How Powerful are K-hop Message Passing Graph Neural Networks

Jiarui Feng[1,2] **Yixin Chen**[1] **Fuhai Li**[2] **Anindya Sarkar**[1] **Muhan Zhang**[3,4]
{feng.jiarui,fuhai.li,anindya}@wustl.edu,
chen@cse.wustl.edu, muhan@pku.edu.cn
[1]Department of CSE, Washington University in St. Louis
[2]Institute for Informatics, Washington University School of Medicine
[3]Institute for Artificial Intelligence, Peking University
[4]Beijing Institute for General Artificial Intelligence

Abstract

The most popular design paradigm for Graph Neural Networks (GNNs) is 1-hop message passing—aggregating information from 1-hop neighbors repeatedly. However, the expressive power of 1-hop message passing is bounded by the Weisfeiler-Lehman (1-WL) test. Recently, researchers extended 1-hop message passing to K-hop message passing by aggregating information from K-hop neighbors of nodes simultaneously. However, there is no work on analyzing the expressive power of K-hop message passing. In this work, we theoretically characterize the expressive power of K-hop message passing. Specifically, we first formally differentiate two different kernels of K-hop message passing which are often misused in previous works. We then characterize the expressive power of K-hop message passing by showing that it is more powerful than 1-WL and can distinguish almost all regular graphs. Despite the higher expressive power, we show that K-hop message passing still cannot distinguish some simple regular graphs and its expressive power is bounded by 3-WL. To further enhance its expressive power, we introduce a KP-GNN framework, which improves K-hop message passing by leveraging the peripheral subgraph information in each hop. We show that KP-GNN can distinguish many distance regular graphs which could not be distinguished by previous distance encoding or 3-WL methods. Experimental results verify the expressive power and effectiveness of KP-GNN. KP-GNN achieves competitive results across all benchmark datasets.

1 Introduction

Currently, most existing graph neural networks (GNNs) follow the *message passing* framework, which iteratively aggregates information from the neighbors and updates the representations of nodes. It has shown superior performance on graph-related tasks [1, 2, 3, 4, 5, 6, 7] comparing to traditional graph embedding techniques [8, 9]. However, as the procedure of message passing is similar to the 1-dimensional Weisfeiler-Lehman (1-WL) test [10], the expressive power of message passing GNNs is also bounded by the 1-WL test [7]. Namely, GNNs cannot distinguish two non-isomorphic graph structures if the 1-WL test would fail.

In normal message passing GNNs, the node representation is updated by the direct neighbors of the node, which are called 1-hop neighbors. Recently, some works extend the notion of message passing into K-hop message passing [11, 12, 13, 14, 15]. **K-hop message passing** is a type of message passing where the node representation is updated by aggregating information from not only 1st hop but all the neighbors within K hops of the node. However, there is no work on theoretically

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

characterizing the expressive power of GNNs with K-hop message passing, e.g., whether it can improve the 1-hop message passing or not, and to what extent it can.

In this work, we theoretically characterize the expressive power of K-hop message passing GNNs. Specifically, 1) we formally distinguish two different kernels of the K-hop neighbors, which are often misused in previous works. The first kernel is based on whether the node can be reached within k steps of the graph diffusion process and it is used in GPR-GNN [14] and MixHop [11]. The second one is based on the shortest path distance of k and it is used in GINE+ [15] and Graphormer [16]. Further, we show that different kernels of K-hop neighbors will result in **different expressive power** of K-hop message passing. 2) We show that **K-hop message passing is strictly more powerful than 1-hop message passing and can distinguish almost all regular graphs. 3) However, it still failed in distinguishing some simple regular graphs, no matter which kernel is used, and its expressive power is bounded by 3-WL**. This motivates us to further improve K-hop message passing.

Here, we introduce **KP-GNN**, a new GNN framework with K-hop message passing, which significantly improves the expressive power of standard K-hop message passing GNNs. In particular, during the aggregation of neighbors in each hop, KP-GNN not only aggregates neighboring nodes in that hop but also aggregates the **peripheral subgraph** (subgraph induced by the neighbors in that hop). This additional information helps the KP-GNN to learn more expressive local structural features around the node. We further show that KP-GNN is able to distinguish many distance regular graphs with a proper encoder for the peripheral subgraph. The proposed KP-GNN has several additional advantages. First, it can be applied to most existing K-hop message-passing GNNs with only slight modification. Second, it only adds little computational complexity to standard K-hop message passing. We demonstrate the effectiveness of the KP-GNN framework through extensive experiments on both simulation and real-world datasets.

2 K-hop message passing and its expressive power

2.1 Notations

Denote a graph as $G = (V, E)$, where $V = \{1, 2, ..., n\}$ is the node set and $E \subseteq V \times V$ is the edge set. Meanwhile, denote $A \in \{0, 1\}^{n\times n}$ as the adjacency matrix of graph G. Denote x_v as the feature vector of node v and denote e_{uv} as the feature vector of the edge from u to v. Finally, we denote $Q^1_{v,G}$ as the set of 1-hop neighbors of node v in graph G and $\mathcal{N}^1_{v,G}$=$Q^1_{v,G} \cup \{v\}$. Note that when we say K-hop neighbors of node v, we mean **all** the neighbors that have distance from node v less than or equal to K. In contrast, k-th hop neighbors mean the neighbors with **exactly** distance k from node v. The definition of distance will be discussed in section 2.3.

2.2 1-hop message passing framework

Currently, most existing GNNs are designed based on 1-hop message passing framework [17]. Denote h^l_v as the output representation of node v at layer l and $h^0_v = x_v$. Briefly, given a graph G and a 1-hop massage passing GNN, at layer l of the GNN, h^l_v is computed by h^{l-1}_v and $\{\!\{h^{l-1}_u \mid u \in Q^1_{v,G}\}\!\}$:

$$m^l_v = \text{MES}^l(\{\!\{(h^{l-1}_u, e_{uv}) | u \in Q^1_{v,G}\}\!\}), \;\; h^l_v = \text{UPD}^l(m^l_v, h^{l-1}_v), \tag{1}$$

where m^l_v is the message to node v at layer l, MES^l and UPD^l are message and update functions at layer l respectively. After L layers of message passing, h^L_v is used as the final node representation of node v. Such a representation can be used to conduct node-level tasks like node classification and node regression. To get the graph representation, a readout function is used:

$$h_G = \text{READOUT}(\{\!\{h^L_v | v \in V\}\!\}), \tag{2}$$

where READOUT is the readout function for computing the final graph representation. Then h_G can be used to conduct graph-level tasks like graph classification and graph regression.

2.3 K-hop message passing framework

The 1-hop message passing framework can be directly generalized to K-hop message passing, as it shares the same message and update mechanism. The difference is that an independent message

and update function can be employed for each hop. Meanwhile, a combination function is needed to combine the results from different hops into the final node representation at this layer. First, we differentiate two different kernels of K-hop neighbors, which are interchanged and misused in previous research.

The first kernel of K-hop neighbors is *shortest path distance (spd) kernel*. Namely, the k-th hop neighbors of node v in graph G is the set of nodes that have the shortest path distance of k from v.

Definition 1. *For a node v in graph G, the K-hop neighbors $\mathcal{N}_{v,G}^{K,spd}$ of v based on shortest path distance kernel is the set of nodes that have the shortest path distance from node v* ***less than or equal to*** *K. We further denote $Q_{v,G}^{k,spd}$ as the set of nodes in G that are* ***exactly*** *the k-th hop neighbors (with shortest path distance of exactly k) and $\mathcal{N}_{v,G}^{0,spd} = Q_{v,G}^{0,spd} = \{v\}$ is the node itself.*

The second kernel of the K-hop neighbors is based on *graph diffusion (gd)*.

Definition 2. *For a node v in graph G, the K-hop neighbors $\mathcal{N}_{v,G}^{K,gd}$ of v based on graph diffusion kernel is the set of nodes that can diffuse information to node v* ***within the number of random walk diffusion steps*** *K and the diffusion kernel A. We further denote $Q_{v,G}^{k,gd}$ as the set of nodes in G that are* ***exactly*** *the k-th hop neighbors (nodes that can diffuse information to node v with k diffusion steps) and $\mathcal{N}_{v,G}^{0,gd} = Q_{v,G}^{0,gd} = \{v\}$ is the node itself.*

Note that a node can be a k-th hop neighbor of v for multiple k based on the graph diffusion kernel but it can only appear in one hop for the shortest path distance kernel. We include more discussions of K-hop kernels in Appendix A. Next, we define the K-hop message passing framework as follows:

$$\begin{aligned} m_v^{l,k} = \ \mathrm{MES}_k^l(\{\!\{(h_u^{l-1}, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}), \ \ h_v^{l,k} = \ \mathrm{UPD}_k^l(m_v^{l,k}, h_v^{l-1}), \\ h_v^l = \ \mathrm{COMBINE}^l(\{\!\{h_v^{l,k} | k = 1, 2, ..., K\}\!\}), \end{aligned} \tag{3}$$

where $t \in \{spd, gd\}$, spd is the shortest path distance kernel and gd is the graph diffusion kernel. Here, for each hop, we can apply unique MES and UPD functions. Note that for $k > 1$, there may not exist the edge feature e_{uv} as nodes are not directly connected. But we leave it here since we can use other types of features to replace it like path encoding. We further discuss it in Appendix I. Comparing to the 1-hop message passing framework described in Equation (1), the COMBINE function is introduced to combine the representations of node v at different hops. It is easy to see that a L layer 1-WL GNNs is actually a L layer K-hop message passing GNNs with $K = 1$. We include more discussions of K-hop message passing GNNs in Appendix A. To aid further analysis, we also prove that K-hop message passing can injectively encode the neighbor representations at different hops into h_v^l in Appendix B.

2.4 Expressive power of K-hop message passing framework

In this section, we theoretically analyze the expressive power of K-hop message passing. We assume there is no edge feature and all nodes in the graph have the same feature, which means that GNNs can only distinguish two nodes with the local structure of nodes. Note that including node features only increases the expressive power of GNNs as nodes/graphs are more easily to be discriminated. It has been proved that the expressive power of 1-hop message passing is bounded by the 1-WL test on discriminating non-isomorphic graphs [7, 18]. In this section, We show that the K-hop message passing is strictly more powerful than 1-WL test when $K > 1$. Across the analysis, we utilize regular graphs as examples to illustrate our theorems since they cannot be distinguished using either 1-hop message passing or the 1-WL test. To begin the analysis, we first define *proper K-hop message passing GNNs*.

Definition 3. *A proper K-hop message passing GNN is a GNN model where the message, update, and combine functions are all injective given the input from a countable space.*

A proper K-hop message passing GNN is easy to find due to the universal approximation theorem [19] of neural network and the Deep Set for set operation [20]. In the latter sections, by default all mentioned K-hop message passing GNNs are proper. Next, we introduce node configuration:

Definition 4. *The node configuration of node v in graph G within k hops under t kernel is a list $A_{v,G}^{k,t} = (a_{v,G}^{1,t}, a_{v,G}^{2,t}, ..., a_{v,G}^{k,t})$, where $a_{v,G}^{i,t} = |Q_{v,G}^{i,t}|$ is the number of i-th hop neighbors of node v.*

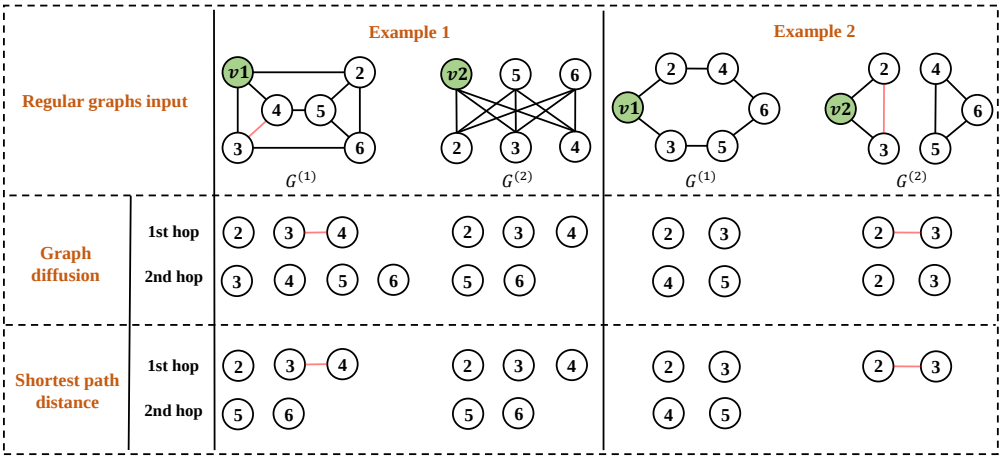

Figure 1: Here are two pairs of non-isomorphic regular graphs. With 2-hop message passing, example 1 can be distinguished by the graph diffusion kernel, and example 2 can be distinguished by the shortest path distance kernel. However, both two examples become indistinguishable if we switch the kernel. Finally, both two examples can be distinguished by adding peripheral subgraph information.

When we say two node configurations $A^{k,t}_{v_1,G^{(1)}}$ and $A^{k,t}_{v_2,G^{(2)}}$ are equal, we mean that these two lists are component-wise equal to each other. Now, we state the first proposition:

Proposition 1. *A proper K-hop message passing GNN is strictly more powerful than 1-hop message passing GNNs when $K > 1$.*

To see why this is true, we first discuss how node configuration is related to the first layer of message passing. In the first layer of K-hop message passing, each node aggregates neighbors from up to K hops. As each node has an identical label, an injective message function can only know how many neighbors at each hop, which is exactly the node configuration. In other words, **The first layer of K-hop message passing is equivalent to inject node configuration to each node label**. When $K = 1$, the node configuration of v_1 and v_2 are $d_{v_1,G^{(1)}}$ and $d_{v_2,G^{(2)}}$, where $d_{v,G}$ is the node degree of v. After L layers, GNNs can only get the node degree information of each node within L hops of node v. Then, it is straightforward to see why these GNNs cannot distinguish any n-sized r-regular graph, as each node in the regular graph has the same degree.

