Graph Representation Learning with Individualization and Refinement

Mohammed Haroon Dupty, Wee Sun Lee

National University of Singapore
{dmharoon, leews}@comp.nus.edu.sg

Abstract

Graph Neural Networks (GNNs) have emerged as prominent models for representation learning on graph structured data. GNNs follow an approach of message passing analogous to 1-dimensional Weisfeiler Lehman (1-WL) test for graph isomorphism and consequently are limited by the distinguishing power of 1-WL. More expressive higher-order GNNs which operate on k-tuples of nodes need increased computational resources in order to process higher-order tensors. Instead of the WL approach, in this work, we follow the classical approach of *Individualization and Refinement* (IR), a technique followed by most practical isomorphism solvers. Individualization refers to artificially distinguishing a node in the graph and refinement is the propagation of this information to other nodes through message passing. We learn to adaptively select nodes to individualize and to aggregate the resulting graphs after refinement to help handle the complexity. Our technique lets us learn richer node embeddings while keeping the computational complexity manageable. Theoretically, we show that our procedure is more expressive than the 1-WL test. Experiments show that our method outperforms prominent 1-WL GNN models as well as competitive higher-order baselines on several benchmark synthetic and real datasets. Furthermore, our method opens new doors for exploring the paradigm of learning on graph structures with individualization and refinement.

Introduction

Graphs are one of the most useful data structure in terms of representing real world data like molecules, social networks and citation networks. Consequently, representation learning (Bengio, Courville, and Vincent 2013; Hamilton, Ying, and Leskovec 2017a,b) on graph structured data has gained prominence in the machine learning community. In recent years, Graph Neural Networks (GNNs) (Kipf and Welling 2016; Defferrard, Bresson, and Vandergheynst 2016; Gilmer et al. 2017; Wu et al. 2020) have become models of choice in learning representations for graph structured data. GNNs operate locally on each node by iteratively aggregating information from its neighbourhood in the form of messages and updating its embedding based on the messages received. This local procedure has been shown to capture some of the

global structure of the graph. However, there are many simple topological properties like detecting triangles that GNNs fail to capture (Chen et al. 2020; Knyazev, Taylor, and Amer 2019).

Theoretically, GNNs have been shown to be no more powerful than 1-dimensional Weisfeiler Lehman (1-WL) test for graph isomorphism (Xu et al. 2018; Morris et al. 2019) in terms of their distinguishing capacity of non-isomorphic graphs. 1-WL is a classical technique of iteratively coloring vertices by injectively hashing the multisets of colors of their adjacent vertices. Although this procedure produces a stable coloring, which can be used to compare graphs, it fails to uniquely color many non-symmetric vertices. This is precisely the reason for the failure of 1-WL and thereby GNNs, to distinguish many simple graph structures (Kiefer et al. 2020; Arvind et al. 2020). Further works on improving the power of GNNs have tried to incorporate higher-order WL generalizations in the form of k-order GNNs (Morris et al. 2019; Maron et al. 2019; Morris, Rattan, and Mutzel 2020), These k-order GNNs jettison the local nature of 1-WL and operate on k-tuples of vertices and hence are computationally demanding. However, this suggests techniques for learning better node representations can be found in classical algorithms used to detect graph isomorphisms.

Another possible approach for learning node features is the *individualization and refinement* (IR) paradigm which provides a useful toolbox for testing graph isomorphism. In fact, all state-of-the-art isomorphism solvers follow this approach and are fairly fast in practice (McKay et al. 1981; Darga et al. 2004; Junttila and Kaski 2011; McKay and Piperno 2014). IR techniques aim to assign a distinct color to each vertex such that the coloring respects permutation invariance and is *canonical*, a unique coloring of its isomorphism class (no two non-isomorphic graphs get the same coloring). This is achieved first by *individualizion*, a process of recoloring a vertex in order to artificially distinguish it from rest of the vertices. Thereafter, a color *refinement* procedure like the 1-WL message passing is applied to propagate this information across the graph. This process of IR is repeated until each vertex gets a unique color. But the coloring is not *canonical* yet. To preserve permutation invariance, whenever a vertex is individualized, we have to individualize, and thereafter refine, all other vertices with the same color as well. As each individualization of a vertex gives a different final discrete coloring,

this generates a search-tree of colorings where each individualization forms a tree-node. The tree generated is *canonical*. The coloring of the graph at each leaf can also be used to label the graph, and the graph with the largest label can be used as a canonical graph. However, the size of search-tree grows exponentially fast and isomorphism-solvers prune the search-tree heavily by detecting symmetries like automorphisms, and by other hand-crafted techniques.

In this paper, we propose to improve representations learnt by GNNs by incorporating the inductive biases suggested by *individualization and refinement* in updating the node embeddings. Unlike isomorphism-solvers, it is not desirable to check for automorphisms to prune the search-tree from a learning perspective. For computational efficiency, we restrict our search to individualizing k nodes in each iteration. In order to restrict the search from exponentially blowing up, we take the approach similar to beam search and reduce the k refined graphs to a single representative graph and repeat the individualization and refinement process. This simple technique lets us learn richer node embeddings while keeping the computational complexity manageable. We validate our approach with experiments on synthetic and real datasets, where we show our model does well on problems where 1-WL GNN models clearly fail. Then we show that our model outperforms other prominent higher-order GNNs by a substantial margin on some real datasets.

Related Work

Over the last few years, there have been considerable number of works on understanding the representative power of GNNs in terms of their distinguishing capacity of non-isomorphic graphs (Xu et al. 2018; Morris et al. 2019; Chen et al. 2019b; Dehmamy, Barabási, and Yu 2019; Srinivasan and Ribeiro 2019; Loukas 2019; Barceló et al. 2019). These works have established that the GNNs are no more expressive than 1-WL kernels in graph classification (Shervashidze et al. 2011). Furthermore, Chen et al. (2020) have shown that many of the commonly occurring substructures cannot be detected by GNNs. One example is counting the number of triangles which can be easily computed from third power of the adjacency matrix of the graph (Knyazev, Taylor, and Amer 2019).

To improve the expressive power of GNNs, multiple works have tried to break away from this limitation of local message passing of 1-WL. Higher-order GNNs which can be seen as neural versions of k-dimensional WL tests have been proposed and studied in series of works (Maron et al. 2018; Morris et al. 2019; Maron et al. 2019; Morris, Rattan, and Mutzel 2020). Although, these models are provably more powerful than 1-WL GNNs, they suffer from computational bottlenecks as they operate on k-tuples of nodes. Maron et al. (2019) proposed a computationally feasible model but was limited to 3-WL expressivity. Note that 1-WL and 2-WL have same expressivity (Maron et al. 2019). Other line of works on improving the expressivity of GNNs tend to introduce extra features which can be computed by preprocessing the graphs in the form of distance encoding (Li et al. 2020) and subgraph isomorphism counting (Bouritsas et al. 2020). These techniques help in practice but fall short in the quest of better algorithms to extract such information by improved learning algorithms.

The main problem of message passing in GNNs is that nodes fail to uniquely identify if the successive messages received are from the same or different nodes. In order to address this issue, Sato, Yamada, and Kashima (2020) and Dasoulas et al. (2019) have proposed to introduce features with random identifiers for each node. These models, though showing some benefit, can only maintain permutation invariance in expectation. Furthermore, structural message passing in the form of matrices with unique identifiers may help as shown in Vignac, Loukas, and Frossard (2020) but its fast version has the same expressivity as in Maron et al. (2019).

In this work, we aim to address some these concerns by leveraging the *individualization and refinement* paradigm of isomorphism solvers which are often fast in practice. We make a few approximations by individualizing fixed number of nodes and keeping a single representative graph in each iteration to manage the computational complexity. To avoid loss in accuracy, we use learning both for adaptively selecting the nodes to individualize and for merging the graphs from data. Our approach provides flexible trade off between robustness and speed of the model using the number of nodes to individualize as a hyperparameter which can be tuned using training data.

Preliminaries

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Two graphs G, G' are *isomorphic*, if there exists an adjacency-preserving bijective mapping $f : \mathcal{V}_G \to \mathcal{V}'_{G'}$, i.e. $(v, u) \in \mathcal{E}$ iff $(f(v), f(u)) \in \mathcal{E}'$. An *automorphism* of G is an isomorphism that maps G onto itself. Intuitively, two vertices can be mapped to each other via an automorphism if they are structurally indistinguishable in the graph. A prominent way of identifying isomorphism is by coloring the nodes based on the structure of the graph in a permutation invariant manner and comparing if two graphs have the same coloring. Formally, a *vertex coloring* π is a surjective function $\pi : \mathcal{V} \to \mathrm{N}$ which assigns each vertex of the graph to a color (natural number). A graph G is called *colored graph* (G, π), if π is a coloring of G. Given a graph G, a *cell* of π is the set of vertices with a given color. Vertex coloring partitions $\mathcal{V}$ into *cells* of color classes and hence is often called a *partition*. If any two vertices of the same color are adjacent to the same number of vertices of each color, then such a coloring is called *equitable coloring*, which cannot be further refined. If π, π' are equitable colorings of a graph, then π' is *finer than or equal to* π, if $\pi(v) < \pi(w) \Rightarrow \pi'(v) < \pi'(w)$ for all $v, w \in V$. This implies that each cell of π' is a subset of a cell of π, but the converse is not true. A *discrete coloring* is a coloring where each color *cell* has a single vertex i.e. each vertex is assigned a distinct color.

