Improving Graph Neural Network with Learnable Permutation Pooling

Yu Jin
Department of Electrical and Computer Engineering
University of Maryland, College Park
yuj@umd.edu

Joseph F. JaJa
Department of Electrical and Computer Engineering
University of Maryland, College Park
josephj@umd.edu

***Abstract*—Graph neural networks (GNN) have achieved great success in various graph-related applications. Most existing graph neural network models follow the message-passing neural network (MPNN) paradigm where the graph pooling function forms a critical component that directly determines the model effectiveness. In this paper, we propose PermPool, a new graph pooling function that provably improves the GNN model expressiveness. The method is based on the insight that the distribution of node permuations, when defined properly, forms characteristic encoding of graphs. We propose to express graph representations as the expectation of node permutations with a general pooling function. We show that the graph representation remains invariant to node-reordering and has strong expressive power than MPNN models. In addition, we propose novel permutation modeling and sampling techniques that integrate PermPool into the differentiable neural network models. Empirical results show that our method outperformed other pooling methods in benchmark graph classification tasks.**

***Index Terms*—Graph neural network, Permutation modeling, Pooling Function, Positional Encoding.**

I. Introduction

Graph neural network (GNN) models emerge as a powerful tool to solve graph related problems such as node classification, link prediction and graph classification [1]–[6]. Most of GNN models follow a message-passing paradigm where node representations are formed by iteratively aggregating information from neighboring nodes, followed by a graph pooling function to generate graph representations [1], [4], [5], [7]. Graph pooling functions, which generate graph representations from node representations, are one of the key components that determine the effectiveness of GNN models. There have been extensive studies on the graph pooling functions. For example, Xu et al. proposed to use multi-set functions for graph pooling [7], [8]. Gilmer et al. proposed a general message-passing scheme utilizing Set2Set to obtain graph representations [5], [9]. Corso et al. proposed to combine different aggregators to form more expressive graph representations [10]. Graph pooling functions are usually expected to be *permutation invariant*, that is, graph representations remain the same for isomorphic graphs. To satisfy the constraint, most graph pooling functions are formulated with simple permutation-invariant function forms, such as MEAN, SUM and MAX, which hampers the flexibility to represent a rich class of graph functions [7], [11]–[13]. Moreover, pooling functions formulated from simple functions suffer from the lack of expressiveness which cannot distinguish between even simple graphs.

One of the most common methods to overcome the limitation of expressiveness is to use Positional Encodings (PE) which equip the graph nodes with additional features that improve the model expressiveness. Sato et al. and Abboud et al. proposed to use random node embeddings [14], [15]. But the problem with random node embeddings is the lack of equivariance and invariance which are key properties of graph pooling functions. Bouritsas et al. used subgraph counts and Kreuzer et al. used the Laplacian eigenvectors as additional features to improve the performance [16], [17]. These positional encodings are usually predefined before the learning task which reduce the adaptativity to specific graph problems. Moreover, position encodings mostly only capture partial of the local information with little consideration for the global graph structure information.

In this work, we propose **PermPool**, a new graph pooling function that provably improves the GNN expressiveness. Our method is based on the key insight that the distribution of node permutations, when defined properly, forms the characteristic representations for graphs. More formally, our method starts by deriving the distributions of node permutations, which could be heuristic or parameterized with graph information. Given the distribution of node permutations, we are able to formulate the graph representation as the expected value of a general pooling function with the permutation equipped as positional embeddings. The general pooling functions are dependent on the specific Combined with the advanced permutation sampling techniques, we can incoporate the differentiable permutation modeling with the GNN models which could be further learned with the specific graph problems . The overall architecture is depicted in Figure 1.

One closely related work is Murphy et al.'s study where the authors proposed *Janossy Pooling*, a similar method that defines the graph representation as average of the pooling function applied on the node representations and all possible permutations [18], [19]. However, their method assumed a uniform distribution across all possible node permutations which is computational expensive and lack the flexibility to consider the underlying graph information. **PermPool** provides additional flexibility to define a wide range of permutation distributions and therefore more expressive pooling functions.

