same movie. These graphs are derived from *Action* and *Romance* genres. And the task is to classify the graphs into their genres. IMDB-MULTI is the multi-class version of IMDB-BINARY and contains a balanced set of ego networks derived from *Comedy*, *Romance*, and *Sci-Fi* genres. COLLAB is a scientific collaboration dataset. Each graph corresponds to a researcher's ego network, in which nodes correspond to the researcher and its collaborators, and an edge indicates the collaboration between two researchers. These researchers come from different fields, i.e., *High Energy Physics*, *Condensed Matter Physics*, and *Astro Physics*. The task is to classify the graphs into the fields of corresponding researchers.

M.1.3. MNIST AND ZINC

MNIST. MNIST (LeCun et al., 1998) is a classical image classification dataset. The original MNIST images are converted into graphs using super-pixels (Achanta et al., 2012). Each graph represents an image, and its adjacency matrix is built with 8-nearest neighbors for each node (super-pixel). Note that since the relationship between each super-pixel (node) and its nearest neighbors is asymmetric, the resultant adjacency matrices are also asymmetric. For more details about the generation, please refer to Appendix A.2 in Dwivedi et al. (2020). The resultant graphs are of sizes 40-75 super-pixels, and each node's features are assigned with super-pixel coordinates and intensity. MNIST has 55,000 training, 5,000 validation, and 10,000 testing graphs, where the 5,000 graphs for the validation set are randomly sampled from the training set. The evaluation metric for MNIST is the classification accuracy between the predicted class and ground-truth label for each graph.

Table 8. Hyper-parameter configurations on synthetic datasets.

Hyper-parameter	GCN	GraphSAGE	GIN	rGIN	RP	LRP	PG-GNN
batch size	32	32	32	32	16	16	16
hidden units	64	64	64	64	64	64	64
layers	3	3	5	5	5	5	5
dropout	0.5	0.0	0.5	0.5	0.0	0.0	0.0
initial lr	0.01	0.01	0.01	0.01	0.01	0.001	0.001

ZINC. ZINC (Irwin et al., 2012) is one of the most popular real-world molecular datasets with 250K graphs, out of which Dwivedi et al. (2020) randomly select 12K for efficiency. Each graph represents a molecule, where nodes and edges represent atoms and chemical bonds, respectively. The node features are the types of heavy atoms encoded in a one-hot manner. The task is to predict the constrained solubility, an important chemical property for molecules. ZINC has 10,000 training, 1,000 validation, and 1,000 testing graphs. The evaluation metric for ZINC is the mean absolute error (MAE) between the predicted and true constrained solubility for each molecular graph.

M.2. Details of Hyper-Parameters

M.2.1. SYNTHETIC DATASETS

We select the architectural hyper-parameters based on the performance in the validation set. The hyper-parameter search space is listed as follows: the batch size in $\{16, 32, 64\}$, the number of hidden units in $\{16, 32, 64\}$, the number of layers in $\{3, 4, 5\}$, the dropout ratio in $\{0.0, 0.5\}$ after the final prediction layer, the initial learning rate in $\{0.01, 0.005, 0.001\}$, the decay rate in $\{0.5, 0.9\}$, the decay rate patience in $\{5, 10, 15, 20, 25\}$, and the aggregator in $\{$SRN, GRU, LSTM$\}$.

Configurations of Baselines. We use the default hyper-parameter configurations reported in their original papers. Specifically, we follow Hamilton et al. (2017) to sample 2-hop neighborhoods for each node, set the neighborhood sample sizes S_1 and S_2 of 1-hop and 2-hop to both 5, and use LSTM (Hochreiter & Schmidhuber, 1997) as the aggregator in GraphSAGE. We use the uniform distribution over $D = \{0, 0.01, 0.02, \dots, 0.99\}$ as the random distribution μ in rGIN like Sato et al. (2021). We set the dimension m of one-hot node IDs to 10 and use GIN (Xu et al., 2019) as the backbone in RP following Murphy et al. (2019a). According to Chen et al. (2020), we set the depth l and width k to 1 and 3 in LRP. Other hyper-parameters on different models are shown in Table 8.