Vertex refinement or 1-WL test

1-dimensional Weisfeiler Lehman coloring is a graph coloring algorithm used to test isomorphism between graphs. Initially, all vertices are labeled uniformly with the same color and then are refined iteratively based on the local neighbourhood

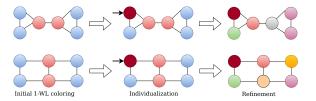Initial 1-WL coloring   Individualization   Refinement

Figure 1: Example graphs where 1-WL cannot distinguish two non-isomorphic graphs. Initially, 1-WL (or GNN) message passing refines the graphs into equitable partition of same two color *cells* and hence the graphs are indistinguishable. In the step of *individualization*, one of the blue vertex is distinguished from other vertices by recoloring it (pointed by black arrows). Subsequently, 1-WL *refinement* i.e. message passing, produces distinct multiset of colors for the two graphs, thereby distinguishing them as non-isomorphic.

of each vertex. In each iteration, two vertices are assigned different colors if the multisets of their colored neighbourhoods are different. Specifically, if $\pi^t(v)$ is the color of vertex v at time step t, then the colors are refined with the following update: $\pi^{t+1}(v) = \text{HASH}\Big(\pi^t(v), \ \{\!\{\pi^t(u), u \in N(v)\}\!\}\Big)$, where $\{\!\{\}\!\}$ denotes a multiset and $N(v)$ is the neighbourhood of v. This procedure produces an equitable coloring where no further refinement is possible and the algorithm stops. This is a powerful technique of node coloring but has been shown to be restricted for many classes of graphs which cannot be distinguished by 1-WL refinement (Kiefer et al. 2020; Arvind et al. 2020; Chen et al. 2020).

Individualization and refinement

Most of the present graph isomorphism solvers i.e. `Nauty` (McKay et al. 1981), `Traces` (McKay and Piperno 2014), etc. are based on a coloring paradigm called *individualization and refinement*. In practice, these solvers are quite fast even though they can take exponential time in worst case. A comprehensive explanation of the of individualization and refinement algorithms can be found in McKay and Piperno (2014). Below we give a brief description.

Individualization refers to picking a vertex among vertices of a given color and distinguishing it with a new color. Once a vertex is distinguished from the rest, this information can be propagated to the other nodes by 1-WL message passing. An example is shown in the Figure 1, where initially 1-WL coloring is unable to distinguish the two graphs. Individualizing one of the blue colored vertices and further refinement produces equitable coloring of the graphs such that they become distinguishable.

The procedure to generate a *canonical* coloring of the graph is as follows. Initially, vertex refinement is used to get the equitable coloring of the graph. This would partition the graph into a set of color cells. Then one of the color cells called *target cell* is chosen and a vertex from it is reassigned a new color. This is propagated across the graph till a further refined equitable coloring of the graph is obtained. But this refinement comes at a cost. This can only be done in a permutation invariant manner if all the vertices of the chosen target cell are individualized and thereafter refined. If we chose a target cell of size k i.e. the chosen color has been

assigned to k nodes in the graph, then after individualization and subsequent refinement, we would have k colored graphs which are *finer* than the one before individualization. Note that these k graphs can still be not discrete and hence the process is repeated for each of k refined graphs until all final graph colorings are discrete. This would take the shape of a search tree where tree nodes are graphs with equitable colorings. This search tree is *canonical* to the isomorphism class of the graph i.e. the search tree is unique to the graph's isomorphism class. Finally, one of the leaves is chosen based on a predefined function which sorts all discrete colorings found as leaves of the search tree.

Furthermore, vertices belonging to the same automorphism group in the graph always get the same color in the initial equitable coloring and all of them induce the same refined coloring of the graph when individualized. Intuitively, this is because there is no structural difference between the vertices of the same automorphism group. For example, in Figure 1, individualizing any of the blue vertices would produce the same set of node colorings. Therefore, isomorphism solvers prune the search tree by detecting automorphism groups. Effectively, even if the target cell is large, the branching can be reduced if automorphisms are detected.

Proposed Method

GNNs learn node embeddings, by iteratively aggregating embeddings of its neighbours. In this work, we propose to leverage the *individualization and refinement* paradigm in learning to distinguish nodes with similar embeddings. The key idea is to approximate the search process generated by IR based isomorphism solvers. To make it computationally feasible, we make neccessary approximations. A conceptual overview of our approach is shown in Figure 2 and pseudocode is shown in Algorithm 1.

Notations: We refer to hidden vectors with h_v for embeddings of node v and h_G for the aggregated embeddings of graph G, generated with appropriate global pooling function applied on node embeddings. We use $\mathbf{H} \in \mathbb{R}^{n \times d}$ to denote matrix of node embeddings. h_v^l is the embedding at layer $l \in \{1 \dots L\}$. Note that layer l refers to one full iteration of individualization-refinement step. h_v^0 are initialized with node features or with a constant value in the absence of node features.

Initialization

Graph Neural Networks (GNNs) are neural versions of 1-WL message passing and hence node embeddings would converge after iterating for fixed number of steps. At this stage, some of the nodes end up with same embeddings. This is equivalent to the equitable partitioning of the graph. This serves as the stable initialization of the node embeddings and partitions the graph akin to *color cells*.

Target cell selection

After initial GNN refinement, the graph would be a multiset of color (embedding) classes. To break the symmetry, we need to choose a color cell called the *target cell*, which contains the chosen set of nodes with the same color, in order to
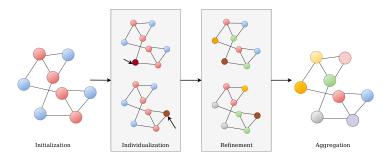
Figure 2: Given a graph, its node embeddings are initialized with a GNN. This partitions the graph into distinct color cells with nodes in a color cell having the same embedding. We select top-k nodes nearest to *target-cell* embedding and separately *individualize* them. Subsequent *refinement* by GNN message passing generates k refined graphs. Graph-aware node embeddings across k graphs are then aggregated into a single representative embedding. This IR process is repeated L times before a readout function is used to make the final prediction.

individualize each one of them. We approximate the *target-cell* selection by choosing one of the node embeddings in the graph and selecting k most similar nodes with the chosen embedding. The choice of the *target cell* is important as it determines the breadth and the depth of the search tree and thereby the richness of the embeddings learnt. Usually, the isomorphism solvers use hand-engineered techniques to choose the target cell. Note that the size of the target cell itself may not matter much with respect to the refinement it induces on the graph. Some solvers like `Nauty` choose the smallest non-singleton cell whereas `Traces` choose the largest cell in order to shorten the depth of the search tree. This is one place where we can leverage learning from data. Given a dataset, the *target cell* chosen should best fit the data. Let ϕ be a scoring function on nodes which can be used to select top-k indices for individualization in each iteration. If $\mathcal{I}^l$ is the set of nodes of the *target cell* to be individualized at layer l then,

$$\mathcal{I}^l = \texttt{Top-k-rank}(\{\phi^l(h_v^l, G; \theta), \forall v \in \mathcal{V}\}) \quad (1)$$

One simple method of implementing such a scoring function would be Top-K pooling as done in (Gao and Ji 2019; Lee, Lee, and Kang 2019). But this may not capture the best *target cell* as these functions are only parameterized by a projection vector and do not provide specific inductive bias towards an end goal. Intuitively, to bias the scoring function ϕ towards better *target cells*, it should determine the *cell* based on the current state of the graph i.e., the number and size of distinct embeddings (*cells*) and the previous embeddings that have already been individualized i.e. history. Therefore, we model the target cell selector as a Recurrent neural network with a GRU cell as it can track previous states via its hidden state.

Precisely, in each step, the GRU cell outputs a projection vector which is used to rank the nodes based on the similarity of the nodes with the projection vector. At iteration l, the GRU cell takes in two inputs, one the pooled feature of the graph h_G^l and the other is the hidden state of the GRU from the previous time step. It then produces two embeddings, the updated hidden state and an output embedding. The output is considered as a surrogate for the target cell embedding on which all the nodes are projected and ranked based on the projection score. Let $\mathbf{p}_{tc}$ be the projection vector output by the GRU, AGG be an aggregator set function, $\mathbf{q}_{tc}$ be the hidden state vector and $\mathbf{H}$ be the set of node embeddings.