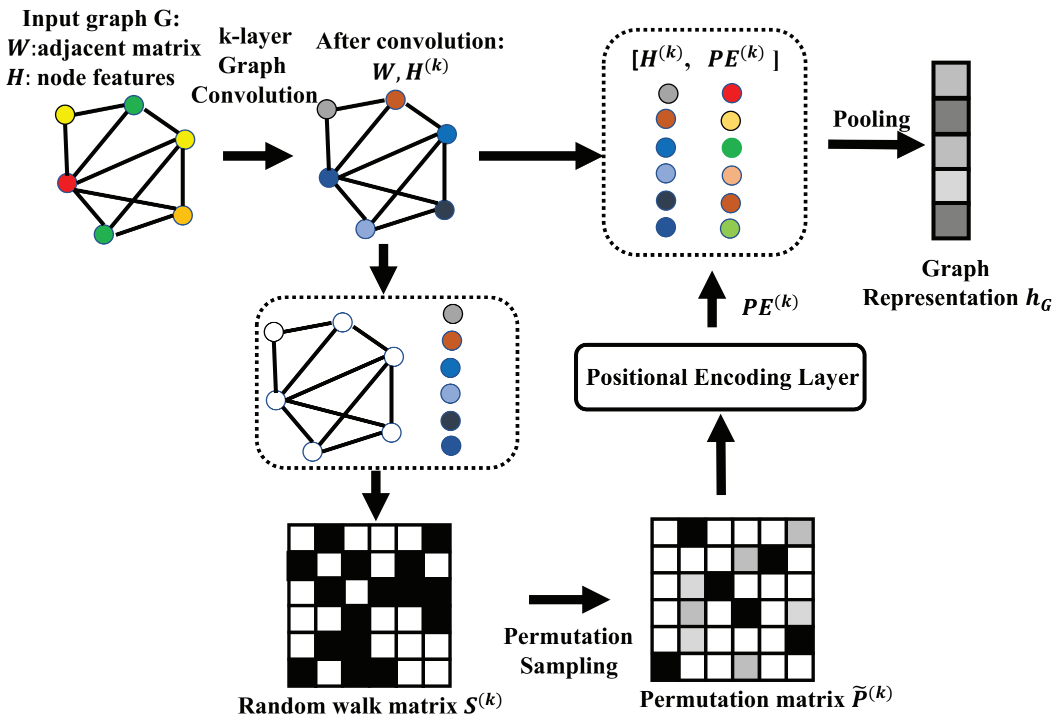
Input graph G:
W:adjacent matrix
H: node features
k-layer Graph Convolution
After convolution: $W, H^{(k)}$
$[H^{(k)}, PE^{(k)}]$
Pooling
Graph Representation h_G
$PE^{(k)}$
Positional Encoding Layer
Permutation Sampling
Random walk matrix $S^{(k)}$
Permutation matrix $\widetilde{P}^{(k)}$

Fig. 1. The overall architecture of the permutation pooling framework.

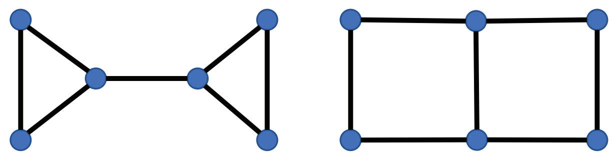

Fig. 2. One example pair of graphs that cannot be distinguished by MPNN models.

Moreover, in practice, the distribution of permuatations learned by our model can have less computational complexity than enumerating all possible $n!$ permutations.

In summary, our work has the following contributions,

- We provide a flexible framework to represent an expressive and invariant graph pooling function
- We propose novel permutation modeling and sampling techniques that integrate the permutation pooling into the graph neural network model.
- Empirical results show that **PermPool** achieves outstanding performance in benchmark graph classification tasks.

II. Related Work

Graph Pooling There have been extensive studies on the design of graph pooling functions. Zhang et al. proposed *SortPool* that first sorts graph nodes with the node features and apply convolutional neural networks on the sorted graph features [11]. Ying et al. proposed *DiffPool* that integrates the differentiable pooling function in the hierarchical neural network model [20]. Gao and Ji proposed a *top-k* graph pooling function by downsampling the graph nodes with a U-net-like neural network architecture [12]. Most graph pooling functions only aggregate node features without considering the graph structures. Therefore, positional encoding techniques are introduced to improve the expressiveness.