Next, when $K > 1$, the K-hop message passing is at least equally powerful as 1-hop message passing since node configuration up to K hop includes all the information the 1 hop has. To see why it is more powerful, we use two examples to illustrate it. The first example is shown in the left part of Figure 1. Suppose here we use graph diffusion kernel and we want to learn the representation of node v_1 and node v_2 in the two graphs, we know that the 1-hop GNNs produce the same representation for two nodes as they are both nodes in 6-sized 3-regular graphs. However, it is easy to see that v_1 and v_2 have different local structures and should have different representations. Instead, if we use the 2-hop message passing with the graph diffusion kernel, we can easily distinguish two nodes by checking the 2nd hop neighbors of the node, as node v_1 has four 2nd hop neighbors but node v_2 only has two 2nd hop neighbors. The second example is shown in the right part of Figure 1. Two graphs in the example are still regular graphs. Suppose here we use shortest path distance kernel, node v_1 and v_2 have different numbers of 2nd hop neighbors and thus will have different representations by performing 2-hop message passing. These two examples convincingly demonstrate that the K-hop message passing with $K > 1$ can have better expressive power than $K = 1$. To further study the expressive power of K-hop message passing on regular graphs, we show the following result:

Theorem 1. *Consider all pairs of n-sized r-regular graphs, let $3 \leq r < (2log2n)^{1/2}$ and ϵ be a fixed constant. With at most $K = \lfloor (\frac{1}{2} + \epsilon) \frac{\log 2n}{\log (r-1)} \rfloor$, there exists a 1 layer K-hop message passing GNN using the shortest path distance kernel that distinguishes almost all $1 - o(n^{-1/2})$ such pairs of graphs.*

We include the proof and simulation results in Appendix C. Theorem 1 shows that even with 1 layer and a modest K, K-hop GNNs are powerful enough for distinguishing almost all regular graphs.

Finally, we characterize the existing K-hop methods with the proposed K-hop message passing framework. Specifically, we show that 1) the expressive power of K layer GINE [15] is bounded by K layer K-hop message passing with the shortest path distance kernel. 2) The expressive power of the Graphormer [16] is equal to K-hop GNNs with the shortest path distance kernel and infinity K. 3) For spectral GNNs and existing K-hop GNNs with the graph diffusion kernel like MixHop [11] and MAGNA [13], we find they actually use a **weak version of K-hop than the definition of us**. Specifically, it is shown that the expressive power of spectral GNNs is also bounded by 1-WL test [21], which seems to contradict our result as graph diffusion can be viewed as a special case of spectral GNN. However, we show that our definition of K-hop message passing with graph diffusion kernel actually injects a non-linear function on the spectral basis, thus achieving superior expressive power. We leave the detailed discussion in Appendix D. Further, Distance Encoding [22] also uses the shortest path distance information to augment the 1-hop message passing, which is similar to K-hop GNNs with the shortest path distance kernel. However, we find the expressive power of the two frameworks is different from each other. We leave the detailed discussion in Appendix E.

2.5 Limitation of K-hop message passing framework

Although we show that K-hop GNNs with $K > 1$ are better at distinguishing non-isomorphic structures than 1-hop GNNs, there are still limitations. In this section, we discuss the limitation of K-hop message passing. Specifically, we show that **the choice of the kernel can affect the expressive power of K-hop message passing**. Furthermore, even with K-hop message passing, we still cannot distinguish some simple non-isomorphic structures and **the expressive power of K-hop message passing is bounded by 3-WL**.

Continue looking at the provided examples. In example 1, if we use the shortest path distance kernel instead of the graph diffusion kernel, two nodes have the same number of neighbors in the 2nd hop, which means that we cannot distinguish two nodes this time. Similarly, in example 2, two nodes have the same number of neighbors in both 1st and 2nd hops with graph diffusion kernel. These results highlight that the choice of the kernel can affect the expressive power of K-hop message passing and none of them can distinguish both two examples with 2-hop message passing.

Recently, Frasca et al. [23] show that any subgraph-based GNNs with node-based selection policy can be implemented by 3-IGN [24, 25] and thus their expressive power is bounded by 3-WL test. Here, we show that the K-hop message passing GNNs can also be implemented by 3-IGN for both two kernels and thus:

Theorem 2. *The expressive power of a proper K-hop message passing GNN of any kernel is bounded by the 3-WL test.*

We include the proof in Appendix F. Given all these observations, we may wonder if there is a way to further improve the expressive power of K-hop message passing?

3 KP-GNN: improving the power of K-hop message passing by peripheral subgraph

In this section, we describe how to improve the expressive power of K-hop message passing by adding additional information to the message passing framework. Specifically, by adding the *peripheral subgraph* information, we can improve the expressive power of the K-hop message passing by a large margin.

3.1 Peripheral edge and peripheral subgraph

First, we define *peripheral edge* and *peripheral subgraph*.

Definition 5. *The peripheral edge $E(Q^{k,t}_{v,G})$ is defined as the set of edges that connect nodes within set $Q^{k,t}_{v,G}$. We further denote $|E(Q^{k,t}_{v,G})|$ as the number of peripheral edge in $E(Q^{k,t}_{v,G})$. The peripheral subgraph $G^{k,t}_{v,G} = (Q^{k,t}_{v,G}, E(Q^{k,t}_{v,G}))$ is defined as the subgraph induced by $Q^{k,t}_{v,G}$ from the whole graph G.*

Briefly speaking, the peripheral edge $E(Q^{k,t}_{v,G})$ record all the edges whose two ends are both from $Q^{k,t}_{v,G}$ and the peripheral subgraph is a graph constituted by peripheral edges. It is easy to see that the peripheral subgraph $G^{k,t}_{v,G}$ automatically contains all the information of peripheral edge $E(Q^{k,t}_{v,G})$. Next, we show that the power of K-hop message passing can be improved by leveraging the information of peripheral edges and peripheral subgraphs. We again refer to the examples in Figure 1. Here we only consider the peripheral edge information. In example 1, we notice that at the 1st hop, there is an edge between node 3 and node 4 in the left graph. More specifically, $E(Q^{1,t}_{v_1,G^{(1)}}) = \{(3,4)\}$. In contrast, we have $E(Q^{1,t}_{v_2,G^{(2)}}) = \{\}$ in the right graph, which means there is no edge between the 1st hop neighbors of v_2. Therefore, by adding this information to the message passing, we can successfully distinguish these two nodes. Similarly, in example 2, there is one edge between the 1st hop neighbors of node v_2 but no such edge exists for node v_1. By leveraging peripheral edge information, we can distinguish the two nodes as well. The above examples demonstrate the effectiveness of the peripheral edge and peripheral subgraph information.

3.2 K-hop peripheral-subgraph-enhanced graph neural network

In this section, we propose **K**-hop **P**eripheral-subgraph-enhanced **G**raph **N**eural **N**etwork (KP-GNN), which equips K-hop message passing GNNs with peripheral subgraph information for more powerful GNN design. Recall the K-hop message passing defined in Equation (3). The only difference between KP-GNN and original K-hop GNNs is that we revise the message function as follows:

$$m^{l,k}_v = \text{MES}^l_k(\{\!\{(h^{l-1}_u,\ e_{uv}) | u \in Q^{k,t}_{v,G}\}\!\},\ G^{k,t}_{v,G}). \tag{4}$$

Briefly speaking, in the message step at the k-th hop, we not only aggregate information of the neighbors but also the peripheral subgraph at that hop. The implementation of KP-GNN can be very flexible as any graph encoding function can be used. To maximize the information the model can encode while keeping it simple, we implement the message function as:

$$\begin{aligned} \text{MES}^l_k =&\ \text{MES}^{l,normal}_k(\{\!\{(h^{l-1}_u, e_{uv}) | u \in Q^{k,t}_{v,G}\}\!\}) + f(G^{k,t}_{v,G}), \\ f(G^{k,t}_{v,G}) =&\ \text{EMB}((E(Q^{k,t}_{v,G}), C^{k'}_k))\ , \end{aligned} \tag{5}$$

where $\text{MES}^{l,normal}_k$ denotes the message function in the original GNN model, $C^{k'}_k$ is the k'-configuration, which encode both node configuration and the number of peripheral edge of all nodes in $G^{k,t}_{v,G}$ up to k' hops. It can be regarded as running another 1 layer KP-GNN on each peripheral subgraph. EMB is a learnable embedding function. With this implementation, any GNN model can be incorporated into and be enhanced by the KP-GNN framework by replacing $\text{MES}^{l,normal}_k$ and UPD^l_k with the corresponding functions for each hop k. We leave the detailed implementation in Appendix I.

3.3 The expressive power of KP-GNN and comparison with existing methods

In this section, we theoretically characterize the expressive power of KP-GNN and compare it with the original K-hop message passing framework. The key insight is that, according to Equation (4), the message function at the k-th hop additionally encodes $G^{k,t}_{v,G}$ compared to normal K-hop message passing. As we have already shown in the last section, K-hop GNNs are bounded by 3-WL and thus cannot distinguish any non-isomorphic distance regular graphs, as well as Distance Encoding [22]. Let $C^{k'}_{j,G}$ be the k'-configuration of peripheral subgraph at j-th hop of nodes in distance regular graph G. Here we show that with simple information from peripheral subgraph, KP-GNN is able to distinguish distance regular graphs:

Proposition 2. *For two non-isomorphic distance regular graphs $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$ with the same diameter d and intersection array $(b_0, b_1, ..., b_{d-1}; c_1, c_2, ..., c_d)$. Given a proper 1-layer d-hop KP-GNN with message functions defined in Equation (5), it can distinguish $G^{(1)}$ and $G^{(2)}$ if $C^{k'}_{j,G^{(1)}} \neq C^{k'}_{j,G^{(2)}}$ for some $0 < j \leq d$.*

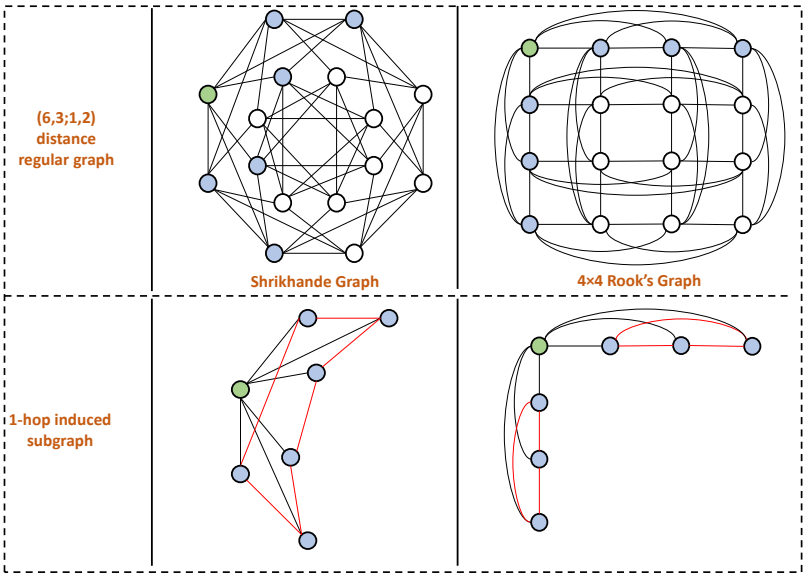

Figure 2: An example of two non-isomorphic distance regular graph with intersection array $(6, 3; 1, 2)$

We include the proof in Appendix G. Here we leverage an example in Figure 2 to briefly show why KP-GNN is able to distinguish distance regular graphs. Figure 2 displays two distance regular graphs with intersection array of $(6, 3; 1, 2)$. The left one is Shrikhande graph and the right one is the 4×4 Rook's graph. Now, let's look at the 1-hop peripheral subgraph of the green node. In the Shrikhande graph, there are 6 peripheral edges marked with red. Further, 6 edges constitute a circle. In the 4×4 Rook's graph, there are still 6 peripheral edges. However, 6 edges constitute two circles with 3 edges in each circle, which is different from the Shrikhande graph. Then, any peripheral subgraph encoder that can distinguish these two graphs like node configuration enables the corresponding KP-GNN to distinguish the example. Proposition 2 shows that the KP-GNN is capable of distinguishing distance regular graphs, which further distinguishes KP-GNN from DE-1 [22] as it cannot distinguish any two connected distance regular graphs with the same intersection arrays according to Theorem 3.7 in [22]. However, it is currently unknown whether can KP-GNN with Equation 5 distinguish all distance regular graphs. We leave the detailed discussion in Appendix G.

Moreover, both the subgraph-based GNNs like NGNN [26], GNNAK [27], ESAN [28], and KP-GNN leverage the information in the subgraph to enhance the power of message passing. However, KP-GNN is intrinsically different from them. Firstly, in KP-GNN, the message passing is performed on the whole graph instead of the subgraphs. This means that for each node, there is only one representation to be learned. Instead, for subgraph-based GNNs, the message passing is performed separately for each subgraph and each node could have multiple representations depending on which subgraph it is in. Secondly, in subgraph-based GNNs, they consider the subgraph as a whole without distinguishing nodes at different hops. Instead, KP-GNN takes one step further by dividing the subgraph into two parts. The first part is the hierarchy of neighbors at each hop. The second part is the connected structure between nodes in each hop. This gives us a better point of view to design a more powerful learning method. From the Corollary 7 in [23], we know that all subgraph-based GNNs with node selection as subgraph policy is bounded by 3-WL, which means they cannot distinguish any distance regular graph and KP-GNN is better at it.