Algorithm 1: The framework of GNN-IR

1: **Input:** Graph $G = (\mathcal{V}, \mathcal{E})$;
2: d-layer GNN with 1-WL expressive power;
3: Target cell selector function ϕ;
4: Individualization function ψ;
5: Multiset aggregators AGG; MLP f;
6: **Output:** Refined embedding for Graph G
7: Initialize $\mathbf{H}^0 = \{h_v^0, \forall v \in \mathcal{V}\}$ with constant value
8: $\mathbf{H}^1 \leftarrow \text{GNN}(\mathbf{H}^0)$
9: **for** $l = 1$ **to** L **do**
10:   $\mathcal{I}^l = \texttt{Top-k-rank}(\{\phi^l(h_v^l, \mathbf{H}^l; \theta), \forall v \in \mathcal{V}\})$
11:   **for** $i \in \mathcal{I}^l$ **do**
12:     $\mathbf{H}^{l^i} \leftarrow \mathbf{H}^l$
13:     $h_i^{l^i} \leftarrow h_i^{l^i} \odot \psi(h_i^{l^i})$ // Individualization
14:     $\tilde{\mathbf{H}}^{l^i} \leftarrow \text{GNN}(\mathbf{H}^{l^i})$ // Refinement
15:     $\tilde{h}_G^{l^i} \leftarrow \text{AGG}(\tilde{\mathbf{H}}^{l^i})$
16:     $\tilde{\mathbf{H}}^{l^i} \leftarrow \{f(\tilde{h}_v^{l^i}, \tilde{h}_G^{l^i}; \theta), \forall v \in \mathcal{V}\}$
17:   **end for**
18:   $\mathbf{H}^{l+1} \leftarrow \{\text{AGG}(\{\tilde{h}_v^{l^i} \mid \forall i \in \mathcal{I}\}) \mid \forall v \in \mathcal{V}\}$ // Merge
19: **end for**
20: **Return** $\{\text{AGG}(\{h_v^l \mid l = 0, \dots L\}) \mid \forall v \in \mathcal{V}\}$

Then the following rule is applied to rank the nodes and get the k individualization indices.

$$h_G^l = \text{AGG}(\mathbf{H}^l) \quad (2)$$

$$\mathbf{p}_{tc}^l, \mathbf{q}_{tc}^l = \text{GRU}(h_G^l, \mathbf{q}_{tc}^{l-1}) \quad (3)$$

$$\text{score}^l = \tanh\left(\frac{\mathbf{H}^l \mathbf{p}_{tc}^l}{\|\mathbf{p}_{tc}^l\|}\right) \quad (4)$$

$$\mathcal{I} = \text{top-k-rank}(\text{score}, k) \quad (5)$$

$$\text{Readout}(\mathbf{p}_{tc}^l \odot \text{score}^l)_{\mathcal{I}} \quad (6)$$

where $\odot$ is elementwise product. To select the nodes, the node embeddings are projected onto $\mathbf{p}_{tc}$ and a score is computed for each node. The top-k ranked indices are selected into the set $\mathcal{I}$. To be able to backpropagate through the scoring function, the top-k scores are multiplied with the projection vector and a Readout function is used to generate a separate feature vector which is concatenated with the final graph embedding before prediction.

Individualization

Now that we have the selected k indices in $\mathcal{I}$, we individualize each of the k nodes separately and branch out. Our individualization function ψ is an MLP and acts only on the selected

node embeddings. The node embeddings of h_i, $\forall i \in \mathcal{I}$ are updated by passing through individualization function; thereafter, the embedding matrix of the graph is updated by the procedure of masking.

$$\mathbf{Z}_{mask} = \mathbf{1}; \quad \mathbf{Z}_{mask}[i] = \psi(h_i^{l^i}); \quad \mathbf{H} = \mathbf{H} \odot \mathbf{Z}_{mask}; \quad (7)$$

This is repeated for each $i \in \mathcal{I}$ separately as we can only color one node at a time in the graph. The node embedding matrix $\mathbf{H}$ is updated with the individualized node embedding. The next step is to refine the graph to propagate this information.

Refinement and Aggregation

After each individualization, we run GNN again for a fixed number of steps till the node embeddings converge again. This process of *Refinement* gives us k refined graphs. Ideally, we should expand all the k graphs further but this incurs extra computational cost which grows exponentially with depth.

In cases where we need to search over a tree, one of the popular greedy heuristic is beam search, where the search tree of a fixed width is kept. Selection of best graph out of k would require a value-function approximator which would score over each of the k graphs. Such a function would be difficult to train given the limited amount of labeled data. As an approximation, we instead construct a multiset function to aggregate the k embeddings of each node into a single embedding. In principle a universal multiset function approximator should be able to approximate selection as well. Also, here we need a set function of a set of sets i.e. set of k graphs. For this we first construct graph-aware node embeddings by combining the node embeddings with the pooled embeddings of the same graph. Finally, node embeddings are aggregated across k graphs to generate a new representative graph. Precisely, let $\tilde{\mathbf{H}}^{l^i}$ be the set of node embeddings after individualization and refinement of index i in layer l. We first compute $\tilde{h}_G^{l^i}$, a representation for G^{l^i} as

$$\tilde{h}_G^{l^i} = \sum_{v \in \mathcal{V}} \text{MLP}(\tilde{\mathbf{H}}^{l^i}) \quad (8)$$

Thereafter, we feed $\tilde{h}_G^{l^i}$ to an MLP and add to each $\tilde{h}_v^{l^i}$ to get a graph-representative node embedding for the nodes of the k refined graphs.

$$\tilde{h}_v^{l^i} \leftarrow \tilde{h}_v^{l^i} + \text{MLP}(\tilde{h}_G^{l^i}) \quad (9)$$

We then max pool the node embeddings across the k graphs i.e. for a node v, we pool together k embeddings $\tilde{h}_v^{l^i}$ to generate the aggregated new representation for node v.

$$h_v^{l+1} = \max_{i \in \{1...k\}} \tilde{h}_v^{l^i} \quad (10)$$

With the new representative graph embeddings, we repeat the IR procedure L times before readout.

On aggregation of graphs and fixing k

IR algorithms usually expand the tree until the colors are discrete. The colors are then used to label the leaves and the leaf with the largest label is selected. One way to simulate this is to learn a method of selecting the correct branch at each internal tree node in order to reach the correct leaf. Instead of selecting a branch, we learn to aggregate the children of the internal nodes into a single node and find that aggregation

works well in our experiments. In an ideal case where the multiset functions used for aggregation are universal approximators (Zaheer et al. 2017; Qi et al. 2017), it would be able to learn to an approximate algorithm that selects one of the k graphs, hence can learn to work as well as an algorithm that learns to select. We use a simpler aggregrator that is not necessarily a universal approximator. However, the use of max-pooling operator will likely make simulation of selection easier – if all embeddings are positive, multiplying non-desired embeddings by a small number would result in max-pooling selecting the desired embedding.

As for determining the value of k, different graphs can have different cell sizes and hence it is best to choose the value of k by cross-validation. The best k that fits the dataset is more likely to be the target cell size on average for a given layer. Cross-validation also helps when the cells consist of nodes from same automorphism group; for such cells, smaller value of k would be sufficient. But as shown in Section , a larger value of k would be more robust by including all nodes of the selected cell.

Analysis

In this section, we discuss some of the properties of the proposed GNN model with *individualization and refinement* as defined in Algorithm 1, which we call as GNN-IR model.

Permutation invariance: Each step of GNN-IR with arbitrary width k preserves the property of permutation-invariance for a large class of graphs, though it may fail to preserve it for some graphs. All operators used in GNN-IR are either permutation-invariant operators on sets or those which operate on individual nodes. If the input graph has unique attributes for all nodes, then the node-operators will operate on same nodes irrespective of input node-permutations. Hence, GNN-IR is permutation-invariant for all graphs with unique node attributes. However, if the nodes are not distinguishable, then node-operators may operate on different nodes when the permutation of input nodes changes. The stage which decides which nodes are operated on is the *target cell* selection stage where top-k nodes are selected for individualization.

Consider the graphs with no node-attributes. Let the initial coloring of the graph be $\pi = \{p_1, \ldots, p_{|\pi|}\}$, where p_i is the set of vertices with same color. For arbitrary width k, it is possible that not all vertices of a color-cell p_i are included in the individualization set $\mathcal{I}$. Then, permutation-invariance is preserved if, upon individualization, all vertices of p_i induce same refinement on the graph. Effectively, which $v \in p_i$ is included in $\mathcal{I}$ is irrelevant. This happens when p_i forms an orbit i.e. all vertices of p_i can be mapped to each other via an automorphism. Therefore, for arbitrary k, if all $p_i \in \pi$ are orbit cells *i.e.* π is an orbit-partition, then each step of GNN-IR preserves permutation invariance of the graph embeddings. For example, consider one of the graphs in Figure 1, where blue vertices form an orbit *i.e.* all blue vertices are structurally equivalent, and hence individualization of any blue vertex of the graph will generate same refinement. Also, we show in Lemma 1 that, output of each step of GNN-IR remains an orbit-partitioned coloring and hence, permutation-invariance is preserved over multiple steps of GNN-IR.