Positional Encoding Neural networks with positional encodings have achieved great success in image and langauge tasks [21], [22]. However, there are no predefined positional embeddings for graphs. Researchers proposed techniques such as random embedding or subgraph counts used as the positional embedding [14]–[16]. Due to the challenge of defining node positions for graphs, the positional encodings don't fully capture the whole graph information.

Permutation Modeling Our work is closely related to the recent work on permutation modeling. Adams et al. proposed to use doubly stochastic matrices to model marginals of the distribution over permutation matrices that is further used to characterize the expectation of the ranking objective [23]. Similarly, Mena et al. proposed a *Sinkhorn Network* model that learns permutations as solutions of the matching problem, approximated by doubly stochastic matrices [24]. Grover proposed to model permutations with stochastic optimization via continuous relaxation [25]. Most of permutation modeling targets on images reconstruction and number sorting problems.

Our work is the first to introduce the permutation modeling to the graph learning domain.

III. Preliminaries

A. Notations

Graph A graph is represented as $\mathcal{G} = (\mathcal{V}, \boldsymbol{W}, \boldsymbol{H})$, with $\mathcal{V}$ as the set of graph nodes with $n = |\mathcal{V}|$, $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ as the adjacent matrix and $\boldsymbol{H} \in \mathbb{R}^{n \times d}$ as the node representations. We denote by $v_i \in \mathcal{V}$ as the node indexed at i and $\mathcal{N}_s(v_i)$ as the set of neighbors of v_i within s hops (We denote $\mathcal{N}(v_i) = \mathcal{N}_1(v_i)$). We use $\boldsymbol{h}_i$ as the node representation of node v_i and $\boldsymbol{h}_\mathcal{G}$ as the graph-level representation.

Permutation We denote a permutation over integers from 1 to n as a list of node indices $\boldsymbol{\pi} = \{v_1, v_2, ..., v_n\}$. The corresponding permutation matrix is denoted as $\boldsymbol{P}_{\boldsymbol{\pi}}$. We use $\boldsymbol{\Pi}$ to denote the set of all possible permutations with $|\boldsymbol{\Pi}| = n!$. $\boldsymbol{\pi}^{-1}$ is denoted as the inverse permutation of $\boldsymbol{\pi}$. A graph permutated with $\boldsymbol{\pi}$ is denoted as $\mathcal{G}_{\boldsymbol{\pi}} = (\mathcal{V}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}, \boldsymbol{H}_{\boldsymbol{\pi}})$ where the node indices, weight matrix and feature matrix are permutated over $\boldsymbol{\pi}$ denoted as $\mathcal{V}_{\boldsymbol{\pi}}$, $\boldsymbol{W}_{\boldsymbol{\pi}} = \boldsymbol{P}_{\boldsymbol{\pi}} \boldsymbol{W} \boldsymbol{P}_{\boldsymbol{\pi}}^T$ and $\boldsymbol{H}_{\boldsymbol{\pi}} = \boldsymbol{P}_{\boldsymbol{\pi}} \boldsymbol{H}$.

B. Message Passing Graph Neural Network

Starting with the initial node features $\boldsymbol{H}^{(0)} = \boldsymbol{H}$, the message passing graph neural network iteratively updates node representations by aggregating information from neighboring nodes. The following functions characterize the graph convolutional process [5], [7], [26],

$$\begin{aligned} \boldsymbol{h}_v^{(i)} &= \textbf{Aggregate}^{(i)}(\boldsymbol{h}_v^{(i-1)}, \boldsymbol{m}_v^{(i)}), \\ \boldsymbol{m}_v^{(i)} &= \textbf{Msg}^{(i)}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}(v)\}) \end{aligned} \tag{1}$$

where $\boldsymbol{h}_u^i$ is the representation of node u at iteration i and **Msg** is the message function that aggregates neighborhood information. The graph representation $\boldsymbol{h}_\mathcal{G}$ is obtained by applying the pooling function **Pool** on the corresponding node representations as,

$$\boldsymbol{h}_\mathcal{G} = \textbf{Readout}(\boldsymbol{h}_v^{(k)}, v \in \mathcal{V}) \tag{2}$$

We use $\textbf{GNN}_k$ to denote graph neural network models with k graph convolutional layers.