3.4 Time, space complexity, and limitation

In this section, we discuss the time and space complexity of K-hop message passing GNN and KP-GNN. Suppose a graph has n nodes and m edges. Then, the K-hop message passing and KP-GNN have both the space complexity of $O(n)$ and the time complexity of $O(n^2)$ for the shortest path distance kernel. Note that the complexity of graph diffusion is no less than the shortest path distance kernel. **We can see that KP-GNN only requires the same space complexity as vanilla GNNs and much less time complexity than the subgraph-based GNNs, which are at least $O(nm)$.** However, K-hop message passing including KP-GNN still have intrinsic limitation. We leave a detailed discussion on the complexity and limitation of KP-GNN in Appendix H.

4 Related Work

Expressive power of GNN. Analyzing the expressive power of GNNs is a crucial problem as it can serve as a guide on how to improve GNNs. Xu et al. [7] and Morris et al. [18] first proved that the power of 1-hop message passing is bounded by the 1-WL test. In other words, 1-hop message passing cannot distinguish any non-isomorphic graphs that the 1-WL test fails to. In recent years, many efforts have been put into increasing the expressive power of 1-hop messaging passing. The first line of research tries to mimic the higher-order WL tests, like 1-2-3 GNN [18], PPGN [24], ring-GNN [29]. However, they require exponentially increasing space and time complexity w.r.t. node number and cannot be generalized to large-scale graphs. The second line of research tries to enhance the rooted subtree of 1-WL with additional features. Some works [30, 31, 32] add one-hot or random features into nodes. Although they achieve good results in some settings, they deteriorate the generalization ability as such features produce different representations for nodes even with the same local graph structure. Some works like Distance Encoding [22], SEAL [33], labeling trick [34] and GLASS [35] introduce node labeling based on either distance or distinguishing target node set. On the other hand, GraphSNN [36] introduces a hierarchy of local isomorphism and proposes structural coefficients as additional features to identify such local isomorphism. However, the function designed to approximate the structural coefficient cannot fully achieve its theoretical power. The third line of research resorts to subgraph representation. Specifically, ID-GNN [37] extracts ego-netwok for each node and label the root node with a different color. NGNN [26] encodes a rooted subgraph instead of a rooted subtree by subgraph pooling thus achieving superior expressive power on distinguishing regular graphs. GNN-AK [27] applies a similar idea as NGNN. The only difference lies in how to compute the node representation from the local subgraph. However, such methods need to run an inner GNN on every node of the graph thus introducing much more computation overhead. Meanwhile, the expressive power of subgraph GNNs are bounded by 3-WL [23].

K-hop message passing GNN. There are some existing works that instantiate the K-hop message passing framework. For example, MixHop [11] performs messing passing on each hop with graph diffusion kernel and concatenates the representation on each hop as the final representation. K-hop [12] sequentially performs the message passing from hop K to hop 1 to compute the representation of the center node. However, it is not parallelizable due to its computational procedure. MAGNA [13] introduces an attention mechanism to K-hop message passing. GPR-GNN [14] use graph diffusion kernel to perform graph convolution on K-hop and aggregate them with learnable parameters. However, none of them give a formal definition of K-hop message passing and theoretically analyze its representation power and limitations.

5 Experiments

In this section, we conduct extensive experiments to evaluate the performance of KP-GNN. Specifically, we 1) empirically verify the expressive power of KP-GNN on 3 simulation datasets and demonstrate the benefits of KP-GNN compared to normal K-hop message passing GNNs; 2) demonstrate the effectiveness of KP-GNN on identifying various node properties, graph properties, and substructures with 3 simulation datasets; 3) show that the KP-GNN can achieve state-of-the-art performance on multiple real-world datasets; 4) analyze the running time of KP-GNN. The detail of each variant of KP-GNN is described in Appendix I and the detailed experimental setting is described in Appendix J. We implement the KP-GNN with PyTorch Geometric package [38]. Our code is available at `https://github.com/JiaruiFeng/KP-GNN`.

Datasets: To evaluate the expressive power of KP-GNN, we choose: 1) EXP dataset [39], which contains 600 pairs of non-isomorphic graphs (1-WL failed). The goal is to map these graphs to two different classes. 2) SR25 dataset [40], which contains 15 non-isomorphic strongly regular graphs with each graph of 25 nodes. The dataset is translated to a 15-way classification problem with the goal of mapping each graph into different classes. 3) CSL dataset [41], which contains 150 4-regular graphs (1-WL failed) divided into 10 isomorphism classes. The goal of the task is to classify them into corresponding isomorphism classes. To demonstrate the capacity of KP-GNN on node/graph properties and substructures, we pick 1) Graph property regression (connectedness, diameter, radius) and node property regression (single source shortest path, eccentricity, Laplacian feature) task on graph random dataset [42]. 2) Graph substructure counting (triangle, tailed triangle, star, and 4-cycle) tasks on random graph dataset [43]. To evaluate the performance of KP-GNN on real-world dataset,

we select 1) MUTAG [44], D&D [45], PROTEINS [45], PTC-MR [46], and IMDB-B [47] from TU database. 2) QM9 [48, 49] and ZINC [50] for molecular properties prediction. The detailed statistics of the datasets are described in Appendix K. Without further highlighting, all error bars in the result tables are the standard deviations of multiple runs (more details are in Appendix J).

Table 1: Empirical evaluation of the expressive power.

Method	K	EXP (ACC)		SR (ACC)		CSL (ACC)	
		SPD	GD	SPD	GD	SPD	GD
K-GIN	K=1	50	50	6.67	6.67	12	12
	K=2	50	50	6.67	6.67	32	22.7
	K=3	100	66.9	6.67	6.67	62	42
	K=4	100	100	6.67	6.67	92.7	62.7
KP-GIN	K=1	50	50	100	100	22	22
	K=2	100	100	100	100	52.7	52.7
	K=3	100	100	100	100	90	90
	K=4	100	100	100	100	100	100

Empirical verification of the expressive power: For empirical evaluation of the expressive power, we conduct the ablation study on hop K for both normal K-hop GNNs and KP-GNN. For K-hop GNNs, we implement K-GIN which uses GIN [7] as base encoder. For KP-GNN, we implement KP-GIN. The results are shown in Table 1. Based on the results, we have the following conclusions: 1) K-hop GNNs with both two kernels have expressive power higher than the 1-WL test as it shows the perfect performance on the EXP dataset and performance better than a random guess on CSL dataset. 2) Increasing K can improve the expressive for both two kernels. 3) K-hop GNNs cannot distinguish any strong regular graphs in SR25 dataset, which is aligned with Theorem 2. 4) KP-GNN has much higher expressive power than normal K-hop GNNs by showing better performance on every dataset given the same K. Further, it achieves perfect results on the SR25 dataset even with $K = 1$, which demonstrates its ability on distinguishing distance regular graphs.

Table 2: Simulation dataset result. The top two are highlighted by **First**, **Second** .

Method	Node Properties ($\log_{10}$(MAE))			Graph Properties ($\log_{10}$(MAE))			Counting Substructures (MAE)			
	SSSP	Ecc.	Lap.	Connect.	Diameter	Radius	Tri.	Tailed Tri.	Star	4-Cycle
GIN	-2.0000	-1.9000	-1.6000	-1.9239	-3.3079	-4.7584	0.3569	0.2373	0.0224	0.2185
PNA	**-2.8900**	**-2.8900**	-3.7700	-1.9395	3.4382	-4.9470	0.3532	0.2648	0.1278	0.2430
PPGN	-	-	-	-1.9804	-3.6147	-5.0878	**0.0089**	**0.0096**	0.0148	**0.0090**
GIN-AK+	-	-	-	**-2.7513**	**-3.9687**	-5.1846	0.0123	0.0112	0.0150	0.0126
K-GIN+	-2.7919	-2.5938	**-4.6360**	-2.1782	**-3.9695**	**-5.3088**	0.1316	0.0813	**0.0017**	0.0916
KP-GIN+	**-2.7969**	**-2.6169**	**-4.7687**	**-4.4322**	-3.9361	**-5.3345**	**0.0012**	**0.0016**	**0.0014**	**0.0067**

Effectiveness on node/graph properties and substructure prediction: To evaluate the effectiveness of KP-GNN on node/graph properties and substructure prediction, we compare it with several existing models. For the baseline model, we use GIN [7], which has the same expressive power as 1-WL test. For more powerful baselines, we use GIN-AK+ [27], PNA [42] and PPGN [24]. For normal K-hop GNNs, we implement K-GIN+, and for KP-GNN, we implement KP-GIN+. The results are shown in Table 2. Baseline results are taken from [27] and [42]. We can see KP-GIN+ achieve SOTA on almost all tasks with a large margin to other baseline methods. Meanwhile, K-GIN+ also get great performance, especially on node/graph properties prediction. These results demonstrate the capability of KP-GNN to identify various properties and substructures.

Table 3: TU dataset evaluation result.

Method	MUTAG	D&D	PTC-MR	PROTEINS	IMDB-B
WL	90.4±5.7	79.4±0.3	59.9±4.3	75.0±3.1	73.8±3.9
GIN	89.4±5.6	-	64.6±7.0	75.9±2.8	75.1±5.1
DGCNN	85.8±1.7	79.3 ±0.9	58.6 ±2.5	75.5±0.9	70.0±0.9
GraphSNN	91.24±2.5	82.46±2.7	66.96±3.5	76.51±2.5	76.93±3.3
GIN-AK+	91.30±7.0	-	68.20±5.6	77.10±5.7	75.60±3.7
KP-GCN	91.7±6.0	79.0±4.7	67.1±6.3	75.8±3.5	75.9±3.8
KP-GraphSAGE	91.7±6.5	78.1±2.6	66.5±4.0	76.5±4.6	76.4±2.7
KP-GIN	92.2±6.5	79.4±3.8	66.8±6.8	75.8±4.6	76.6±4.2
GIN-AK+*	95.0±6.1	OOM	74.1±5.9	78.9±5.4	77.3±3.1
GraphSNN*	94.70±1.9	**83.93±2.3**	70.58±3.1	78.42±2.7	78.51±2.8
KP-GCN*	**96.1±4.6**	83.2±2.2	**77.1±4.1**	80.3±4.2	79.6±2.5
KP-GraphSAGE*	**96.1±4.6**	83.6±2.4	76.2±4.5	**80.4±4.3**	80.3±2.4
KP-GIN*	95.6±4.4	83.5±2.2	76.2±4.5	79.5±4.4	**80.7±2.6**

Table 4: QM9 results. The top two are highlighted by **First**, **Second**

Target	DTNN	MPNN	Deep LRP	PPGN	Nested 1-2-3-GNN	KP-GIN+
μ	**0.244**	0.358	0.364	**0.231**	0.433	0.367
α	0.95	0.89	0.298	0.382	**0.265**	**0.242**
$\varepsilon_{\text{HOMO}}$	0.00388	0.00541	**0.00254**	0.00276	0.00279	**0.00247**
$\varepsilon_{\text{LUMO}}$	0.00512	0.00623	0.00277	0.00287	**0.00276**	**0.00238**
$\Delta\varepsilon$	0.0112	0.0066	**0.00353**	0.00406	0.00390	**0.00345**
$\langle R^2 \rangle$	17.0	28.5	19.3	**16.7**	20.1	**16.49**
ZPVE	0.00172	0.00216	0.00055	0.00064	**0.00015**	**0.00018**
U_0	2.43	2.05	0.413	0.234	**0.205**	**0.0728**
U	2.43	2.00	0.413	0.234	**0.200**	**0.0553**
H	2.43	2.02	0.413	**0.229**	0.249	**0.0575**
G	2.43	2.02	0.413	0.238	**0.253**	**0.0526**
C_v	0.27	0.42	0.129	0.184	**0.0811**	**0.0973**

Evaluation on TU datasets: For baseline models, we select: 1) graph kernel-based method: WL subtree kernel [51]; 2) vanilla GNN methods: GIN [7] and DGCNN [6]; 3) advanced GNN methods: GraphSNN [36] and GIN-AK+ [27]. For the proposed KP-GNN, we implement GCN [1], GraphSAGE [3], and GIN [7] using the KP-GNN framework, denoted as KP-GCN, KP-GraphSAGE, and KP-GIN respectively. The results are shown in Table 3. For more fair and comprehensive comparison, we report the results from two different evaluation settings. The first setting follows Xu et al. [7] and the second setting follows Wijesinghe and Wang [36]. We denote the second setting with * in the table. We can see KP-GNN achieves state-of-the-art performance on all datasets under the second setting and still comparable performance to other baselines under the first setting.

Table 5: ZINC result.

Method	# param.	test MAE
MPNN	480805	0.145±0.007
PNA	387155	0.142±0.010
Graphormer	489321	0.122±0.006
GSN	~500000	0.101±0.010
GIN-AK+	-	0.080±0.001
CIN	-	**0.079±0.006**
KP-GIN+	499099	0.111±0.006

Evaluation on molecular prediction tasks: For QM9 dataset, we report baseline results of DTNN and MPNN from [49]. We further select Deep LRP [43], PPGN [24], and Nested 1-2-3-GNN [26] as baseline models. For ZINC dataset, we report results of MPNN [17] and PNA [42] from [50]. We further pick Graphormer [16], GSN [52], GIN-AK+ [27], and CIN [53]. The results of QM9 dataset are shown in Table 4. We can see KP-GNN achieves state-of-the-art performance on most of the targets. The results of ZINC dataset is shown in Table 5. Although KP-GNN does not achieve the best result, it is still comparable to other methods.