Lemma 1. *If the input to a GNN-IR step, which includes target-cell selection, individualization-refinement and aggregation, is an orbit-partitioned coloring, then the output will also be an orbit-partitioned coloring.*
Proof: Proof is included in the Appendix.

Expressive power: We characterise the expressive power of GNN-IR in terms of its distinguishing capacity of the non-isomorphic graphs.

Proposition 1. *Assume we use universal set approximators in GNN-IR for target-cell selection, refinement and aggregation steps. GNN-IR is more expressive than all 1-WL equivalent GNNs i.e., GNN-IR can distinguish all graphs distinguishable by GNN and there exist graphs non-distinguishable by GNN which can be distinguished by GNN-IR.*
Proof sketch. A detailed proof is given in Appendix. Here, we give a brief sketch of the proof. First, we show that graphs distinguishable by GNNs generate orbit-partitions on 1-WL refinement. Hence, by Lemma 1, GNN-IR preserves permutation-invariance on these graphs and since, the first layer of GNN-IR is a GNN, they are distinguishable by GNN-IR. Next, we illustrate two graphs which are not-distinguishable by GNN and show how a step of GNN-IR including individualization-refinement and aggregation operators, distinguishes these two graphs. $\square$

Runtime analysis: For bounded degree graphs, runtime of GNN for fixed iterations is $O(n)$. GNN-IR builds on GNN and in each IR step, GNN is run k times. Assuming constant time aggregation with 1-layer MLPs, running GNN-IR for L steps takes $O(Lkn)$.

Experiments

In this section, we report evaluation results of GNN-IR with multiple experiments.

Model architecture: We follow the MPNN (Gilmer et al. 2017) architecture in all our experiments. We first convolve with a 1-WL convolution operator, then update with a GRU cell (Chung et al. 2014) and for readout, we use either sum-pooling or set2set (Vinyals, Bengio, and Kudlur 2015) function. For convolution, we use either GIN (Xu et al. 2018), NNConv (Gilmer et al. 2017) or PNA (Corso et al. 2020) convolution operators depending on the dataset and availability of edge features. Also, we run GNN in each IR layer for 3 steps and share parameters of the GNN in each iteration of IR which allows us to go deeper without increasing the parameters. We use 1-hidden layer MLPs in all set aggregators along with sum pooling. Also, in datasets with edge features, we consider edges as variables and readout from node and edge variables before final prediction. Code was implemented in Pytorch-Geometric (Fey and Lenssen 2019).

Counting Triangles

Counting triangles is a simple task and has an analytic solution of $\text{trace}(A^3)/6$, where A is the adjacency matrix of the graph. But, 1-WL GNN models provably cannot count triangles (Chen et al. 2020). Therefore, we evaluate GNN-IR on a publicly available synthetic dataset TRIANGLES (Knyazev, Taylor, and Amer 2019) of 45000 graphs where the task is to count the number of triangles. The dataset comes with 4 splits,

training (30000), validation (5000), test-original (5000) and test-large (5000). The first three sets all have graphs with nodes < 25 and the test-large set has as many as 100 nodes. The test-large is challenging and it tests the generalization ability of the model. We compare with baseline 1-WL GNN models, GAT, GIN and ChebyGIN (Knyazev, Taylor, and Amer 2019) which is a more powerful GNN.

Table 1 shows the accuracy of the models on both the test sets. To analyse the effect of depth and width of IR, we report results for various values of L, number of IR layers and k, number of individualizations in each layer. As shown in results, just one layer of IR with $k = 2$ indi-

		Train	Test		Time	
			orig	large	sec/ep	ratio
GIN, top-k		$90_{\pm 3}$	$47_{\pm 2}$	$18_{\pm 1}$	-	-
ChebyGIN		-	$64_{\pm 5}$	$25_{\pm 2}$	-	-
GAT		$92_{\pm 2}$	$50_{\pm 1}$	$25_{\pm 1}$	-	-
GIN*		$90_{\pm 3}$	$47_{\pm 2}$	$25_{\pm 1}$	17.1	1
GNN - IR						
$L = 1$	$k = 2$	$90_{\pm 3}$	$76_{\pm 3}$	$28_{\pm 2}$	42.1	2.46
$L = 1$	$k = 4$	$92_{\pm 3}$	$86_{\pm 1}$	$34_{\pm 2}$	51.7	3.02
$L = 2$	$k = 2$	$86_{\pm 2}$	$78_{\pm 2}$	$28_{\pm 4}$	57.0	3.33
$L = 2$	$k = 4$	$98_{\pm 1}$	$97_{\pm 1}$	$41_{\pm 2}$	80.8	4.73
$L = 3$	$k = 2$	$93_{\pm 3}$	$91_{\pm 2}$	$46_{\pm 2}$	85.6	5.00
$L = 3$	$k = 4$	$99_{\pm 0}$	$99_{\pm 1}$	$51_{\pm 1}$	98.7	5.77

Table 1: Accuracy on counting the number of triangles in TRIANGLES dataset. L is the number of layers of IR and k is the width parameter. *Our baseline model with $L = 0, k = 0$.

vidualizations significantly increases the accuracy from 47 to 76 for Test-orig. Furthermore, a clear pattern emerging from the results is that accuracy and generalization improves for longer depth L and larger width k, with highest score saturating at $L = 3$ and $k = 4$. We also report the extra time taken by each IR layer, which includes 3 steps of GNN per individualization, and compare it with GIN as it is our base GNN. The time ratio w.r.t GIN shows that with more IR layers and increased width k, the amount of computation also increases; but the increase is only linear both in k and L.

Recognizing Circulant Skip Links (CSL) graphs

A type of graphs that 1-WL GNN models cannot classify are regular graphs that do not provide any information in node degrees. In order to evaluate GNN-IR for

		mean	median	max	min	std
GIN*		10	10	10	10	0
RP-GIN		37.6	43.3	53.3	10	12.9
3WLGNN		97.8	−	100.0	30	10.9
GNN - IR						
$L - 5$	$k - 04$	50	60	70	10	23.45
$L = 3$	$k = 08$	74	70	90	60	15.17
$L = 5$	$k = 08$	82	80	100	70	13.04
$L = 2$	$k = 16$	82	80	90	80	4.47
$L = 3$	$k = 16$	96.67	100.0	100.0	90.0	4.7
$L = 4$	$k = 16$	98.67	100.0	100.0	96.67	1.8

Table 2: Graph classification on the CSL dataset. *Our baseline model with $L = 0, k = 0$.

highly regular graphs, we use Circulant Skip Links (CSL) dataset released by (Murphy et al. 2019). A CSL graph $\mathcal{G}_{skip}(M, R)$ is a 4-regular graph with $\{0, 1, \ldots M - 1\}$ vertices with an edge between vertices R distance from each other. The dataset consists of 150 graphs with 10 classes (R). We evaluate GNN-IR with CSL dataset with 5-fold cross-validation as in (Murphy et al. 2019) and report results along with 1-WL and more expressive models like RP-GNN (Murphy et al. 2019) and 3-WL GNN.

Data set	ZINC 10K mae	ALCHEMY 10K mae	log-mae
GINE-ϵ	$0.278_{\pm0.022}$	$0.185_{\pm0.007}$	$-1.864_{\pm0.062}$
2-WL-GNN	$0.399_{\pm0.006}$	$0.149_{\pm0.004}$	$-2.609_{\pm0.029}$
δ-2-GNN	$0.374_{\pm0.022}$	$\mathbf{0.118}_{\pm0.001}$	$-2.679_{\pm0.044}$
δ-2-LGNN	$0.306_{\pm0.044}$	$0.122_{\pm0.003}$	$-2.573_{\pm0.078}$
DGN	$0.168_{\pm0.010}$		-
PNA	$0.187_{\pm0.010}$	$0.162_{\pm0.005}$	$-2.033_{\pm0.054}$
GNN-IR	$\mathbf{0.137}_{\pm0.010}$	$0.119_{\pm0.002}$	$\mathbf{-2.742}_{\pm0.070}$

Table 3: MAE on ZINC10K and ALCHEMY10K

Data set	QM9 (mae)
GINE-ϵ	$0.081_{\pm0.003}$
MPNN	$0.034_{\pm0.001}$
1-2-GNN	$0.068_{\pm0.001}$
1-3-GNN	$0.088_{\pm0.007}$
1-2-3-GNN	$0.062_{\pm0.001}$
3-IGN	$0.046_{\pm0.001}$
δ-2-LGNN	$0.029_{\pm0.001}$
LRBP-net	$0.027_{\pm0.001}$
GNN-IR	$\mathbf{0.020}_{\pm0.001}$

Table 4: Graph regression on QM9 dataset in std mae.