For most graph neural network models, the **Readout** function only depends on the node representations without considering graph topological information, which hampers the expressiveness of graph neural network models.

In this work, we explicitly denote the graph pooling functions as $\textbf{PermPool} : (\mathbb{R}^{n \times d}, \mathbb{R}^{n \times n}) \to \mathbb{R}^d$ which takes variable-sized node representations $\boldsymbol{H}$ and adjacent matrix $\boldsymbol{W}$ as inputs and outputs a fixed-sized graph representation $\boldsymbol{h}_\mathcal{G} \in \mathbb{R}^d$.

IV. Graph Neural Network with Learnable Permutation Pooling

A. Main Framework

We represent the permutation pooling function as the expectation of a general pooling function over some permutation distribution. We denote the *permutation-invariant* pooling functions as $\textbf{PermPool} : (\mathbb{R}^{n \times d}, \mathbb{R}^{n \times n}) \to \mathbb{R}^d$ and permutation-dependent pooling functions as $\textbf{Pool} : (\mathbb{R}^{n \times d}, \mathbb{R}^{n \times n}, \boldsymbol{\pi}) \to \mathbb{R}^d$. Then the permutation pooling function is expressed as,

$$\begin{aligned} \textbf{PermPool}(\boldsymbol{H}, \boldsymbol{W}) &= \rho\left(\mathbb{E}_{\boldsymbol{\pi} \sim \boldsymbol{p_\theta}}[\textbf{Pool}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi})]\right) \\ &= \rho\left(\sum_{\boldsymbol{\pi} \in \Pi} \boldsymbol{p_\theta}(\boldsymbol{\pi} | \boldsymbol{H}, \boldsymbol{W}) \textbf{Pool}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi})\right) \end{aligned} \tag{3}$$

where $\boldsymbol{p_\theta}(\boldsymbol{\pi} | \boldsymbol{H}, \boldsymbol{W})$ is the permutation distribution with parameters $\boldsymbol{\theta}$, $\textbf{Pool}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi})$ is a general pooling function defined on the node features and adjacent matrix with the node permutation $\boldsymbol{\pi}$ used as positional encodings. ρ is a non-linear function such as **MLP**. **PermPool** is expected to be permutation-invariant as follows,

$$\textbf{PermPool}(\boldsymbol{H}, \boldsymbol{W}) = \textbf{PermPool}(\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}), \forall \boldsymbol{\pi} \in \boldsymbol{\Pi}. \tag{4}$$

The permutation pooling functions provide the flexibility to represent a broad class of permutation-invariant pooling functions with general pooling functions. One example is that when $\boldsymbol{p_\theta}(\boldsymbol{\pi} | \boldsymbol{H}, \boldsymbol{W})$ is a uniform distribution over all possible $n!$ permutations, the pooling function **PermPool** becomes the *Janossy pooling* function proposed by Murphy et al. as follows [18],

$$\textbf{PermPool}(\boldsymbol{H}, \boldsymbol{W}) = \rho\left(\frac{1}{n!} \sum_{\boldsymbol{\pi} \in \Pi} \textbf{Pool}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi})\right) \tag{5}$$

Beyond the uniform distribution, the following proposition states the general condition of the permutation distribution that guarantees the permutation-invariance of the pooling function,

Proposition 1 (Permutation Invariance)**.** ***PermPool***$(\boldsymbol{H}, \boldsymbol{W})$ *is permutation invariant if the probability distribution satisfies*

$$\boldsymbol{p_\theta}(\boldsymbol{\pi}\boldsymbol{\pi}' | \boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}) = \boldsymbol{p_\theta}(\boldsymbol{\pi}' | \boldsymbol{H}, \boldsymbol{W}), \forall \boldsymbol{\pi}, \boldsymbol{\pi}' \in \boldsymbol{\Pi}. \tag{6}$$

and

$$\textbf{\textit{Pool}}(\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}, \boldsymbol{\pi}') = \textbf{\textit{Pool}}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi}\boldsymbol{\pi}') \tag{7}$$

In the following, we present the permutation modeling and sampling techniques to formullate the permutation pooling function.