Table 6: Running time (s/epoch).

Method	D&D	ZINC	Graph property
GIN	1.10	3.59	1.02
K-GIN	3.94	6.44	1.67
KP-GIN	4.19	7.38	1.94
KP-GIN+	4.28	6.74	1.93

Running time comparison: In this section, we compare the running time of KP-GNN to 1-hop message passing GNN and K-hop message passing GNN. We use GIN [7] as the base model. We also include the KP-GIN+. All models use the same number of layers and hidden dimensions for a fair comparison. The results are shown in Table 6. We set $K = 4$ for all datasets. We can see the computational overhead is almost linear to K. This is reasonable as practical graphs are sparse and the number of K-hop neighbors is far less than n when using a small K.

6 Conclusion

In this paper, we theoretically characterize the power of K-hop message passing GNNs and propose the KP-GNN to improve the expressive power by leveraging the peripheral subgraph information at each hop. Theoretically, we prove that K-hop GNNs can distinguish almost all regular graphs but are bounded by the 3-WL test. KP-GNN is able to distinguish many distance regular graphs. Empirically, KP-GNN achieves competitive results across all simulation and real-world datasets.

7 Acknowledgement

This work is partially supported by NSF grant CBE-2225809 and NSF China (No. 62276003).

References

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

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

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

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

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

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

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

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

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

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

[11] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

[12] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020.

[13] Guangtao Wang, Zhitao Ying, Jing Huang, and Jure Leskovec. Multi-hop attention graph neural network, 2021. URL `https://openreview.net/forum?id=muppfCkU9H1`.

[14] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=n6jl7fLxrP`.

[15] Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

[16] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=OeWooOxFwDa`.

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

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

[19] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[20] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

[21] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks, 2022. URL `https://arxiv.org/abs/2205.11172`.

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

[23] Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. *ArXiv*, abs/2206.11140, 2022.

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

[25] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[26] Muhan Zhang and Pan Li. Nested graph neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=7_eLEvFjCi3`.

[27] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Mspk_WYKoEH`.

[28] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=dFbKQaRk15w`.

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

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

[31] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization, 2021. URL `https://openreview.net/forum?id=L7Irrt5sMQa`.

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

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

[34] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=Hcr9mgBG6ds`.

[35] Xiyuan Wang and Muhan Zhang. GLASS: GNN with labeling tricks for subgraph representation learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=XLxhEjKNbXj`.

[36] Asiri Wijesinghe and Qing Wang. A new perspective on "how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=uxgg9o7bI_3`.

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

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

[39] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

[40] Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[41] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

[42] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

[43] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 2020.

[44] Asim Kumar Debnath, de Compadre RL Lopez, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[45] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[46] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10): 1183–1193, 2003.

[47] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[48] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[49] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[50] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[51] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep): 2539–2561, 2011.

[52] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting, 2021. URL https://openreview.net/forum?id=LTOKSFnQDWF.

[53] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montufar, and Michael M. Bronstein. Weisfeiler and lehman go cellular: CW networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=uVPZCMVtsSG.

[54] Béla Bollobás. Distinguishing vertices of random graphs. In Béla Bollobás, editor, *Graph Theory*, volume 62 of *North-Holland Mathematics Studies*, pages 33–49. North-Holland, 1982. doi: https://doi.org/10.1016/S0304-0208(08)73545-X. URL https://www.sciencedirect.com/science/article/pii/S030402080873545X.

[55] Béla Bollobás. A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics*, 1(4):311–316, December 1980. ISSN 0195-6698. doi: 10.1016/S0195-6698(80)80030-8. URL https://www.sciencedirect.com/science/article/pii/S0195669880800308.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[57] Floris Geerts. The expressive power of kth-order invariant graph networks. *ArXiv*, abs/2007.12035, 2020.

[58] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://aclanthology.org/D15-1166.

[59] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *CoRR*, abs/1806.03536, 2018. URL http://arxiv.org/abs/1806.03536.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The implementation of KP-GNN can be found at https://github.com/anoncodesub/KP-GNN.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The license can be found in their github.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All datasets we used are all open-sourced.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] There is no personal information in the datasets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

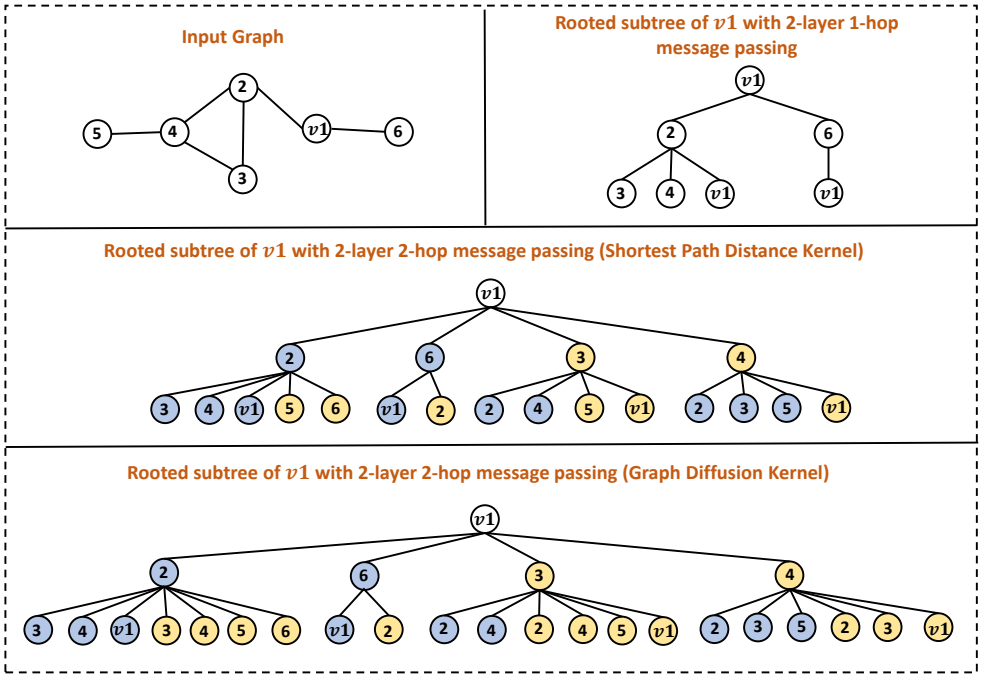

Figure 3: The rooted subtree of node $v1$ with 1-hop message passing and K-hop message passing. Here we assume that $K = 2$ and the number of layers is 2.

A More about the K-hop kernel and K-hop message passing

In this section, we further discuss two different types of K-hop kernel and K-hop message passing.

A.1 More about K-hop kernel

First, recall the shortest path distance kernel and graph diffusion kernel defined in Definition 1 and 2. Given two definitions, the first thing we can conclude is that the K-hop neighbors of node v under two different kernels will be the same, namely $\mathcal{N}_{v,G}^{K,spd} = \mathcal{N}_{v,G}^{K,gd}$ as both two kernels capture all nodes that can be reached from node v within the length of K. Second, we have $\mathcal{N}_{v,G}^{1,spd} = Q_{v,G}^{1,spd} = \mathcal{N}_{v,G}^{1,gd} = Q_{v,G}^{1,gd}$, which means the neighbor set is same for both the shortest path distance kernel and the graph diffusion kernel when $K = 1$. The third thing is that $Q_{v,G}^{k,spd}$ will not always equal to $Q_{v,G}^{k,gd}$ for some k. Since for shortest path distance kernel, one node will only appear in at most one of $Q_{v,G}^{k,spd}$ for $k = 1, 2, ..., K$. Instead, nodes can appear in multiple $Q_{v,G}^{k,gd}$. This is the key reason why choice of the kernel can affect the expressive power of K-hop message passing.

A.2 More about K-hop message passing

Here, we use an example shown in Figure 3 to illustrate how K-hop message passing works and compare it with 1-hop message passing. The input graph is shown on the left top of the figure. Suppose we want to learn the representation of node $v1$ using 2 layer message passing GNNs. First, if we perform 1-hop message passing, it will encode a 2-height rooted subtree, which is shown on the right top of the figure. Note that each node is learned using the same set of parameters, which is indicated by filling each node with the same color (white in the figure). Now, we consider performing 2-hop message passing with the shortest path distance kernel. The rooted subtree of node $v1$ is shown in the middle of the figure. we can see that at each height, both 1st hop neighbors and 2nd hop neighbors are included. Furthermore, different sets of parameters are used for different hops, which is indicated by filling nodes in the different hops with different colors (blue for 1st hop and yellow for 2nd hop). Finally, at the bottom of the figure, we show the 2-hop message passing graph neural network with graph diffusion kernel. It is easy to see the rooted subtree is different from the one that uses the shortest path distance kernel, as nodes can appear in both 1st hop and 2nd hop of neighbors.

B Proof of injectiveness of Equation (3)

In this section, we formally prove that Equation (3) is an injective mapping of the neighbor representations at different hops. As here each layer l is doing exactly the same procedure, we only need to prove it for one iteration. Therefore we ignore the superscript and rewrite Equation (3) as:

$$\begin{aligned} m_v^k = \text{MES}_k(\{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}), \;\; h_v^k = \text{UPD}_k(m_v^k, h_v), \\ \hat{h}_v = \text{COMBINE}(\{\!\{h_v^k | k = 1, 2, ..., K\}\!\}) \end{aligned} \tag{6}$$

Next, we state the following proposition:

Proposition 3. *There exist injective functions MES$_k$, UPD$_k$, $k = 1, 2, ..., K$, and an injective multiset function COMBINE, such that $\hat{h}_v$ is an injective mapping of $\{\!\{(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v) \mid k = 1, 2, ..., K\}\!\}$.*

Proof. The existence of injective message passing (MES$_k$,UPD$_k$) and multiset pooling (COMBINE) functions are well proved in [7]. So below we prove the injectiveness of $\hat{h}_v$. First, we combine MSE$_k$ and UPD$_k$ together into ϕ_k:

$$h_v^k = \phi_k(\{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v). \tag{7}$$

Note that ϕ_k is still injective as the composition of injective functions is injective. Next, we need to prove that h_v^k is an injective mapping of $(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v)$. To prove it, we rewrite the function $\phi_k(\cdot)$ into $\phi(k)(\cdot)$, that is, ϕ is an injective function taking k as input and outputs a function $\phi_k = \phi(k)$. We let $\phi(k_1)(x_1)$ and $\phi(k_2)(x_2)$ output different values for $k_1 \neq k_2$ given any input x_1, x_2, e.g., always let the final output dimension be k. Then, we can rewrite h_v^k as

$$\begin{aligned} h_v^k &= \phi(k)(\{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v) \\ &= \psi(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v), \end{aligned} \tag{8}$$

where we have composed two injective functions $\phi(\cdot)$ and $\phi(k)(\cdot)$ into a single one $\psi(\cdot)$. Since given different k, $\phi(k)(\cdot)$ always outputs distinct values for any input, and given fixed k, $\phi(k)(\cdot)$ always outputs different values for different $(\{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v)$, the resulting $\psi(\cdot)$ always outputs different values for different $(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v)$, i.e., it injectively maps $(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v)$. Thus, we have proved that h_v^k is an injective mapping of $(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v)$.

Finally, since COMBINE is an injective multiset function, we conclude that its output $\hat{h}_v$ is an injective mapping of $\{\!\{(k, \{\!\{(h_u, e_{uv}) | u \in Q_{v,G}^{k,t}\}\!\}, h_v) \mid k = 1, 2, ..., K\}\!\}$. □

C Proof of Theorem 1 and simulation result

C.1 Proof of Theorem 1

Here we restate Theorem 1: Consider all pairs of n-sized r-regular graphs, let $3 \leq r < (2log2n)^{1/2}$ and ϵ be a fixed constant. With at most $K = \lfloor (\frac{1}{2} + \epsilon) \frac{\log 2n}{\log (r-1)} \rfloor$, there exists a 1 layer K-hop message passing GNN using the shortest path distance kernel that distinguishes almost all $1 - o(n^{-1/2})$ such pairs of graphs.

As we state in the main paper, 1 layer K-hop GNN is equivalent to inject node configuration for each node label. Therefore, it is sufficient to show that given node configuration $A_{v,G}^{K,spd}$ for all $v \in V$, we can distinguish almost every pair of regular graphs. First, we introduce the following lemma.

Lemma 1. *For two graphs $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$ that are randomly and independently sampled from n-sized r-regular graphs with $3 \leq r < (2log2n)^{1/2}$. We pick two nodes v_1 and v_2 from two graphs respectively. Let $K = \lfloor (\frac{1}{2} + \epsilon) \frac{\log 2n}{\log (r-1)} \rfloor$ where ϵ is a small constant, $A^{K,spd}_{v_1,G^{(1)}} = A^{K,spd}_{v_2,G^{(2)}}$ with the probability at most $o(n^{-3/2})$.*

Proof. This Lemma can be obtained based on Theorem 6 in [54] with minor corrections. Here we state the proof.

We first introduce the configuration model [55] of n-sized r-regular graphs. Suppose we have n disjoint sets of items, $W_i, i \in \{1, 2, ..., n\}$, where each set has r items and corresponds to one node in the configuration model. A configuration is a partition of all nr items into $\frac{nr}{2}$ pairs. Denote by Ω the set of configurations and turn it into a probability space with each configuration the same probability. Turns out among all configurations in Ω, given $r < (2logn)^{1/2}$, there are about $\exp(-\frac{r^2-1}{4})$ or $\Omega(n^{-1/2})$ portion of them are simple r-regular graphs [55]. Then for these configurations, If there is a pair containing items W_i and W_j, then there is an edge between node i and j in the corresponding r-regular graph.