Data set	$O(.)$	ZINC10K		ALCHEMY10K	
		Time-ratio	mae	Time-ratio	mae
GINE (1-WL)	$O(n)$	1.0	0.27	1.0	0.18
2-WL-GNN	$O(n^2)$	17.11	0.39	7.33	0.14
δ-2-WL-GNN	$O(n^2)$	21.86	0.37	10.71	0.11
GNN - IR	$O(Lkn)$				
L=1 k=2	$O(2n)$	4.61	0.16	2.38	0.13
L=1 k=4	$O(4n)$	6.87	0.14	4.47	0.12
L=2 k=2	$O(4n)$	7.36	0.13	4.72	0.12
L=2 k=4	$O(8n)$	11.56	0.13	7.22	0.11

Table 5: Runtime analysis: Time-ratio compared to GINE (1-WL).

	CSL					TRIANGLES		
	mean	median	max	min	std	Train	Test	
							orig	large
GIN	10	10	10	10	0	90	47	18
GNN-IR with								
a) random k nodes	11.9	13.3	13.3	10	1.8	94.7	93.6	39.5
b) without GRU	90.7	96.7	96.7	70	16.6	97.1	97.2	41.2
c) sum aggr	54.0	60.0	70.0	20	19.4	95.8	92.4	35.2
GNN-IR	**98.7**	**100.0**	**100.0**	**96.67**	**1.8**	**99.4**	**99.3**	**51.1**

Table 6: GNN-IR ablation models

Since CSL graphs have highly regular structure, the number of individualizations needed to break the symmetry is high. Table 2 shows that 1-WL GIN model fails whereas there is clear improvement by GNN-IR. Note that higher values of k result in much better gain compared to increase in number of IR layers L. This suggests that with larger width k, either target color-cells with larger width are more helpful in breaking the symmetry or more number of smaller color-cells included for individualization helps in including the optimal target-cell in the selected k nodes.

Real world benchmarks

We now evaluate GNN-IR on real-world datasets on graph regression task. We use ZINC10K (Jin, Barzilay, and Jaakkola 2018; Dwivedi et al. 2020), ALCHEMY10K (Chen et al. 2019a) and QM9 (Ruddigkeit et al. 2012; Ramakrishnan et al. 2014) datasets, since these datasets are used by recent prominent models (Morris, Rattan, and Mutzel 2020; Corso et al. 2020) and to compare with the neural versions of higher-order GNNs (Morris, Rattan, and Mutzel 2020) which are more expressive than 1-WL GNNs. Note that all kernel versions usually perform much better than their neural counterparts. We use NNConv as our base GNN for QM9 and PNAConv for Alchemy and ZINC. For readout, we use Set2set output for QM9 following (Gilmer et al. 2017) and sum pooling for other datasets. For all datasets we follow the dataset splits and report mean absolute error (MAE) as in (Morris, Rattan, and Mutzel 2020). For more recent comparison with state-of-the-art models, we add PNA (Corso et al. 2020) and DGN (Beani et al. 2021)for ZINC and ALCHEMY datasets and LRBP-net (Dupty and Lee 2020) for QM9.

Table 3 and 4 shows that the improvement over standard baselines is consistent across the datasets. Specifically for ZINC10K and QM9, the improvement of GNN-IR is sub-stantial over k-WL GNN variants. Note that it is non-trivial to compare the expressive power of GNN-IR in terms of k-WL models. Results show that GNN-IR outperforms neural versions of k-WL. This suggests that GNN-IR has at least better inductive bias for these problems than k-WL GNNs. But it is not clear which properties of the algorithm result in better inductive bias for these tasks. It would be interesting to understand the conditions under which GNN-IR and k-WL GNNs do well for different tasks. We leave this study for future work.

Runtime and ablation study: We compare GNN-IR with k-WL GNNs in terms of their runtimes against accuracy. Table 5 shows ratio of time per epoch of models w.r.t GINE (1-WL). There is only linear increase in runtime of GNN-IR with increasing L, k and much better gains compared to k-WL GNNs with lesser runtimes.

We also conducted ablation experiments to study the effectiveness of various components of GNN-IR. For this, we use the best models of GNN-IR for CSL and TRIANGLES dataset and consider randomly selecting top-k nodes instead of learning, replacing GRU in equation 3 with an MLP and using sum instead of max in aggregating the k refined graphs. Table 6 shows that, compared to GNN, randomly selecting top-k nodes helps in TRIANGLES dataset but does not help at all in CSL dataset; suggesting that learning top-k nodes is needed for breaking the symmetry in graph structures. Replacing GRU with MLP and using sum aggregator decreases the performance slightly but fare much better than GNN.

Note: Due to space constraints, we provide results on TU-Dataset benchmark in the Appendix; GNN-IR gives competitive performance against GNN-based methods although it does not give state-of-the-art performance when compared to all methods.

Conclusion

In this work, we propose learning richer representations on graph structured data with *individualization and refinement*, a technique followed by most practical isomorphism solvers. Our approach is computationally feasible and can adaptively select nodes to break the symmetry in GNN embeddings. Experimental evaluation shows that our model substantially outperforms other 1-WL and more expressive GNNs on several benchmark datasets. Future work includes understanding the power and limitations of learning individualization and refinement functions for improving GNN models.

References

Arvind, V.; Fuhlbrück, F.; Köbler, J.; and Verbitsky, O. 2020. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113: 42–59.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J. P. 2019. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*.

Beani, D.; Passaro, S.; Létourneau, V.; Hamilton, W.; Corso, G.; and Liò, P. 2021. Directional graph networks. In *International Conference on Machine Learning*, 748–758. PMLR.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828.

Bouritsas, G.; Frasca, F.; Zafeiriou, S.; and Bronstein, M. M. 2020. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*.

Chen, G.; Chen, P.; Hsieh, C.-Y.; Lee, C.-K.; Liao, B.; Liao, R.; Liu, W.; Qiu, J.; Sun, Q.; Tang, J.; et al. 2019a. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*.

Chen, Z.; Chen, L.; Villar, S.; and Bruna, J. 2020. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025*.

Chen, Z.; Villar, S.; Chen, L.; and Bruna, J. 2019b. On the equivalence between graph isomorphism testing and function approximation with gnns. *arXiv preprint arXiv:1905.12560*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Corso, G.; Cavalleri, L.; Beaini, D.; Liò, P.; and Veličković, P. 2020. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*.

Darga, P. T.; Liffiton, M. H.; Sakallah, K. A.; and Markov, I. L. 2004. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference*, 530–534.

Dasoulas, G.; Santos, L. D.; Scaman, K.; and Virmaux, A. 2019. Coloring graph neural networks for node disambiguation. *arXiv preprint arXiv:1912.06058*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Dehmamy, N.; Barabási, A.-L.; and Yu, R. 2019. Understanding the representation power of graph neural networks in learning graph topology. *arXiv preprint arXiv:1907.05008*.

Du, S. S.; Hou, K.; Póczos, B.; Salakhutdinov, R.; Wang, R.; and Xu, K. 2019. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *arXiv preprint arXiv:1905.13192*.

Dupty, M. H.; and Lee, W. S. 2020. Neuralizing Efficient Higher-order Belief Propagation. *arXiv preprint arXiv:2010.09283*.

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

Errica, F.; Podda, M.; Bacciu, D.; and Micheli, A. 2019. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gao, H.; and Ji, S. 2019. Graph u-nets. In *international conference on machine learning*, 2083–2092. PMLR.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1263–1272. JMLR. org.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017a. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017b. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, 2323–2332. PMLR.

Junttila, T.; and Kaski, P. 2011. Conflict propagation and component recursion for canonical labeling. In *International Conference on Theory and Practice of Algorithms in (Computer) Systems*, 151–162. Springer.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels, 2016. *URL http://graphkernels. cs. tu-dortmund. de*, 795.

Kiefer, S.; Immerman, N.; Schweitzer, P.; and Grohe, M. 2020. Power and limits of the Weisfeiler-Leman algorithm. Technical report, Fachgruppe Informatik.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klicpera, J.; Groß, J.; and Günnemann, S. 2020. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*.

Knyazev, B.; Taylor, G. W.; and Amer, M. R. 2019. Understanding attention and generalization in graph neural networks. *arXiv preprint arXiv:1905.02850*.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In *International Conference on Machine Learning*, 3734–3743. PMLR.

Li, P.; Wang, Y.; Wang, H.; and Leskovec, J. 2020. Distance Encoding–Design Provably More Powerful GNNs for Structural Representation Learning. *arXiv preprint arXiv:2009.00142*.