B. Permutation Modeling

Permutations over graph nodes are generally considered as the non-repeating node paths sampled from graph traversal algorithms. Random-walk based traversal algorithms have been used to sample node neighbors to learn effective node representations [27]–[29]. However, most graph traversal algorithms generate node paths that only depend on graph structures. In

Algorithm 1 Discrete Permutation Modeling from Doubly Stochastic Matrix

```
Input: doubly stochastic matrix $\boldsymbol{S} \in \mathbb{R}^{n\times n}$
Output: $\boldsymbol{\pi} = \{v_1, v_2, ..., v_n\}$ and the associated probability $\boldsymbol{p}(\boldsymbol{\pi})$

$v_1 \sim$ Uniform$(1, n)$, unvisited_list = [1, 2, ..., n]
$\boldsymbol{S}_{:,v_1} = 0$, remove $v_1$ from unvisited_list
for i = 2, ..., n do
    if any$(\boldsymbol{S}_{v_{i-1},:}) > 0$ then
        $v_i \sim$ DiscreteSample$(\boldsymbol{S}_{v_{i-1},:})$
    else
        $v_i \sim$ Uniform$(unvisited\_list)$
    end if
    $\boldsymbol{S}_{:,v_i} = 0$, remove $v_i$ from unvisited_list
end for
Return $\boldsymbol{\pi} = \{v_1, v_2, ..., v_n\}$ and its associated probability $\boldsymbol{p}(\boldsymbol{\pi})$
```

addition, it is nontrivial to integrate the node path sampling into the neural network framework.

In this work, we model the permutation distribution through a *symmetric real-value doubly-stochastic matrix* (DSM). The symmetric doubly-stochastic matrix $\boldsymbol{S} \in \mathbb{R}^{n\times n}$ satisfies the following,

$$\boldsymbol{S}_{i,j} \geq 0 \ \forall i,j, \sum_{i=1}^{n} \boldsymbol{S}_{i,j} = 1 \ \forall j, \sum_{j=1}^{n} \boldsymbol{S}_{i,j} = 1 \ \forall i, \boldsymbol{S} = \boldsymbol{S}^T \tag{8}$$

DSMs can be constructed from any real-value symmetric matrices in a differentiable form that provides the convenience to be integrated into an end-to-end neural network framework. Given any real-value symmetric matrix $\boldsymbol{X} \in \mathbb{R}^{n\times n}$, we can apply a simplified *Sinkhorn normalization* that transforms any real-value matrix to a DSM form as,

$$\boldsymbol{S} = \boldsymbol{X} \oslash (\boldsymbol{X}\mathbf{1}_n\mathbf{1}_n^\top) \tag{9}$$

with $\oslash$ denoting the element-wise division and $\mathbf{1}_N$ a column vector of ones. Since $\boldsymbol{X}$ is symmetric, the $\boldsymbol{S}$ can also be defined as

$$\boldsymbol{S} = \boldsymbol{X} \oslash (\mathbf{1}_n\mathbf{1}_n^\top\boldsymbol{X}) \tag{10}$$

The DSM $\boldsymbol{S}$ forms the key matrix based on which the permutation distribution is defined. Each element $\boldsymbol{S}_{i,j}$ represents the random walk probability from node i to node j. One example is to directly use the adjacent matrix $\boldsymbol{W}$ to define $\boldsymbol{S}$ which becomes the traditional random walk matrix for graphs.

More broadly, $\boldsymbol{S}$ constructed from any symmetric matrix $\boldsymbol{X}$ provides the flexibility to integrate rich graph information. For example, we can define $\boldsymbol{X}$ combining both the graph structure and the node features as,

$$\boldsymbol{X} = \boldsymbol{W} + \lambda\boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^T \tag{11}$$

where $\boldsymbol{D} \in \mathbb{R}^{d\times d}$ and $\lambda \in \mathbb{R}$ are learnable parameters.