Configurations of PG-GNN. We report the hyper-parameters chosen by our model selection procedure as follows. For all tasks and datasets, 5 GNN layers (including the input layer) are applied, and the LSTMs with 2 layers are used as the aggregation functions. Batch normalization (Ioffe & Szegedy, 2015) is applied to every hidden layer. All models are initialized using Glorot initialization (Glorot & Bengio, 2010) and trained using the Adam SGD optimizer (Kingma & Ba, 2015) with an initial learning rate of 0.001. If the performance on the validation set does not improve after 20 epochs, the learning rate is then decayed by a factor of 0.5, except for the 4-clique counting task on ER graphs, whose patience is set to 25 epochs. The training is stopped when the learning rate reaches the minimum value of 5E-6.

M.2.2. REAL-WORLD DATASETS

TUDataset. We select the architectural hyper-parameters based on the accuracy in one random training fold. The hyper-parameter search space is listed as follows: the batch size in $\{16, 32, 64\}$, the number of hidden units in $\{8, 16, 32, 64\}$, the number of layers in $\{3, 4, 5\}$, the dropout ratio in $\{0.0, 0.5\}$ after the final prediction layer, the initial learning rate in $\{0.01, 0.005, 0.001\}$, the decay rate in $\{0.5, 0.9\}$, the readout function in $\{$SUM, MEAN$\}$, and the aggregator in $\{$SRN, GRU, LSTM$\}$.

MNIST and ZINC. We select the architectural hyper-parameters based on the performance in the validation set. The hyper-parameter search space is listed as follows: the batch size in $\{32, 64, 128\}$, the number of hidden units in $\{32, 64, 128\}$, the number of layers in $\{3, 4, 5\}$, the dropout ratio in $\{0.0, 0.5\}$ after the final prediction layer, the initial learning rate in $\{0.01, 0.005, 0.001\}$, the decay rate in $\{0.5, 0.9\}$, the decay rate patience in $\{5, 10, 15, 20, 25\}$, the readout function in $\{$SUM, MEAN$\}$, and the aggregator in $\{$SRN, GRU, LSTM$\}$.

Table 9. Hyper-parameter configurations on real-world datasets.

Hyper-parameter	PROTEINS	NCI1	IMDB-B	IMDB-M	COLLAB	MNIST	ZINC
batch size	16	32	16	32	32	64	64
hidden units	8	32	16	16	64	128	128
layers	5	5	5	5	3	5	5
dropout	0.5	0.0	0.0	0.5	0.5	0.0	0.0
degree	False	False	True	True	True	N/A	N/A
readout	SUM	SUM	SUM / MEAN	SUM	SUM / MEAN	MEAN	SUM

Configurations. We report the hyper-parameters chosen by our model selection procedure as follows. For all datasets, 3 or 5 GNN layers (including the input layer) are applied, and the LSTMs with 2 layers are used as the aggregation functions. Batch normalization (Ioffe & Szegedy, 2015) is applied to every hidden layer. All models are initialized using Glorot initialization (Glorot & Bengio, 2010) and trained using the Adam SGD optimizer (Kingma & Ba, 2015) with an initial learning rate of 0.001. For TUDataset, the learning rate is decayed by a factor of 0.5 every 50 epochs. The training is stopped when the number of epochs reaches the maximum value of 400. For MNIST and ZINC, if the performance on the validation set does not improve after 20 and 25 epochs, the learning rate is then decayed by a factor of 0.5. The training is stopped when the learning rate reaches the minimum value of 5E-6. Other hyper-parameters on different datasets are shown in Table 9.

M.3. Computing Infrastructures

Hardware Infrastructures. The experiments are conducted on Linux servers equipped with an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, 256GB RAM and 8 NVIDIA TITAN RTX GPUs.

Software Infrastructures. All models are implemented using Python version 3.6, NetworkX version 2.4 (Hagberg et al., 2008), PyTorch version 1.4.0 (Paszke et al., 2019) with CUDA version 10.0.130, and cuDNN version 7.6.5. In addition, the benchmark datasets are loaded by Deep Graph Library (DGL) version 0.4.2 (Wang et al., 2019).