Loukas, A. 2019. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*.

Maron, H.; Ben-Hamu, H.; Serviansky, H.; and Lipman, Y. 2019. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*.

Maron, H.; Ben-Hamu, H.; Shamir, N.; and Lipman, Y. 2018. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*.

McKay, B. D.; and Piperno, A. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60: 94–112.

McKay, B. D.; et al. 1981. Practical graph isomorphism.

Morris, C.; Rattan, G.; and Mutzel, P. 2020. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33.

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

Murphy, R.; Srinivasan, B.; Rao, V.; and Ribeiro, B. 2019. Relational pooling for graph representations. In *International Conference on Machine Learning*, 4663–4673. PMLR.

Nikolentzos, G.; and Vazirgiannis, M. 2020. Random Walk Graph Neural Networks. *Advances in Neural Information Processing Systems*, 33: 16211–16222.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 652–660.

Ramakrishnan, R.; Dral, P. O.; Rupp, M.; and Von Lilienfeld, O. A. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1: 140022.

Ruddigkeit, L.; Van Deursen, R.; Blum, L. C.; and Reymond, J.-L. 2012. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. *Journal of chemical information and modeling*, 52(11): 2864–2875.

Sato, R.; Yamada, M.; and Kashima, H. 2020. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*.

Schütt, K. T.; Sauceda, H. E.; Kindermans, P.-J.; Tkatchenko, A.; and Müller, K.-R. 2018. SchNet–A deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24): 241722.

Shervashidze, N.; Schweitzer, P.; Van Leeuwen, E. J.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Srinivasan, B.; and Ribeiro, B. 2019. On the Equivalence between Positional Node Embeddings and Structural Graph Representations. In *International Conference on Learning Representations*.

Unke, O. T.; and Meuwly, M. 2019. PhysNet: A neural network for predicting energies, forces, dipole moments, and partial charges. *Journal of chemical theory and computation*, 15(6): 3678–3693.

Vignac, C.; Loukas, A.; and Frossard, P. 2020. Building powerful and equivariant graph neural networks with structural message-passing. *arXiv e-prints*, arXiv–2006.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yanardag, P.; and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 1365–1374.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R.; and Smola, A. 2017. Deep sets. *arXiv preprint arXiv:1703.06114*.

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

Appendix

In this section, we first provide proofs of the propositions. Thereafter, we expand our Experiments section by including additional set of experiments. Specifically, we provide results on TUDataset benchmark and additional details on experiments and datasets.

Proofs

Permutation invariance:
Lemma 1. *If the input to a GNN-IR round is an orbit-partitioned coloring, then the output will also be an orbit-partitioned coloring.*

Proof: Each round of GNN-IR consists of selection of k nodes, individualization-refinement of k nodes to generate k refined colorings and thereafter, aggregation of k colorings into a single coloring. Firstly, since input is an orbit-partition, the node selection stage is permutation-invariant. The k refined colorings generated after individualization-refinement stage are all orbit-partitions. This is because any refined sub-partition of an orbit-partition is also an orbit-partition. As a proof by contradiction for this: assume the sub-partition is not an orbit-partition. Then this implies there exist two vertices in a color-cell of the sub-partition which cannot be mapped to each other via an automorphism. From the definition of color-refinement, all color cells of sub-partition are subsets of some orbit-cell in the initial orbit-partition which implies there exists an automorphism between the two vertices which is a contradiction of the assumption. Hence, the refined colorings are orbit-partitions.

Now, the k refined colorings of the initial orbit-partition are mapped to a single coloring by the aggregation operation. Given that all the k refined colorings are finer than the initial coloring, we need to show that any node-wise aggregation will either be finer or at least equal to the initial orbit-partition in terms of the number and size of the color cells, though the embeddings may be transformed. Equivalently, if two vertices have different colors before IR step, then they should have different colors after IR step.

Consider the node-wise aggregation operation. It takes k colorings as input and for each node, injectively maps the set of k colors/embeddings to a new color. If two vertices have different colors in at least one of the k colorings, then they will be mapped to different colors except in one case. The only case when two vertices v_a and v_b with different colors in at least one of the k colorings, are mapped to the same color is when the refinements are rotations of each other. For simplicity, consider aggregating $k = 2$ refinements; similar argument follows for higher values of k. Let v_i and v_j be the individualizing vertices for generating the 2 refinements and $\pi_{v_i}^l(v)$ be the color of v in the refined coloring after individualizing v_i. If the two refined colorings are rotations of each other, then $\pi_{v_i}^l(v_a) = \pi_{v_j}^l(v_b)$ and $\pi_{v_j}^l(v_a) = \pi_{v_i}^l(v_b)$ is true for some vertices v_a and v_b. Then, when we aggregate nodes across k colorings as a set of colors, the set becomes same for both v_a and v_b i.e., $\{\pi_{v_i}^l(v_a), \pi_{v_j}^l(v_a)\} = \{\pi_{v_i}^l(v_b), \pi_{v_j}^l(v_b)\}$. Below, we show that such a condition cannot occur for any pair of vertices with different colors in the initial orbit-partition.

For this, consider two vertices v_a and v_b with different colors in the initial orbit-partition i.e. $\pi^l(v_a) \neq \pi^l(v_b)$ where π^l is the coloring before layer l of GNN-IR. For clarity, let x and y be the colors of v_a and v_b respectively with $x \neq y$, in the initial orbit-partition. Now, if the individualizing vertices v_i and v_j belong to different color-cells, then the refinements induced will be different. This can be shown from the procedure of vertex-refinement (Shervashidze et al. 2011). Therefore, the new colors of v_a and v_b will be different in both the refinements i.e. colorings due to v_i will be x', y' and colorings due to v_j will be x'', y''. Note that $x' \neq y''$ and $y' \neq x''$, since after vertex-refinement, the new color of a vertex depends both on its previous color and color of individualizing vertex. In this case, we have different colors for all four vertices i.e., $\pi^l(v_i) \neq \pi^l(v_j) \neq \pi^l(v_a) \neq \pi^l(v_b)$ which implies $x' \neq x'' \neq y' \neq y''$. Therefore, the node-wise aggregated sets are $\{x', x''\} \neq \{y', y''\}$ and hence, $\pi^{l+1}(v_a) \neq \pi^{l+1}(v_b)$ after aggregation.

If the individualizing vertices v_i and v_j belong to same color-cells, then the refinements induced will be same. But since their initial colors were different, the node-wise aggregated set of colors would be different for each vertex. Equivalently, if the colors of v_a, v_b due to individualizing v_i are x', y' respectively, then the colors due to v_j will also be x', y'. In this case, the node-wise aggregated sets are $\{x', x'\} \neq \{y', y'\}$ and hence, $\pi^{l+1}(v_a) \neq \pi^{l+1}(v_b)$ after aggregation.

Hence, the aggregation operation generates coloring which is also an orbit partition. $\square$

Expressive power of GNN-IR: We now proceed to the proof of Proposition 1 which states that GNN-IR is more expressive than GNN. For the proof, we need to know the conditions under which the aggregation procedure of GNN-IR as described in Algorithm 1 maps two sets of k colorings to different colorings. The following Lemma shows one of the conditions under which the aggregation can map refinements of two non-isomorphic graphs to distinct aggregated colorings.

Lemma 2. *Assume we use we use universal set approximators in aggregation function to merge k refined colorings (colored graphs) in Algorithm 1. Consider two sets of k colorings Π_a and Π_b such that $\Pi_a = \{\pi_{a_1}, \pi_{a_2}, \ldots, \pi_{a_k}\}$ and $\Pi_b = \{\pi_{b_1}, \pi_{b_2}, \ldots, \pi_{b_k}\}$. If $\Pi_a \cap \Pi_b = \varnothing$ i.e. Π_a and Π_b do not share any coloring out of k colorings, then, irrespective of the node embeddings, the aggregation function of GNN-IR maps Π_a and Π_b to different colorings.*

Proof: With the assumption of universal set approximators for AGG in line 15 and 18 of Algorithm 1, the aggregation function first maps each ith coloring to a unique $\tilde{h}_G^{l_i}$ embedding. Note that, we then concatenate $\tilde{h}_G^{l_i}$ with each of the node embeddings of

the ith coloring. We then use another set function approximator to reduce k node embeddings to one. For Π_a and Π_b, since $\Pi_a \cap \Pi_b = \varnothing$, the corresponding set of $\tilde{h}_G^{l^i}$'s will be different and consequently, irrespective of node embeddings $\tilde{h}_v^{l^i}$, every vertex across k colorings, will be mapped to different embedding after the concatenation $[\tilde{h}_v^{l^i}, \tilde{h}_G^{l^i}]$. Therefore, with the set aggregation of the k concatenated node embeddings, Π_a and Π_b will be mapped to different colorings. $\square$

Below, we give the extended proof of Proposition 1.