Let $l_0 = \lfloor (\frac{1}{2} + \epsilon) \frac{\log n}{\log (r-1)} \rfloor$. We first look at two nodes randomly selected from the configuration model and consider the following procedure to generate a graph. Let node i and node j be the selected nodes. In the first step, we select all the edges that directly connect to node i and node j. Then we have all nodes that are at a distance of 1 to i or j. In the second step, we select all the edges that connect to nodes at a distance of 1 to either node i or node j. Doing this iteratively for $n-1$ steps and we end up with the union of components of i and j in a random configuration.

We call an edge is *indispensable* if it is the first edge that ensure a node k is at certain distance to $\{i, j\}$. Similarly, an edge is *dispensable* if 1) both two ends of the edge are connected to either i or j; 2) nodes in both two ends of the edge already have at least one edge. Note that edges with both two ends in the same node are dispensable. As the first $k-1$ edges selected so far can connect to at most $k+1$ nodes, the probability that the k-th edge selected is dispensable is at most:

$$\frac{(k+1)(r-1)}{(n-1)r}$$

Therefore the probability that more than 2 of the first $k_o = \lfloor n^{1/6} \rfloor$ edges are dispensable is at most:

$$\binom{k_o}{3} \left(\frac{k_o}{n-k_o} \right)^3 = o(n^{-2}) \tag{9}$$

The probability that more than $l_1 = \lfloor n^{1/8} \rfloor$ of the first $k_1 = \lfloor n^{6/13} \rfloor$ edges are dispensable is at most

$$\binom{k_1}{l_1+1} \left(\frac{k_1}{n-k_1} \right)^{l_1+1} = o(n^{-2}) \tag{10}$$

The probability that more than $l_2 = \lfloor n^{5/13} \rfloor$ of the first $k_2 = \lfloor n^{2/3} \rfloor$ edges are dispensable is at most

$$\binom{k_2}{l_2+1} \left(\frac{k_2}{n-k_2} \right)^{l_2+1} = o(n^{-2}) \tag{11}$$

Now, let A be the event that at most 2 of the first k_o, at most l_1 of the first k_1, and at most l_2 of the first k_2 edges be dispensable. Given Equation (9)-(11), we know the probability of A is $1 - o(n^{-2})$. Therefore, it is sufficient to prove Lemma condition on A is $o(n^{-3/2})$.

Now, we consider another way of generating a graph with configuration model. Similar to the above, suppose we have selected all the edges connecting to at least one node with a distance less than k from $\{i, j\}$ after the k-th step. At the $k+1$-th step, we first select all the edges with one node at a distance of k from node i. Next, we select all the edges with both two ends at a distance of k from node j. It is easy to see that after these two procedures, the only edges left for completing the $k+1$-th step are the edges that have one end at the distance of k from the node j and another end at the distance of $k+1$ from the node j. Suppose there are t_k edges in the nodes at distance k from

node j that have not been generated so far and there are s_k nodes that have not generate any edge yet. Then the final procedure of completing the $k+1$-th step is to connect t_k edges to s_k nodes. This procedure goes on for every k to generate the final graph. Assuming ϵ in the statement of Lemma satisfies that $0 < \epsilon < \frac{1}{8}$. Then we have

$$2 + r(r-1)^{l_0} \leq 2 + rn^{1/2+\epsilon} < n^{2/3}$$

Now let us assume that A holds. It is easily seen that then for $k \leq l_0$, we have:

$$t_k \geq (r-1)^{k-3} \quad \text{and} \quad s_k \geq n/2 \tag{12}$$

Now, after the first two procedures of $k+1$-th step, we have all the nodes at the distance of $k+1$ from node i, which means we already determined $Q_{i,G}^{k+1,spd}$. If $|Q_{i,G}^{k+1,spd}| = |Q_{j,G}^{k+1,spd}|$, then the connection of t_k edges must belong to $|Q_{i,G}^{k+1,spd}|$ nodes from totally s_k nodes. The probability of $|Q_{i,G}^{k+1,spd}| = |Q_{j,G}^{k+1,spd}|$ condition on Equation (12) is at most the maximum of the probability that t_k edges connect to l nodes from s_k nodes. This probability is bounded by:

$$\max_l P(|Q_{j,G}^{k+1,spd}| = l) \leq c_o \frac{s_k^{1/2}}{t_k}, \tag{13}$$

where $t_k \leq c s_k^{5/8}$ for some constant c and c_o is also a constant. The proof of this can be found in Lemma 7 of [55]. Given Equation (13), the probability that $A_{i,G}^{l,spd} = A_{j,G}^{l,spd}$ for $l \leq l_0$ is at most

$$1 - P(A) + \prod_h^{l_0} c_0 \frac{n^{1/2}}{(r-1)^{l-3}},$$

where $h = \lfloor \frac{1}{2} \frac{logn}{log(r-1)} + 3 \rfloor$. Since $(r-1)^{l_0} \geq n^{(1+\epsilon)/2}$, the product above is $o(n^{-2})$. Since there are at least $\Omega(n^{-1/2})$ of all the graphs generated by the configuration model that are simple r-regular graphs, there are at most $o(n^{-2}/n^{-1/2}) = o(n^{-3/2})$ of probability that $A_{i,G}^{l_0,spd} = A_{j,G}^{l_0,spd}$.

Next, for any pair of n-sized r-regular graphs $G^{(1)}$ and $G^{(2)}$, we can combine these two graphs and generate a single regular graph with $2n$ nodes. Denote this combined graph as G_c and G_c has two disconnected components. It is easy to see that the above proof is still valid on G_c. This means that: suppose we randomly pick a node v_1 from the first component and node v_2 from another component. Then, given $3 \leq r < (2log2n)^{1/2}$ and $K = \lfloor (\frac{1}{2} + \epsilon) \frac{\log 2n}{\log (r-1)} \rfloor$, we have $A_{v_1,G_c}^{K,spd} = A_{v_2,G_c}^{K,spd}$ with probability of $o(n^{-3/2})$. As node v_1 and v_2 are in two disconnected components $G^{(1)}$ and $G^{(2)}$. Therefore, it is easy to see $A_{v_1,G_c}^{K,spd} = A_{v_1,G^{(1)}}^{K,spd}$ and $A_{v_2,G_c}^{K,spd} = A_{v_1,G^{(2)}}^{K,spd}$, which completes the proof.

□

Theorem 1 is easy to prove with the aid of Lemma 1. Basically, we consider a node v_1 in graph $G^{(1)}$ and compare the $A_{v_1,G^{(1)}}^{K,spd}$ with $A_{v_2,G^{(2)}}^{K,spd}$ for all nodes $v_2 \in V^{(2)}$. The probability that $A_{v_2,G^{(2)}}^{K,spd} \neq A_{v_1,G^{(1)}}^{K,spd}$ for all possible v_2 is $1 - o(n^{-3/2}n) = 1 - o(n^{-1/2})$. Therefore, with an injective readout function, we can guarantee that 1 layer K-hop message passing GNN can generate different embedding for two graphs.

C.2 Simulation experiments to verify Theorem 1

In this section, we conduct simulation experiments to verify both Lemma 1 and Theorem 1. The results are shown in Figure 4. We randomly generate 100 n-sized 3 regular graphs with n range from 20 to 1280. Then, we apply a 1-layer untrained K-GIN model on these graphs (1 as node feature) with K range from 1 to 6. On the left side, we compare the final node representations for all nodes output by K-GIN, If the difference between two node representations $||h_v - h_u||_2$ is less than machine accuracy, they are regarded as indistinguishable. The colors of the scatter plot indicate the portion of two nodes that are not distinguishable by K-GIN. The darker, the more indistinguishable

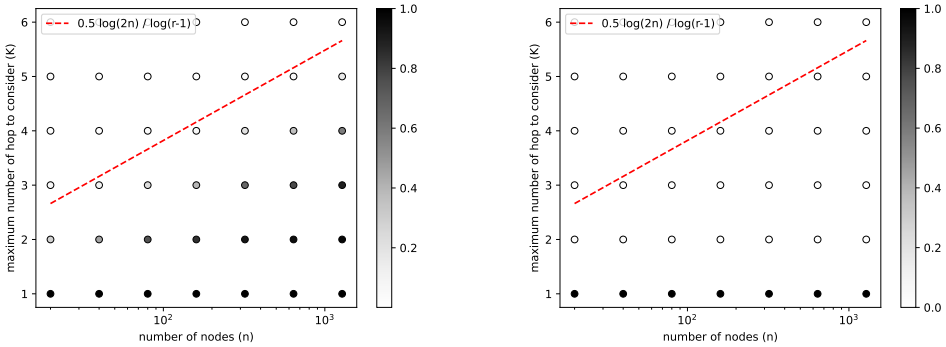

Figure 4: Simulation results. The left side is the node-level result to verify Lemma 1. The right side is the graph-level result to verify Theorem 1.

node pairs. We can see that the result matches almost perfectly with Lemma 1, where K is larger than $\frac{1}{2}\frac{\log 2n}{\log(r-1)}$, almost all nodes are distinguishable by K-GIN. On the right side, we compare the final graph representation output by K-GIN. We can see even with $K = 2$, almost all graphs are distinguishable. This is because as long as there exists one single node from one graph that has a different representation from all nodes in another graph, 1 layer K-hop message passing with an injective readout function can distinguish two graphs.

D General K-hop color refinement and discussion on existing K-hop message passing GNNs

In this section, we introduce a general K-hop color refinement algorithm and use this algorithm to characterize the expressive power of existing K-hop methods further.

D.1 General K-hop color refinement algorithm

It is well-known that 1-WL test updates the label of each node in the graph by color refinement algorithm, which iteratively aggregates the label of its neighbors. Here, we extend it and define a more general color refinement algorithm. First, we denote $[n] = \{0, 1, ..., n\}$ and introduce *refinement configuration*.

Definition 6. *Given L and K, the refinement configuration $\mathbf{C}^{L,K} = (C^{0,K}, C^{1,K}, ..., C^{L,K})$, where $C^{l,K} = (C_0^{l,K}, C_1^{l,K}, ..., C_K^{l,K})$ and $C_k^{l,K} \subseteq [l]$ for any $l \in [L]$ and $k \in [K]$.*

Briefly speaking, refinement configuration defines how the color refinement algorithm aggregates information from K-hop neighbors at each iteration l given the maximum iteration of L and the maximum number of hop K. Given the refinement configuration $\mathbf{C}^{L,K}$, we define general color refinement algorithm $\mathbf{CR}(\mathbf{C}^{L,K})$:

$$\begin{aligned} R_{v,G}^0 &= label(v), \\ R_{v,G}^{l+1} &= HASH((\{\!\{R_{u,G}^s | u \in Q_{v,G}^{k,t}\}\!\} | s \in C_k^{l,k}\}\!\}| \text{ for all } k \in [K])). \end{aligned} \tag{14}$$

Equation (14) define the general color refinement algorithm. Then, given different refinement configurations, we can end up with different procedures for the algorithm. Specifically, we define the following refinement configurations:

Definition 7. *The 1-WL refinement configuration is defined as $\mathbf{C}_{\text{1-WL}}^{L,K} = (C_{\text{1-WL}}^{0,K}, C_{\text{1-WL}}^{1,K}, ..., C_{\text{1-WL}}^{L,K})$, where $C_{\text{1-WL},k}^{l,K} = \{l\}$ for $k = 0, 1$, and $C_{\text{1-WL},k}^{l,K} = \emptyset$ for others.*

Definition 8. *The K-hop refinement configuration is defined as $\mathbf{C}_{\text{K-hop}}^{L,K} = (C_{\text{K-hop}}^{0,K}, C_{\text{K-hop}}^{1,K}, ..., C_{\text{K-hop}}^{L,K})$, where $C_{\text{K-hop},k}^{l,K} = \{l\}$ for all $k \in [K]$.*

Definition 9. *The GINE+ refinement configuration is defined as $\mathbf{C}_{\text{GINE+}}^{L,K} = (C_{\text{GINE+}}^{0,K}, C_{\text{GINE+}}^{1,K}, ..., C_{\text{GINE+}}^{L,K})$, where $C_{\text{GINE+},k}^{l,K} = \{l\}$ for $k = 0$, and $C_{\text{GINE+},k}^{l,K} = \{l - k + 1\}$ for $k = 1, 2, ..., l + 1$, $C_{\text{GINE+},k}^{l,K} = \emptyset$ if $k > l + 1$.*

It is easy to see that $\mathbf{CR}(\mathbf{C}^{L,K}_{\text{1-WL}})$ is exactly the same as the color refinement algorithm in 1-WL test. Next, we can analyze the expressive power of the general color refinement algorithm given different refinement configurations. We say $\mathbf{C}^{L,K}_1 \succeq \mathbf{C}^{L,K}_2$ if $\mathbf{CR}(\mathbf{C}^{L,K}_1)$ is at least equally powerful as $\mathbf{CR}(\mathbf{C}^{L,K}_2)$ in terms of expressive power. Then, we have the following properties for the general refinement algorithm.

Property 1. $\mathbf{C}^{L+1,K} \succeq \mathbf{C}^{L,K}$.

Property 2. *If $C^{l,K}_{1,k} \subseteq C^{l,K}_{2,k}$ for any $k \in [K]$ and $l \in [L]$, then $\mathbf{C}^{L,K}_1 \succeq \mathbf{C}^{L,K}_2$.*

These two properties are easy to validate, as given the injective HASH function, an algorithm with more information is always at least equally powerful as an algorithm with less information. Moreover, we have the following proposition.