Algorithm 2 Continuous Permutation Sampling from Doubly Stochastic Matrix

```
Input: doubly stochastic matrix $\boldsymbol{S} \in \mathbb{R}^{n\times n}$, temperature parameter $\tau$, threshold $\epsilon$
Output: $\widetilde{\boldsymbol{P}} \in \mathbb{R}^{n\times n}$ that approximates the permutation matrix

$\widetilde{\boldsymbol{P}}_{:,1} \sim$ Gumbel_max(Uniform$(1, n), \tau)$,
$\boldsymbol{S}_{:,j} = \boldsymbol{S}_{:,} \odot (\mathbf{1} - \widetilde{\boldsymbol{P}}_{:,1}), \forall j = 1, ..., n$
$\boldsymbol{v}_{\text{unvisited}} = \mathbf{1} - \widetilde{P}_{:,1}$
for i = 2, ..., n do
    $\boldsymbol{p}_i = \boldsymbol{S} \cdot \widetilde{P}_{:,i-1}$
    if sum$(\boldsymbol{p}_i) > \epsilon$ then
        $\widetilde{P}_{:,i} \sim$ Gumbel_max$(\boldsymbol{p}_i, \tau)$
    else
        $\widetilde{P}_{:,i} \sim$ Gumbel_max$(\boldsymbol{v}_{\text{unvisited}}, \tau)$
    end if
    $\boldsymbol{S}_{:,j} = \boldsymbol{S}_{:,} \odot (\mathbf{1} - \widetilde{\boldsymbol{P}}_{:,i}), \forall j = 1, ..., n$
    $v_{\text{unvisited}} = v_{\text{unvisited}} - \widetilde{P}_{:,1}$
end for
Return the permutation matrix $\widetilde{\boldsymbol{P}}$
```

We formally define the permutation modeling algorithm in Algorithm 1. The algorithm computes the probability for each possible permutation. However, in practice, it is intractable to enumerate all possible permutations and the associate probabilities. In addition, the sampling method in Algorithm 1 is discrete which is challenging to be integrated into an end-to-end differentiable graph neural networks. Therefore, we introduce the continuous permutation sampling techniques to obtain the approximation of the distribution of node permutations.

C. Continuous Permutation Sampling

Recently, Jang et al. and Maddison et al. introduced Gumbel-max tricks to sample discrete elements from categorical distributions [30]–[33]. The Gumbel-max trick is able to recast the sampling problem as an optimization problem with re-parametrization. The major benefit of Gumbel tricks is that the samples are a continuous approximation of the categorical samples where the parameter gradients can be easily computed via the re-parameterization trick.

We define $\boldsymbol{z} \in \mathbb{R}^n$ as the vector consisting of i.i.d. elements sampled from the standard Gumbel distribution [1]. Given a categorical distribution $\boldsymbol{p} \in \mathbb{R}^n$ representing the probability, the perturbed input is given as,

$$\boldsymbol{y} = \text{argmax}(\log \boldsymbol{p}_i + \boldsymbol{z}_i) \tag{12}$$

Jang et al. and Maddison et al. show that the distribution of $\boldsymbol{y}$ follows the categorical distribution $\boldsymbol{y} \sim \text{Cat}(\boldsymbol{p})$ [30], [31]. Softmax function is often used as a continuous, differentiable

[1]The standard Gumbel distribution can be sampled as $z = -\log(-\log(u))$ where u is sampled from Uniform$(0, 1)$

approximator to argmax. Then the sample vector $\widetilde{\boldsymbol{y}}$ is represented as

$$\widetilde{\boldsymbol{y}} = \frac{\exp\left(\log \boldsymbol{p}_i + \boldsymbol{z}_i\right)/\tau}{\sum_{j=1}^{n} \exp\left(\log \boldsymbol{p}_j + \boldsymbol{z}_j\right)/\tau} \tag{13}$$

where τ is the temperature parameter determining the function smoothness.

Based on the Gumbel-max trick, we describe the continuous sampling algorithm in Algorithm 2. Different from the discrete permutation modeing in Algorithm 1, the continuous sampling outputs a single matrix $\widetilde{P} \in \mathbb{R}^{n\times n}$ that approximates the permutation matrix. The sampling process can be directly integrated in the end-to-end neural network structure.