Proposition 1. *Assume we use universal set approximators in GNN-IR for target-cell selection, refinement and aggregation steps. GNN-IR is more expressive than all 1-WL equivalent GNNs i.e., GNN-IR can distinguish all graphs distinguishable by GNN and there exist graphs non-distinguishable by GNN which can be distinguished by GNN-IR.*

Proof: Consider the procedure of generating the final embedding of GNN-IR used for graph prediction. After every step of GNN-IR, we pool the node embeddings for readout. Then we concatenate the pooled embeddings of each layer and feed it to an MLP for the final prediction. If h_G^l is the pooled graph embedding after layer l, then the concatenated embedding will be $[h_G^0, h_G^1, \ldots, h_G^L]$ for GNN-IR with L layers. Clearly, this concatenated embedding must be different for all the graphs distinguishable by GNN-IR.

First, consider $\mathcal{G}^{1WL}$ as the set of graphs such that the equitable coloring induced after 1-WL/GNN refinement on any $G \in \mathcal{G}^{1WL}$ uniquely identifies G. In other words, $\mathcal{G}^{1WL}$ is the set of graphs which are distinguishable by any 1-WL equivalent GNN. Note that if an equitable coloring π uniquely identifies a graph, then its color-cells form the vertex orbits of the automorphism group of the graph (Kiefer et al. 2020). Therefore, the equitable coloring induced by GNN refinement on any $G \in \mathcal{G}^{1WL}$ also forms an orbit partition and GNN-IR with any width k preserves its permutation-invariance. Therefore, the final GNN-IR embedding $[h_G^0, h_G^1, \ldots, h_G^L]$ will be permutation-invariant regardless of width k and length L. Now, since $G \in \mathcal{G}^{1WL}$ is distinguishable by GNN, h_G^0 will be unique for G and will be distinguishable by GNN-IR.

Next, we need to show that there exist graphs which are not distinguishable by any 1-WL equivalent GNN but are distinguishable by GNN-IR. Graphs A and B in Figure 3 are two such graphs which are not distinguishble by GNN and hence, h_G^0 will be same for these two graphs. It can be shown that initial 1-WL refinement of graphs A and B generates orbit partitioned colorings. Such a refinement will effectively partition the graphs into two cells of orbits i.e. blue and red colored vertices. Since, the initial 1-WL refinement of both the graphs generates orbit-partitioned colorings, by Lemma 1, GNN-IR preserves permutation-invariance for any number of IR steps.

As illustrated in Figure 3, the individualization of either of the blue or red vertices induces different refined colorings for graphs A and B. Since the number of vertices are 6, any value of $k <= 6$ will result in distinct multiset of colorings Π_A and Π_B for graphs A and B respectively, such that $\Pi_A \cap \Pi_B = \varnothing$. With Lemma 2., the aggregation function maps the two graphs to distinct colorings which can be pooled to get h_G^l. Hence, in the final embedding $[h_G^0, h_G^1, \ldots, h_G^L]$, h_G^l for any $l > 0$, will be different for the two graphs. Therefore, these graphs can be distinguished by GNN-IR. $\square$
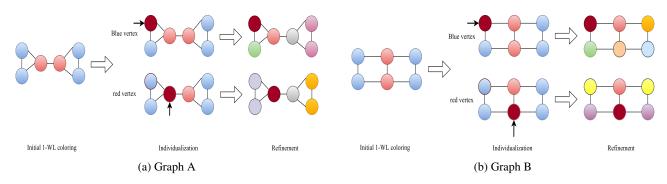

Figure 3: Two non-isomorphic graphs inducing different refinements for both color classes. Note that in both Graphs A and B, the initial 1-WL coloring forms orbit partition and hence refinements induced are same for all vertices of either blue or red vertex class in the graph.

Experiments

Graph classification on TUDataset bechmark   We evaluate GNN-IR on six datasets from the publicly available TUDataset bechmark, the graph classification benchmark suite (Kersting et al. 2016; Yanardag and Vishwanathan 2015). The datasets are broadly in domains of social media (IMDB-BINARY, IMDB-MULTI, COLLAB) and chemical science (NCI1, PROTEINS, MUTAG). We use GIN as our base GNN convolution operator and sum/mean pooling for readout.

Experimental setup

The evaluation procedures on TUDataset vary considerably in the recent literature on GNNs. This has given rise to different numbers for the same models. One evaluation setup (Xu et al. 2018) is to split the datasets into training and validation sets in the ratio of 90:10. Thereafter, take the mean of the validation curves across 10 folds and report mean/std of the best epoch of the mean validation curve. This evaluation protocol is non-standard but is used because of the small size of datasets. However, recently Errica et al. (2019) did a fair comparison of the prominent GNN models. They performed 10-fold standard 90:10 train-test split and report the average accuracy on the test set while model selection is done on a separate 10% split on the training set. We adopt this standard method of evaluation and report the accuracies in 10-fold experiment as described in Errica et al. (2019) and Nikolentzos and Vazirgiannis (2020).

Note that we are only comparing with neural GNN models and do not claim state-of-the-art score on these datasets. Kernel versions like (Du et al. 2019; Morris, Rattan, and Mutzel 2020) usually perform better than their neural counterparts on these particular datasets. In fact, the higher-order Local k-WL models in kernel versions have the state-of-the-art score for most of the TUDatasets. (Morris, Rattan, and Mutzel 2020) show results of only kernel versions of "Local k-WL" models on TUDatasets and not neural versions. Nonetheless, since we are comparing with higher-order k-WL GNNs in our experiments, we report results of 3-WL-GNN with this setup as well.

For our experiments, we compare the proposed GNN-IR model against the following GNN baselines : DGCNN (Zhang et al. 2018), DiffPool (Ying et al. 2018), GIN (Xu et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017a) and RWNN (Nikolentzos and Vazirgiannis 2020) and the higher-order 3-WL-GNN (Morris et al. 2019). Results for these baselines are as reported in (Nikolentzos and Vazirgiannis 2020) and we use the code of 1-2-3-GNN (Morris et al. 2019) to generate results for 3-WL-GNN with this setup.

Data set	IMDB-B	IMDB-M	COLLAB	NCI1	PROTEINS	MUTAG
DGCNN	69.2±3.0	45.6±3.4	71.2±1.9	76.4±1.7	72.9±3.5	84.0±6.7
Diffpool	68.4±3.3	45.6±3.4	68.9±2.0	76.9±1.9	73.7±3.5	79.8±7.1
GIN*	71.2±3.9	48.5±3.3	75.6±2.3	80.0±1.4	73.3±4.0	84.7±6.7
GRAPHSAGE	68.8±4.5	47.6±3.5	73.9±1.7	76.0±1.8	73.0±4.5	83.6±9.6
RWNN	70.8±4.8	48.8±2.9	71.9±2.5	73.9±1.3	74.7±3.3	**89.2**±4.3
3-WL-GNN	71.5±3.6	**50.4**±4.5	OOM	75.7±3.4	74.9±5.4	86.2±7.0
GNN-IR	**73.7**±4.3	50.1±3.8	**77.7**±3.1	**80.7**±2.2	**75.5**±2.2	88.7±6.9

Table 7: Graph classification on TUDataset. Our convolution operator is GINconv. OOM is Out of Memory.

Data set	IMDB-B	IMDB-M	COLLAB
DGCNN	53.3±5.0	38.6±2.2	57.4±1.9
Diffpool	68.3±6.1	45.1±3.2	67.7±1.9
GIN	66.8±3.9	42.2±4.6	**75.9**±1.9
GRAPHSAGE	69.9±4.6	47.2±3.6	71.6±1.5
GNN-IR	**71.9**±4.5	**48.4**±5.5	71.8±2.1

Table 8: Graph classification on social media datasets without node-degree as features.

Without node-degree features

The social media datasets do not come with node attributes and are initialized with node-degrees as node features. In principle, since node-degrees can be computed by 1-WL message passing, the presence of node-degree features should not affect the performance of GNN models. However, in practice GNN models perform poorly without node-degree features. This was shown
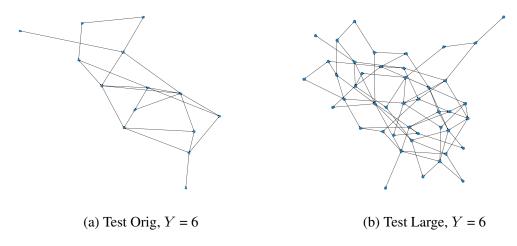
(a) Test Orig, $Y = 6$ (b) Test Large, $Y = 6$

Figure 4: Graphs with $Y = 6$ triangles from the TRIANGLES dataset.

in the results of (Errica et al. 2019). In order to assess the GNN-IR's robustness to the presence of node-degree features, we report additional results without node-degree features as well. In this case, we initialize node features with constant values.