Proposition 4. *If for any $l \in [L]$ and any $k \in [K]$, $\mathbf{C}^{l,K}_1$ and $\mathbf{C}^{l,K}_2$ satisfy $i \in C^{l,K}_{1,k}$ and $j \in C^{l,K}_{2,k}$ if and only if $i \geq j$, then $\mathbf{C}^{L,K}_1 \succeq \mathbf{C}^{L,K}_2$.*

Proof. Let $R^{l,i}_{v,G}$ denote the color refinement result after iteration l for node v in graph $G = (V, E)$ using refinement configuration $\mathbf{C}^{L,K}_i$.

1. At iteration 1, if a node aggregates its neighbor's label from some hop, it can only aggregate the initial label of nodes, or namely $R^{0,i}_{v,G}$. Then, if $C^{1,K}_{1,k} = \{\!\{0\}\!\}$, $C^{1,K}_{2,k}$ can be either $\{\!\{0\}\!\}$ or $\emptyset$. If $C^{1,K}_{1,k} = \emptyset$, then $C^{1,K}_{2,k} = \emptyset$ It is trivial to see that if $R^{1,1}_{v_1,G} = R^{1,1}_{v_2,G}$ for any pair of nodes $v_1, v_2 \in G$, then $R^{1,2}_{v_1,G} = R^{1,2}_{v_2,G}$, which means $\mathbf{C}^{0,K}_1 \succeq \mathbf{C}^{0,K}_2$ holds.
2. At iteration l Assume $\mathbf{C}^{l-1,K}_1 \succeq \mathbf{C}^{l-1,K}_2$ holds.
3. At iteration $l+1$, the condition in the proposition means color refinement algorithm with $C^{l,K}_2$ can only aggregate results from earlier iteration than $C^{l,K}_1$ at any hops. Meanwhile, as $\mathbf{C}^{l-1,K}_1 \succeq \mathbf{C}^{l-1,K}_2$ holds, as long as $R^{l,1}_{v_1,G} = R^{l,1}_{v_2,G}$, we have $R^{t,i}_{v_1,G} = R^{t,i}_{v_2,G}$ holds for any $t \in [l]$ and $i = 1, 2$ given the injectiveness of HASH function. This means that if $R^{l+1,1}_{v_1,G} = R^{l+1,1}_{v_2,G}$, then $R^{l+1,2}_{v_1,G} = R^{l+1,2}_{v_2,G}$. Therefore, $\mathbf{C}^{l,K}_1 \succeq \mathbf{C}^{l,K}_2$ also holds. this completes the proof.

□

Given two properties and Proposition 4, we have the following results.

Theorem 3. $\mathbf{C}^{L,K}_{\textit{K-hop}} \succeq \mathbf{C}^{L,K}_{\textit{GINE+}} \succeq \mathbf{C}^{L,K}_{\textit{1-WL}} \succeq \mathbf{C}^{0,K}_{\textit{GINE+}} \succeq \mathbf{C}^{0,K}_{\textit{1-WL}}$.

Proof. Using the Property 1 and Property 2 of general color refinement algorithm, it is easy to prove that $\mathbf{C}^{L,K}_{\text{K-hop}} \succeq \mathbf{C}^{L,K}_{\text{1-WL}} \succeq \mathbf{C}^{0,K}_{\text{GINE+}} \succeq \mathbf{C}^{0,K}_{\text{1-WL}}$ and $\mathbf{C}^{L,K}_{\text{GINE+}} \succeq \mathbf{C}^{L,K}_{\text{1-WL}}$. The only thing left is the comparison between $\mathbf{C}^{L,K}_{\text{K-hop}}$ and $\mathbf{C}^{L,K}_{\text{GINE+}}$. For any $l \in [L]$, $C^{l,K}_{\text{K-hop},k} = \{\!\{l\}\!\}$ for all $k \in [K]$. Instead, $C^{l,K}_{\text{GINE+},k} = \{\!\{l-k+1\}\!\}$ for $k = 1, 2, ..., l+1$ and $C^{l,K}_{\text{GINE+},0} = \{\!\{l\}\!\}$. Then it is easy to see that $C^{l,K}_{\text{K-hop},k}$ and $C^{l,K}_{\text{GINE+},k}$ satisfy the condition of Proposition 4 and thus $\mathbf{C}^{L,K}_{\text{K-hop}} \succeq \mathbf{C}^{L,K}_{\text{GINE+}}$ holds. □

Theorem 3 provide a general comparison between different color refinement configurations. Based on Theorem 3, we can actually extend the Proposition 1 in the main paper as

Corollary 1. *Suppose $K \geq L$, L layer K-hop message passing GNNs defined in Equation (3) with the shortest path distance kernel is at least equally powerful as L layer GINE+ [15]. L layer GINE+ is strictly more powerful than L layer 1-hop message passing GNNs.*

The above Corollary is trivial to prove as corresponding models with injective message and update functions have at most the same expressive power as general color refinement algorithm with corresponding refinement configurations and permutation invariant readout function. However, one remaining question given the general color refinement algorithm is the comparison of the expressive power between K layer 1-hop GNNs and 1 layer K-hop GNNs. Here we show that:

Proposition 5. *Assume we use the shortest path distance kernel for K-hop message passing GNNs. There exists pair of graphs that can be distinguished by 1 layer K-hop message passing GNNs but not K layer 1-hop message passing GNNs and vice versa.*

To prove Proposition 5, we provide two examples. The first example is exactly example 2 in Figure 1. we know that a 1-hop GNN with 2 layers cannot distinguish two graphs as they are all regular graphs. Instead, a 1-layer 2-hop GNN is able to achieve that. From another direction, we show two graphs in Figure 5. We know that 1 layer 2-hop message passing GNN is equivalent to inject node configuration up to 2-hop into nodes. As we show in Figure 5, two graphs have the same node configuration set, which means an injective readout function will produce the same representation. Therefore, two graphs cannot be distinguished by 1 layer 2-hop message passing GNNs. Instead, it is easy to validate that these two graphs can be distinguished by 2 layer 1-hop message passing GNNs.

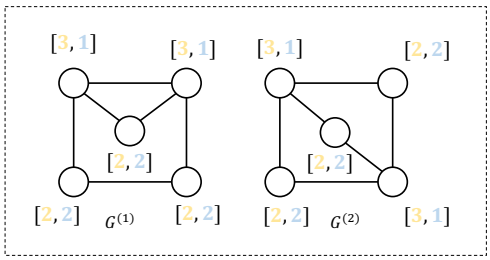

Figure 5: A pair of non-isomorphic graphs that can be distinguished by 2-layer 1-hop GNNs but not 1-layer 2-hop GNNs.

D.2 Discussion on existing K-hop models

GINE+ [15]: GINE+ tries to increase the representation power of graph convolution by increasing the kernel size of convolution. However, in l layer of GINE+, it only aggregates information from the neighbor of hop $1, 2, ..., l$, which means that after L layer of convolution, the GINE+ still has a receptive field of size L. As we discussed in the previous section, L layer K-hop message passing GNNs with the shortest path distance kernel and $K \geq L$ is at least equally powerful as L layer GINE+.

Graphormer [16]: Graphormer introduce a new way to apply transformer architecture [56] on graph data. In each layer of Graphormer, the shortest path distance is used as spatial encoding to adjust the attention score between each node pair. Although the Graphormer does not apply the K-hop message passing directly, the attention mechanism (each node can see all the nodes) with the shortest path distance feature implicitly encodes a rooted subtree similar to the K-hop message passing with the shortest path distance kernel. To see it clearly, suppose we have K-hop message passing with $K = \infty$ and graphs only have one connected component. It will aggregate all the nodes at each layer, which is similar to Graphormer. Meanwhile, the injective message and update function implicitly encode the shortest path distance to each node in aggregation. Then it is trivial to see that Graphormer is actually a special case of K-hop message passing with the shortest path distance kernel.

Spectral GNNs: spectral-based GNNs serve as an important type of graph neural network and gain lots of interest in recent years. Here we only consider one layer as spectral-based GNNs usually only use 1 layer. the general spectral GNNs can be written as:

$$Z = \phi\left(\sum_{k=0}^{K} \alpha_k \rho(\hat{L}^k)\varphi(x)\right), \tag{15}$$

where ϕ and φ are typically multi-layer perceptrons (MLPs), $\hat{L}$ is normalized laplacian matrix, α_k is weight for each spectral basis, and ρ is an element-wise function of matrix. In normal spectral GNNs, ρ is always an identity mapping function. We can see spectral GNNs have a close relationship to K-hop message passing with graph diffusion kernel as $\hat{L}^k$ actually compute the $Q_{v,G}^{k,gd}$ for each node by only keeping element in the matrix that is larger than 0. As K in Equation (15) can be greater than 1, it looks like spectral GNNs fit Proposition 1 as well. However, according to Proposition 4.3 in [21], all such spectral-based methods have expressive power no more than 1-WL test. This seems like a discrepancy. The reason lies in the ρ function in Equation (15). K-hop message passing with graph diffusion kernel can be regarded as using a non-linear ρ function shown in the following:

$$\rho(x) = \begin{cases} 1 & x > 0 \\ 0 & others \end{cases} \tag{16}$$

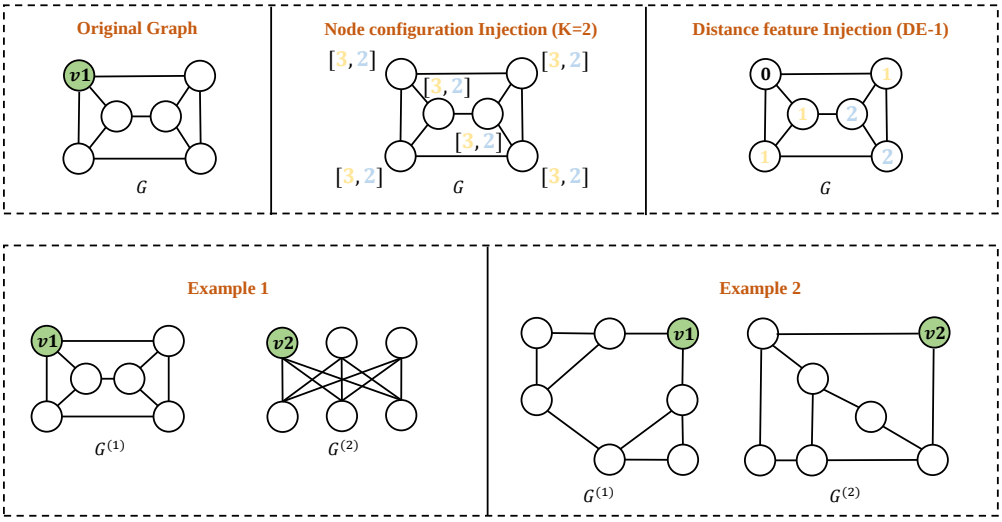

Figure 6: The upper part: graph with node configuration as the injected label and DE-1 as the injected label on the center node. The bottom part: two pairs of non-isomorphic graphs where the node pair in example 1 can be distinguished by DE-1 and the node pair in example 2 can be distinguished by K-hop message passing.

This non-linear function is the key difference between the normal spectral GNNs and K-hop message passing and it endows normal spectral GNNs with extra expressive power than 1-WL test.

MixHop [11], GPR-GNN [14], and MAGNA [13]: All three methods extend the scope of 1-hop message passing by consider multiple graph diffusion step simultaneously. However, all these methods use normal spectral GNNs as the base encoder and thus they obey Proposition 4.3 in [21] instead of Proposition 1. It can be seen as a "weak" version of K-hop message passing with graph diffusion kernel.

E Comparison between K-hop message passing and Distance Encoding

In this section, we further discuss the connection and difference between K-hop message passing and Distance Encoding [22]. Here we assume that the kernel of K-hop message passing is the shortest path distance kernel and Distance Encoding with the shortest path distance as distance feature.

To simplify the discussion, suppose there are two graphs $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$. We pick two nodes v_1 and v_2 from each graph respectively and learn the representation for these two nodes. First, let us consider what 1 layer K-hop message passing and DE-1 without message passing. As we stated in the main paper, 1 layer of K-hop message passing actually injects each node with the label of node configuration. Instead, DE-1 injects each node with the distance to the center node (v_1 and v_2 here). Then we can see there is a clear difference between the two methods even if they all implicitly or explicitly use the distance feature as shown in the upper part of Figure 6. Next, it is easy to see that applying $L+1$ layer K-hop message passing is equivalent to applying L layer K-hop message passing on a graph with node configuration as the initial label. Applying L layer DE-1 is equivalent to applying L layer 1-hop message passing on a graph with the distance to the center node as the initial label. In the following, we show that the two methods cannot cover each other in terms of distinguishing different nodes:

Proposition 6. *For two non-isomorphic graphs $G^{(1)} = (V^{(1)}, E^{(1)})$ and $G^{(2)} = (V^{(2)}, E^{(2)})$, we pick two nodes $v1$ and v_2 from each graph respectively. Then there exist pairs of graphs that nodes v_1 and v_2 can be distinguished by K-hop message passing with the shortest path distance kernel but not DE-1, and vice versa.*

To prove the Proposition 6, we provide two examples in the lower part of Figure 6. Example 1 in the figure is exactly the same pair of regular graphs in example 1 of Figure 1. As discussed before, K-hop message passing with the shortest path distance kernel cannot distinguish node v_1 and v_2. However, after injecting the distance feature to the center node in two graphs and performing 2 layers of message passing, DE-1 is able to assign different representations for two nodes. In example 2, DE-1 cannot distinguish nodes v_1 and v_2 in the two graphs, even if they are not regular graphs. Instead,

using K-hop message passing with only $K = 2$, the two nodes will get different representations. We omit the detailed procedure here as it is easy to validate. Using these two examples, we have shown that K-hop message passing and DE-1 are not equivalent to each other even if they all use the shortest path distance feature. The root reason is that although DE-1 uses the distance feature, it still aggregates information only from 1-hop neighbors in each iteration, while K-hop message passing directly aggregates information from all K-hop neighbors. That is, their ways of using distance information are different. However, we also notice that in example 2, if we label the graph with DE-1 on another pair of nodes, two nodes can be distinguished, which means DE-1 is able to distinguish these two graphs. **However, to achieve that, DE-1 need to label the graph n times and run the message passing on all n labeled graphs. Instead, K-hop message passing only needs to run the message passing once, which is both space and time efficient.**

Besides DE-1, Li et al. [22] also proposed the DEA-GNN which is at least no less powerful than DE-1. The DEA-GNN extends DE-1 by simultaneously aggregating all other nodes in the graph but the message is encoded with the distance to the center node. This can be seen as exactly performing K-hop message passing. In other words, DEA-GNN is the combination of K-hop message passing and DE-1. Therefore it is easy to see:

Proposition 7. *The DEA-GNN [22] is at least equally powerful as K-hop message passing with the shortest path distance kernel.*

F Proof of Theorem 2

In this section, we prove Theorem 2. We first restate Theorem 2: The expressive power of a proper K-hop message passing GNN of any kernel is bounded by 3-WL test. Our proof is inspired by the recent results in SUN [23], which bound all subgraph-based GNN with the 3-WL test by proving that all such methods can be implemented by 3-IGN. Here we prove that K-hop message passing can also be implemented by 3-IGN. We will not discuss the detail of 3-IGN and all its operations. Instead, we directly follow all the definitions and notations and refer readers to Appendix B of [23] for more details.