Results

Table 7 shows the accuracy of the GNN models on graph classification in TUDataset. Clearly, GNN-IR scores are competitive in all the datasets. GNN-IR outperforms higher-order 3-WL-GNN by significant margin in IMDB-B, NCI1 datasets while in other datasets increase of 1-2% points can be seen. Compared to all other baselines, GNN-IR achieves the best score in 4 out of 6 datasets and is close to the best performing in other 2 datasets. Note the results in comparison to GIN, which is the base convolution operator used in GNN-IR. GNN-IR's clear edge over GIN shows that the improvement is coming from the *individualization and refinement* mechanism of GNN-IR.

Furthermore, the difference is clearer in Table 8, which shows the accuracy of the models on graphs without node-degree features. With the exception of COLLAB dataset, accuracy of GNN-IR does not decrease significantly in both IMDB datasets. This suggests that the GNN-IR can capture structural information from the graphs significantly better than the 1-WL equivalent GNN models. Since, GNN-IR works by breaking the symmetry between nodes, it is more robust to the absence of node-degree features.

Experimental details In this section, we give further details of experiments conducted. The code was written in Pytorch Geometric (Fey and Lenssen 2019) library and the experiments were performed on a GeForce RTX 2080Ti GPU card. Hyperparameter tuning was done on a separate validation set formed from the training set on all the datasets. For a fair comparison, we used implementations of Morris, Rattan, and Mutzel (2020) for ZINC10K, ALCHEMY10K, TUDataset bechmark and Pytorch Geometric example implementations for the rest of the datasets.

For all the datasets, our message passing model consists of a convolution operator for message function and a GRU for update following the MPNN implementation of Gilmer et al. (2017). The specific convolution operator used is given in Table 9 for each dataset. Additionally, we share the Conv-GRU parameters across the IR layers as this architecture scales well for deeper layers without increasing the parameter load and does not lose performance as well. Also, it can be said that the improvement shown in model's performance across datasets is not because of using more parameters. In each IR layer, GNN is run for 3 steps. In the set aggregators, we use 1-hidden layer MLP before sum-pooling the vectors of the set. And finally, in datasets with edge features, we treat edges as variables and readout from both node and edge variables before the final fully connected layer.

We use cross entropy and mean absolute error (mae) loss functions for graph classification and regression respectively. We report dataset statistics and all the hyperparameters which were used for the results in Table 9. These hyperparameters were chosen with a separate validation set for each dataset. Below we describe details of all datasets and the splits used in the experiments.

Datasets

TRIANGLES:

The dataset TRIANGLES (Knyazev, Taylor, and Amer 2019) consists of 45000 graphs with the task of counting the number of triangles in the graph. The number of classes is 10. We use node degrees as node features. The data splits are as follows, training (30000), validation (5000), test-original (5000) and test-large (5000). Except test-large, all data splits have smaller graphs with $N < 25$ nodes. The test-large set has $25 < N < 100$ nodes and tests the generalization ability of the model. Figure 4 shows example graphs from the two test sets where both have same number of triangles, $Y = 6$. It can be seen in Table 1 that

Table 9: Dataset statistics and hyperparameters used for all the datasets

Dataset	TRIANGLES	CSL	ZINC10K	ALCHEMY10K	QM9	COLLAB	IMDB-B	IMDB-M	NCI1	PROTEINS	MUTAG
#graphs	45000	150	12000	12000	130831	5000	1000	1500	4,110	1113	188
Node feat	Yes	No	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Edge feat	No	No	Yes	Yes	Yes	No	No	No	No	No	No
batch size	60	16	128	128	64	64	64	64	64	64	64
hidden layer size	64	64	75	75	64	64	64	64	64	64	64
epochs	300	300	200	200	200	200	200	200	150	150	50
start lr	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
decay rate	0.5	0.5	0.5	0.5	0.7	0.7	0.7	0.7	0.7	0.7	0.7
decay steps	–	–	–	–	–	50	50	50	–	–	–
patience	15	15	15	15	5	–	–	–	15	15	10
#IR layers L	–	–	2	2	2	3	3	2	2	2	2
width k	–	–	2	4	4	4	4	2	4	4	2
Conv Operator	GIN	GIN	PNAConv	PNAConv	NNConv	GIN	GIN	GIN	GIN	GIN	GIN

(a) $\mathcal{G}_{skip}(41, 4)$ (b) $\mathcal{G}_{skip}(41, 11)$
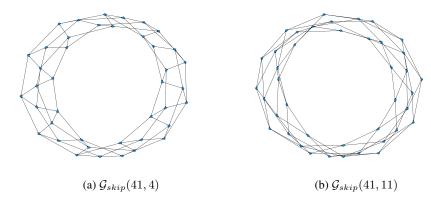
Figure 5: Non-isomorphic 4-regular graphs from CSL dataset.

there is significant improvement in the Test-large set with increasing IR layers and width of each IR-layer suggesting better generalization to larger graphs by our model.

Circulant Skip Links (CSL) :

The Circulant Skip Link dataset is a graph classification dataset introduced in (Murphy et al. 2019) in order to test the expressive power of GNNs. A CSL graph $\mathcal{G}_{skip}(M, R)$ is a 4-regular graph with $\{0, 1, \ldots M-1\}$ vertices and edge between pairs of vertices which are R distance away in a cyclical form. The graphs are from 10 classes representing $R \in \{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}$. Figure 5 shows two non-isomorphic graphs from the CSL dataset with 11 nodes $R = 4, 11$. The dataset has 150 graphs with each class having 15 graphs. Following Murphy et al. (2019), we use a 5-fold cross validation split, with each split having train, validation and test data in the ratio of $3 : 1 : 1$. We use the validation split to decay the learning rate and for model selection.

ZINC10K:

ZINC10K dataset (Jin, Barzilay, and Jaakkola 2018; Dwivedi et al. 2020) contains 12000 Zinc molecules out of which 10000 are for training and 1000 each for validation and test sets. The dataset comes with constrained solubility values associated with the molecules and the task is to regress over these values. The performance measure is the mean absolute error (MAE) for regression on each graph. We use the baseline results as in Morris, Rattan, and Mutzel (2020) and add other recent better performing model PNA (Corso et al. 2020) for a more thorough comparison. We use L1 loss and report absolute error on the test set.

ALCHEMY10K:

ALCHEMY10K (Chen et al. 2019a) is a recently released graph regression dataset. The task to regress over 12 values related to quantum chemistry. The targets are same as in QM9 but the molecules come with more heavy atoms. We use the same smaller version of the dataset as used in Morris, Rattan, and Mutzel (2020) with 10000 training, 1000 validation and 1000 test molecules which are randomly sampled from the original full dataset. Like in ZINC10K, we add PNA (Corso et al. 2020) for a more recent comparison. For this, we ran PNA on the same dataset with code provided officially by Pytorch-Geometric library. We use L1 loss and report MAE on the regression targets.

QM9:

QM9 (Ruddigkeit et al. 2012; Ramakrishnan et al. 2014) is a prominent large scale dataset in the domain of quantum chemistry. The dataset contains more than 130K drug-like molecules with sizes ranging from 4-29 atoms per molecule. Each molecule may contain up to 9 heavy (non-Hydrogen) atoms. The task is to regress on 12 quantum-mechanical properties associated with each
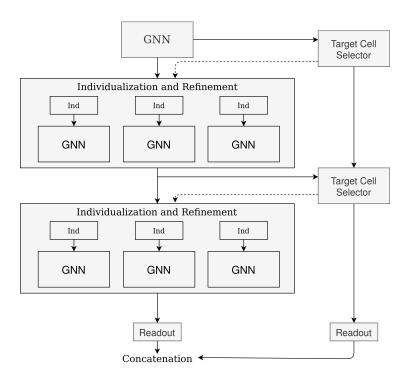
Figure 6: Model architecture of GNN-IR

molecule. In our experiments, we follow Gilmer et al. (2017) MPNN architecture in node and edge features. We further add edge variables for message passing and readout from both node and edge variables. For edge message passing, we simply concatenate the two node hidden vectors and pass it through an MLP and update the edge variables before finally reading out from both node and edge variables. We follow the same random split of $8:1:1$ for training, validation and testing as in Morris et al. (2019); Morris, Rattan, and Mutzel (2020). We compare with the recent GNN models as in Morris, Rattan, and Mutzel (2020) which covers prominent GNN models tested on this dataset. Note that as in Morris, Rattan, and Mutzel (2020), we are not comparing with Schnet (Schütt et al. 2018), Physnet (Unke and Meuwly 2019) and Dimenet (Klicpera, Groß, and Günnemann 2020) which incorporate physical knowledge in the modeling of message passing. All compared models are general GNN models. We train with L1 loss and report MAE on the targets.