K-hop neighbor extraction: To implement the K-hop message passing, we first implement the extraction of K-hop neighbors. The key insight is that we can use the l-th channel in X_{iij} to store whether node j is the neighbor of node i at l-th hop, which is similar to the extraction of ego-network. Here we suppose $d \geq K$. Same as all node selection policies, we first lift the adjacency A to a three-way tensor $\mathcal{Y} \in \mathbb{R}^{n^3 \times d}$ using broadcasting operations:

$$X^{(0)}_{iii} = \boldsymbol{\beta}_{i,i,i} A_{ii} \tag{17}$$

$$X^{(0)}_{jii} = \boldsymbol{\beta}_{*,i,i} A_{ii} \tag{18}$$

$$X^{(0)}_{iij} = \boldsymbol{\beta}_{i,i,j} A_{ij} \tag{19}$$

$$X^{(0)}_{iji} = \boldsymbol{\beta}_{i,j,i} A_{ij} \tag{20}$$

$$X^{(0)}_{kij} = \boldsymbol{\beta}_{*,i,j} A_{ij} \tag{21}$$

Now, $X^{(0),1}_{iij}$ store the 1-hop neighbors of node i. Next, l-th hop neighbor of node i is computed and stored in $X^{:l}_{iij}$. To get neighbor of l-th hop for $l = 2, 3, ..., K$, we first copy it into $d + 1$ channels:

$$X^{(1)}_{ijj} = \boldsymbol{\kappa}^{:d}_{:d/(d+1)} X^{(0)}_{ijj} + \boldsymbol{\kappa}^{l:l}_{d+1:d+1} \boldsymbol{\beta}_{i,j,j} X^{(0)}_{iij} \tag{22}$$

The $d + 1$-th channel is used to compute high-order neighbors. Next, we extract all K hop neighbors by iteratively following steps $K - 1$ times and describe the generic l-th step. We first broadcast the current reachability pattern into X_{ijk}, writing it into the l-th channel:

$$X^{(l,1)}_{ijk} = X^{(l-1)}_{ijk} + \boldsymbol{\kappa}^{d+1:d+1}_{l:l} \boldsymbol{\beta}_{i,*,j} X^{(l-1)}_{ijj} \tag{23}$$

Then, a logical AND is performed to get the new reachability. Then write back the results into the l-th channel:

$$X_{ijk}^{(l,2)} = \boldsymbol{\varphi}_{l:l/(d+1)}^{(\wedge)1,l} X_{ijk}^{(l,1)} \tag{24}$$

Next, we get all nodes that can be reached within l hops by performing pooling, clipping, and copying back to $d+1$ channel:

$$X_{ijj}^{(l,3)} = \boldsymbol{\kappa}_{:d/(d+1)}^{:d} X_{ijj}^{(l,2)} + \boldsymbol{\kappa}_{d+1:}^{l:l} \boldsymbol{\beta}_{i,j,j} \boldsymbol{\pi}_{i,j} X_{ijk}^{(l,2)} \tag{25}$$

$$X_{ijj}^{(l,4)} = \begin{bmatrix} \boldsymbol{\kappa}_{:d/(d+1)}^{:d} & \boldsymbol{\varphi}_{d+1:d+1}^{(\downarrow)d+1:d+1} \end{bmatrix} X_{ijj}^{(l,3)} \tag{26}$$

Now $X_{ijj}^{(l,4),d+1}$ save all the nodes that can be reached at l-th hop. Finally, we extract the l-th hop neighbor and copy it into l-th channel. For graph diffusion kernel, the current result is itself result for graph diffusion kernel, which means we only need to copy it:

$$X_{iij}^{(l)} = X_{iij}^{(l,4)} + \boldsymbol{\kappa}_{l:l}^{d+1:d+1} \boldsymbol{\beta}_{i,i,j} X_{ijj}^{(l,4)} \tag{27}$$

For the shortest path distance kernel, we need to nullify all the nodes that already existed in the previous hops. To achieve this, we need to first compute if a node exists in the previous hops and then nullify it:

$$X_{iij}^{(l,5)} = X_{iij}^{(l,4)} + \boldsymbol{\kappa}_{l:l}^{d+1:d+1} \boldsymbol{\beta}_{i,i,j} X_{ijj}^{(l,4)} \tag{28}$$

$$X_{iij}^{(l,6)} = \sum_{i=1}^{l-1} \boldsymbol{\kappa}_{d+1:d+1}^{i:i} X_{iij}^{(l,5)} \tag{29}$$

$$X_{iij}^{(l)} = X_{iij}^{(l,4)} + \boldsymbol{\varphi}_{l:l}^{(\wedge)l,d+1} \boldsymbol{\varphi}_{d+1}^{(!)d+1} X_{iij}^{(l,6)} \tag{30}$$

Where $\boldsymbol{\varphi}_{b}^{(!)a}$ is logical not function that output 0 if input is not 0 and vice versa for channel a and write result into channel b. Here we omit the detailed implementation as it is easy to implement using ReLu function. Finally, we bring all other orbits tensors to the original dimensions:

$$X_{iii} = \boldsymbol{\kappa}_{:d}^{:d} X_{iii}^{(l)} \tag{31}$$

$$X_{ijj} = \boldsymbol{\kappa}_{:d}^{:d} X_{ijj}^{(l)} \tag{32}$$

$$X_{iij} = \boldsymbol{\kappa}_{:d}^{:d} X_{iij}^{(l)} \tag{33}$$

$$X_{iji} = \boldsymbol{\kappa}_{:d}^{:d} X_{iji}^{(l)} \tag{34}$$

$$X_{ijk} = \boldsymbol{\kappa}_{:1/(d)}^{:1} X_{ijk}^{(l)} \tag{35}$$

Now, we have successfully implemented K-hop neighbor extraction algorithm using 3-IGN.

K-hop message passing: To implement the message and update function for each layer, we use the same base encoder as [18]. Other types of base encoders and combine functions can be implemented in a similar way. We follow the same procedure of implementing the base encoder of DSS-GNN as stated in [23]. Note here for each hop the procedure is similar therefore we state the generic l-th hop. The key insight is that, in K-hop message passing, we are actually not working on each subgraph but the original graph, which means all the operations can be implemented in the orbit tensor X_{iii} and X_{iij}.

Message broadcasting: This procedure is similar to the base encoder of DSS-GNN but here we only need to broadcast X_{iij}. However, since we need to perform message passing for K times, we need to broadcast it K times:

$$X_{iij}^{(t,1)} = \boldsymbol{\kappa}_{:d/((K+1)d)}^{:d} X_{iij}^{(t)} + \sum_{i=1}^{K} \boldsymbol{\kappa}_{id+1:(i+1)d/(K+1)d}^{:} \boldsymbol{\beta}_{iij} X_{ijj}^{(t)} \tag{36}$$

Message sparsification and aggregation: Similar to DSS-GNN, here we need to nullify the message from nodes that are not l-th hop neighbors. Here we define the following function:

$$f_{iij}^{\odot}\left(X_{aab}^{(t,1)}\right)_l = \begin{cases} \mathbf{0}_d & \text{if } X_{aab}^{(t,1),l} = 0 \\ X_{aab}^{(t,1),l(d+1):(l+1)d} & \text{otherwise} \end{cases} \tag{37}$$

The function is used to nullify the message for l-th hop. It is easy to validate the existence of such function following the same procedure in [23] therefore we omit the detail. Next, the message sparsification can be implemented by:

$$X_{iij}^{(t,3)} = \left[\begin{array}{cccc} \boldsymbol{\kappa}_{:d}^{:d} & \boldsymbol{\varphi}_1^{(\odot_{ijj}):} & \dots & \boldsymbol{\varphi}_K^{(\odot_{ijj}):} \end{array} \right] X_{iij}^{(t,2)} \tag{38}$$

Then, the message function for K-hop message passing is:

$$X_{iii}^{(t,4)} = \boldsymbol{\kappa}_{:d}^{:d} X_{iii}^{(t,3)} + \boldsymbol{\kappa}_{d+1:}^{d+1:} \boldsymbol{\beta}_{iii} \boldsymbol{\pi}_i X_{ijj}^{(t,3)} \tag{39}$$

Update Then, the update function is implemented using linear transformation. In K-hop message passing, each hop needs an independent parameter set. Here in order to operate on constructed tensor, we define $W_t^l = \left[W_{t,1}^l || W_{t,2}^l || ... || W_{t,K}^l\right]$, where $W_{t,i}^l = \mathbf{0}_{\boldsymbol{d}}$ if $i \neq l$. Then, the update function is:

$$X_{iii}^{(t,5)} = \sum_{l=1}^{K} \sigma\left(W_t^l X_{iii}^{(t,4)}\right) \tag{40}$$

Combine: Finally, we implement the sum combine function. The combine function is can be implemented by a simple MLP:

$$X_{iii}^{(t+1)} = \boldsymbol{\varphi}_d^{\boldsymbol{f}}(X_{iii}^{(t,4)}) \tag{41}$$

$$X_{jii}^{(t+1)} = \boldsymbol{\beta}_{jii} X_{iii}^{t+1} \tag{42}$$

Now, we successfully implement both K-hop neighbor extraction and K-hop message passing layer. Note that other parts like the readout function can be easily implemented and we omit the detail. This means the expressive power of K-hop message passing with either graph diffusion kernel or shortest path distance kernel is bounded by 3-IGN. Based on the Theorem 2 in [57], we conclude the Theorem 2. Further, it is intuitive that all methods that use the node pair distance feature are also bounded by 3-WL as this feature can be computed by 3-IGN. We leave the formal proof in our future work.

G Proof and discussion of Proposition 2

Proposition 2 seems intuitive and easy to prove. However, it also unclear how powerful are KP-GNN. So here we still give a more detailed discussion on it. we first give the definition of *distance regular graph*.

Definition 10. *A distance regular graph is a regular graph such that for any two nodes v and u, the number of nodes with the distance of i from v and distance of j from u depends only on i, j and the distance between v and u.*

Furthermore, we only consider the connected distance regular graphs with no multi-edge or self-loop. Such graphs can be characterized by *intersection array*.

Definition 11. *The intersection array of a distance regular graph with diameter d is an array of integers $(b_0, b_1, ..., b_{d-1}; c_1, c_2, ..., c_d)$, where for all $1 \leq j \leq d$, b_j gives the number of neighbors of u with distance $j+1$ from v and c_j gives the number of neighbors of u with distance $j-1$ from v for any pair of (v, u) in graph with distance j.*

Given the definition of distance regular graph and intersection array, we can propose the first lemma.

Lemma 2. *Given a distance regular graph G with intersection array $(b_0, b_1, ..., b_{d-1}; c_1, c_2, ..., c_d)$. Pick a node v from G, the peripheral subgraph $G^{j,spd}_{v,G}$ is a n-sized r-regular graph with $n = |Q^{j,spd}_{v,G}|$ and $r = b_0 - b_j - c_j$*

Proof. Given a distance regular graph G with intersection array $(b_0, b_1, ..., b_{d-1}; c_1, c_2, ..., c_d)$, from the definition of intersection array, for node v in G, $Q^{j,spd}_{v,G}$ is the node set that have distance j from node v. Then, b_j is the number of neighbors for each node in $Q^{j,spd}_{v,G}$ that have distance $j+1$ to node v. It is easy to see that these neighbors must belong to $Q^{j+1,spd}_{v,G}$, which means that b_j is also the number of edge for a node in $Q^{j,spd}_{v,G}$ that connect to nodes in $Q^{j+1,spd}_{v,G}$. Similarly, c_j is the number of edge for a node in $Q^{j,spd}_{v,G}$ that connect to nodes in $Q^{j-1,spd}_{v,G}$. For node $u \in Q^{j,spd}_{v,G}$, we know that the edges of u must connect to either $Q^{j+1,spd}_{v,G}$, $Q^{j,spd}_{v,G}$, or $Q^{j-1,spd}_{v,G}$. Since the degree of node u is b_0, then the number of edge that connect node u to nodes in $Q^{j,spd}_{v,G}$ is $b_0 - b_j - c_j$. The above statement holds for each $u \in Q^{j,spd}_{v,G}$, which means all nodes $u \in Q^{j,spd}_{v,G}$ have same node degree. Meanwhile, the node set of peripheral subgraph $G^{j,spd}_{v,G}$ is exactly $Q^{j,spd}_{v,G}$. Combine two statements, we can conclude that $G^{j,spd}_{v,G}$ is a n-sized r-regular graph with $n = |Q^{j,spd}_{v,G}|$ and $r = b_0 - b_j - c_j$. □

Given the Lemma 2, we know that the peripheral subgraph of a node in any distance regular graph is itself a regular graph. Next, given two non-isomorphic distance regular graphs $G^{(1)}$ and $G^{(2)}$ with the same intersection array, there are total d pairs of regular peripheral subgraphs for any pair of nodes v_1 and v_2. If the KP-GNN can distinguish two regular graphs at some hop $j \leq d$, then the KP-GNN can distinguish v_1 and v_2 in graph $G^{(1)}$ and $G^{(2)}$. Meanwhile, it is easy to see that each node in a distance regular graph has the same local structure. Therefore, as long as v_1 can be distinguished from v_2, KP-GNN can distinguish two graphs.

Now given the implementation of KP-GNN in Equation 5, if either the peripheral set $E(Q^{k,t}_{v_1,G^{(1)}})$ is different from $E(Q^{k,t}_{v_2,G^{(2)}})$ or $C^{k'}_{j,G^{(1)}}$ is different from $C^{k'}_{j,G^{(2)}}$. However, as peripheral subgraph is itself regular graphs, $E(Q^{k,t}_{v_1,G^{(1)}})$ must equal to $E(Q^{k,t}_{v_2,G^{(2)}})$. Therefore, KP-GNN can only distinguish two peripheral subgraph by $C^{k'}_{j,G^{(1)}}$ and $C^{k'}_{j,G^{(2)}}$.

It looks like we can use the result from Theorem 1 to prove that Equation 5 can distinguish almost all distance regular graphs. However, although the peripheral subgraphs of v_1 and v_2 are regular graphs with the same size and r, they are not randomly generated by the configuration model and do not satisfy Theorem 1. We leave the quantitative expressive power analysis of KP-GNN in our future works.

H Time, space complexity and limitation of K-hop message passing and KP-GNN

H.1 Time and space complexity

In this section, we analyze the time and space complexity. To simplify the analysis, we first consider the shortest path distance kernel, as the graph diffusion kernel has both time and space complexity no less than the shortest path distance kernel. Denote graph G with n node and m edges.

Space complexity: For both K-hop message passing and KP-GNN, as we only need to maintain one representation for each node, the space complexity is $O(n)$ like in vanilla 1-hop message passing.

Time complexity: First, we analyze the time complexity of K-hop message passing GNNs. For each node, suppose in the worst case we extract neighbors from all nodes from all hops, the number of neighbors we need to aggregate from all hops is n (all nodes in the graph). Then, n nodes need at most $O(n^2)$ time complexity. Next, we analyze the time complexity of KP-GNN. The additional time complexity comes from counting the peripheral edges and k'-configuration. Since the counting can be done in a preprocessing step and reused at each message passing iteration, it will be amortized to zero finally. So the practical time complexity is still $O(n^2)$.

H.2 Discussion on the complexity

From the above analysis, we know that K-hop GNNs and KP-GNN only need $O(n)$ space complexity, which is equal to vanilla message passing GNNs and much less than subgraph-based GNNs which require $O(n^2)$ space (due to maintaining a different representation for each node when appearing in different subgraphs). Thus, KP-GNN has a much better space complexity than subgraph-based GNNs, PPGN [24] (also $O(n^2)$) and 3-WL-GNNs ($O(n^3)$).

The worst-case time complexity of K-hop GNNs and KP-GNN is much higher than that of normal GNNs due to aggregating information from more than 1-hop nodes in each iteration. However, we also note that the larger receptive field could reduce the number of message passing iterations because 1-layer of K-hop message passing can cover the receptive field of K-layer 1-hop message passing. Furthermore, K-hop GNN and KP-GNN have better time complexity $O(n^2)$ than that of subgraph-based GNNs (which require $O(nm)$ for doing 1-hop message passing $O(m)$ in all n subgraphs). For sparse graphs, we can already save a factor of $d_{\text{avg}} = m/n$ complexity. For dense graphs, the worst-case time complexity of subgraph-based GNNs becomes $O(n \cdot n^2) = O(n^3)$, and our time complexity advantage becomes even more significant.

H.3 Limitations

We discuss the limitation of the proposed KP-GNN from two aspects.

Stability: Using K-hop instead of 1-hop can make the receptive field of a node increase with respect to K. For example, to compute the representation of a node with L layer K-hop, GNN, the node will get information from all LK-hop neighbors. The increased receptive field can hurt learning, as mentioned in GINE+ [15]. It is an intrinsic limitation that exists in all K-hop GNNs. GINE+ [15] proposed a new way to fix the receptive field as L by only considering $L - i$ layer representation of neighbor in i hop during the aggregation. We also apply this approach and it helps mitigate the issue and shows great practical performance gain.

Time complexity: As we show above, we need $O(n^2)$ time complexity for KP-GNN, which is much higher than $O(m)$ of MPNN. However, this limitation exists for all subgraph-based methods like NGNN [26], GNN-AK [27], and ESAN [28] as they all require $O(nm)$ time complexity [23]. This is the sacrifice for better expressive power.

I Implementation detail of KP-GNN

In this section, we discuss the implementation detail of KP-GNN.

Combine function: 1-hop message passing GNNs do not have $COMBINE^l$ function. Here we introduce two different $COMBINE^l$ functions. The first one is the attention [58] based combination mechanism, which automatically learns the importance of representation for each node at each hop. The second one uses the well-known geometric distribution [13]. The weight of hop i is computed based on $\theta_i = \alpha(1-\alpha)^i$, where $\alpha \in (0, 1]$. The final representation is calculated by weighted summation of the representation of all the hops. In our implementation, the α is learnable and different for each feature channel and each hop.

Peripheral subgraph information: In the current implementation, KP-GNN will compute two pieces of information. The first one is the peripheral edge set $E(Q^{k,t}_{v,G})$. In our implementation, we compute the number of edges for each distinct edge type. The second one is node configuration

and peripheral edge set for all nodes in the peripheral subgraph $C^{k'}$. this is equivalent to running 1 layer of KP-GNN on each peripheral subgraph. Specifically, for node configuration, we compute the node configuration for each node and sum them up. For the peripheral edge set, we compute the total number of edges across all hops (do not consider edge type here). Finally, these two pieces of information are combined as $C^{k'}$. All these steps are down in the preprocessing stage.

KP-GCN, KP-GIN, and KP-GraphSAGE: We implement KP-GCN, KP-GIN, and KP-GraphSAGE using the message and update function defined in GCN [1], GIN [7], and GraphSAGE [3] respectively. In each hop, independent parameter sets are used and the computation in each hop strictly follows the corresponding model. However, increasing the number of K will also increase the total number of parameters, which is not scalable to K. To avoid this issue, we design the K-hop message passing in the following way. Suppose the total hidden size of the model is H, then the hidden size of each hop is H/K. In this way, the model size is still on the same scale even with large K.

KP-GIN+: In a normal K-hop message passing framework, all K-hop neighbors will be aggregated for each node. It means that, after L layers, the receptive field of GNN is LK. This may cause the unstable of the training as unrelated information may be aggregated. To alleviate this issue, we adapted the idea from GINE+ [15]. Specifically, we implement KP-GIN+, which applies exactly the same architecture as GINE+ except here we add peripheral subgraph information. At layer l, GINE+ only aggregates information from neighbors within l-hop, which makes a L layer GINE+ still have a receptive field of L. Note that in KP-GIN+, we use a shared parameter set for each hop.

Path encoding: To further utilize the information of graph structure on each hop, we introduce the path encoding to KP-GNN. Specifically, we not only count whether two nodes are neighbors at hop k, but also count the number of paths with length k between two nodes. Such information can be obtained with no additional cost as the A^k of a graph G with adjacency A is a path counter with length k. Then the information is added to the $AGG_k^{l,normal}$ function as additional features.

Other implementation: For all GNNs, we apply the Jumping Knowledge method [59] to get the final node representation. The possible methods include sum, mean, concatenation, last, and attention. Batch normalization is used after each layer.

J Experiential setting details

EXP dataset: For both K-GIN and KP-GIN, we use a hidden size of 48. The final node representation is output from the last layer and the pooling method is the summation. In the experiment, we use 10-fold cross-validation. For each fold, we use 8 folds for training, 1 fold for validation, and 1 fold for testing. We select the model with the best validation accuracy and report the mean results across all folds. The training epoch is set to 40. In this experiment, we do not use path encoding for fair comparison. The learning rate is set to 0.001 and we use *ReduceLROnPlateau* learning rate scheduler with a patience of 5 and a reduction factor of 0.5.

SR25 dataset: For both K-GIN and KP-GIN we use a hidden size of 64. The final node representation is output from the last layer and the pooling method is the summation. For SR25 dataset, we directly train the validate the model on the whole dataset and report the best performance across 200 epochs. For each fold, we use 8 folds for training, 1 fold for validation, and 1 fold for testing. We select the model with the best validation accuracy and report the mean results across all folds. The training epoch is set to 200. In this experiment, we do not use path encoding for a fair comparison. The learning rate is set to 0.001.

CSL dataset: For both K-GIN and KP-GIN we use a hidden size of 48. The final node representation is output from the last layer and the pooling method is the summation. In the experiment, we use 5-fold cross-validation.

Graph&Node property dataset: For graph and node property prediction tasks, we train models with independent 4 runs and report the mean results. For both K-GIN+ and KP-GIN+, the hidden size of models is set as 96. The final node representation is the concatenation of each layer. The pooling method is attention for graph property prediction tasks and sum for node property prediction tasks. The learning rate is 0.01 and we use *ReduceLROnPlateau* learning rate scheduler with a patience of 10 and a reduction factor of 0.5. We use the shortest path distance kernel. The maximum number of epochs for each run is 250. For both K-GIN+ and KP-GIN+, we search K from 3 to 6 with/without path encoding and report the best result.

Graph substructure counting dataset: For graph substructure counting tasks, we train models with independent 4 runs and report the mean results. For both K-GIN+ and KP-GIN+, the hidden size of models is set as 96. The final node representation is the concatenation of each layer. The pooling method is the summation. The learning rate is 0.01 and we use *ReduceLROnPlateau* learning rate scheduler with a patience of 10 and a reduction factor of 0.5. We use the shortest path distance kernel. The maximum number of epochs for each run is 250. For both K-GIN+ and KP-GIN+, we search K from 1 to 4 with/without path encoding and report the best result.

TU datasets: For TU datasets, we use 10-fold cross-validation. We report result for both setting in [7] and [36]. For the first setting, we use 9 folds for training and 1 fold for testing in each fold. After training, we average the test accuracy across all the folds. Then a single epoch with the best mean accuracy and the standard deviation is reported. For the second setting, we still use 9 folds for training and 1 fold for testing in each fold but we directly report the mean best test results. For KP-GNN, we implement GCN [1], GraphSAGE [3] and GIN [7] version. we search (1)the number of layer $\{2, 3, 4\}$, (2)the number of hop $\{2, 3, 4\}$, (3) the kernel of K-hop $\{spd, gd\}$, and (4) the $COMBINE$ function $\{attention, geometric\}$. The hidden size is 33 when $K = 3$ and 32 for the rest of the experiments. The final node representation is the last layer and the pooling method is the summation. The dropout rate is set as 0.5, the number of training epochs for each fold is 350 and the batch size is 32. The initial learning rate is set as $1e - 3$ and decays with a factor of 0.5 after every 50 epochs.

QM9 dataset: For QM9 dataset, we implement KP-GIN+. The hidden size of the model is 128. The number of hops and the number of layers are both 8. The final node representation is the concatenation of each layer and the pooling method is attention. The dropout rate is 0. We use the shortest path distance kernel. The maximum number of the peripheral edge is 6 and the maximum number of the component is 3. We also use additional path encoding in each layer. The learning rate is 0.001 and we use *ReduceLROnPlateau* learning rate scheduler with a patience of 5 and a reduction factor of 0.7. If the learning rate is less than 0.000001, the training is stopped.

ZINC datraset: For ZINC dataset, we run the experiment 4 times independently and report the mean results. For each run, the maximum number of epochs is 500 and the batch size is 64. We implement the KP-GIN+ for the dataset. The hidden size is 104. The final node representation is the concatenation of each layer and the pooling method is the summation. We use the shortest path distance kernel. The number of hops and the number of layers are both 8. The initial learning rate is 0.001 and we use *ReduceLROnPlateau* learning rate scheduler with a patience of 10 and a reduction factor of 0.5. If the learning rate is less than 0.000001, the training is stopped.

K Datasets Description and Statistics

Table 7: Dataset statistics.

Dataset	#Tasks	# Graphs	Ave. # Nodes	Ave. # Edges
EXP	2	1200	44.4	110.2
SR25	15	15	25	300
CSL	10	150	41.0	164.0
Graph&Node property	3	5120/640/1280	19.5	101.1
Substructure counting	4	1500/1000/2500	18.8	62.6
MUTAG	2	188	17.93	19.79
D&D	2	1178	284.32	715.66
PTC-MR	2	344	14.29	14.69
PROTEINS	2	1113	39.06	72.82
IMDB-B	2	1000	19.77	96.53
QM9	12	129433	18.0	18.6
ZINC	1	10000 / 1000 / 1000	23.1	49.8