D. Pooling Functions

The permutations obtained from the above sections can be used as Positional Encodings (PE) that provides additional information with the node features. Compared with previous definition of PE, permutation-based PE are inherently permutation invariant and encode rich graph information with the characteristic permutation distributions.

We use a *learnable* PE layer $\boldsymbol{E} \in \mathbb{R}^{n\times d}$ to encode the permutation matrix $\widetilde{P} \in \mathbb{R}^{n\times n}$ to $\widetilde{P}\boldsymbol{E}$ that forms the PE for the specific permutation. Then, we define the pooling function as the function of the concatenation of node features and positional embeddings $\boldsymbol{C} = \text{Concat}(\boldsymbol{H}, \widetilde{\boldsymbol{P}}\boldsymbol{E})$ as

$$\textbf{Pool}(\boldsymbol{H}, \boldsymbol{W}, \boldsymbol{\pi}) = f(\boldsymbol{C}) = g(\sum_{j\in\mathcal{V}} \Phi(\boldsymbol{C}_j)) \tag{14}$$

where g and Φ are transformations such as MLP. Zaheer et al. showed that any permutation-invariant functions can be represented in the above equation wit suitable transformations g and Φ. In addition, it is straightforward to check that the pooling function satisfying Equation 7.

E. Improved Expressive Power with permutation pooling

Previous work have established theoretical results on the expressive power of graph neural network. For example, Xu et al. proves that Graph Isomorphism Network (GIN) models are at least as powerful as the *Weisfeiler-Lehman test of isormphism* [7], [34]. Loukas et al. studied the capacity limit of general message-passing based graph neural models and categorized graph problems that graph neural network models can and cannot solve [26].

In this section, we discuss the expressive power of graph neural network with permutation pooling functions and compare with pooling functions used in GIN models [7]. The GIN pooling function is formulated as follows,

Theorem 1 (Theorem 2 in [8] and Lemma 5 in [7])**.** *Assuming the set of node features is countable, any permutation-invariant function GIN can be decomposed as*

$$\textbf{\textit{GIN}}(\boldsymbol{H}, \boldsymbol{W}) = \rho\left(\sum_{v\in\mathcal{V}} f(\boldsymbol{h}_v)\right) \tag{15}$$

for some functions ρ and f.

We define the expressive power of the model as the set of functions that the model can express. Assuming the function ρ is the same for permutation pooling and GIN, we have the following results,

Proposition 2. ***GNN*** *with the permutation pooling is at least as powerful as* ***GNN*** *with GIN pooling functions.*

In fact, compared with GIN pooling, the expressive power of permutation pooling functions is strengthened by the additive representation ability of the general pooling function **Pool** that capture relationships that are several hops away. We denote $\textbf{Pool}_s$ as the pooling function that captures node relationships of at most s hops away. The expressive power of GNN with permutation pooling is shown in the following proposition,

Proposition 3. $\textbf{\textit{GNN}}_k$ *with the permutation pooling with pooling function* $\textbf{\textit{Pool}}_s$ *is at least as powerful as* $\textbf{\textit{GNN}}_{k+s}$ *models with GIN pooling.*

With the pooling function $\textbf{Pool}_s$, the permutation pooling function is able to learn higher-order relationships in the **GNN** model.

However according to Theorem 1, GIN pooling functions can still approximate any permutation-invariant functions when ρ is chosen as a universal approximator function [7]. The following proposition states that when the function ρ has limited expressive power, the permutation pooling is strictly more powerful than GIN pooling functions,

Proposition 4. *If the function ρ is the identity function,* $\textbf{\textit{GNN}}_k$ *with the permutation pooling is strictly more powerful than* $\textbf{\textit{GNN}}_k$ *with GIN pooling functions.*

Proof. Proposition 2 already shows that for the same function ρ, permutation pooling is as least as powerful than GIN pooling. In the following, we show that there are graph functions that can be represented with permutation pooling but not with GIN pooling.

Consider the graph function as the weighted pairwise difference of node representations,

$$\mathcal{F}(\boldsymbol{H}, \boldsymbol{W}) = \sum_{u,v\in\mathcal{V}} \boldsymbol{W}(u,v)\left\|\boldsymbol{h}_u - \boldsymbol{h}_v\right\| \tag{16}$$

GIN pooling with the identical function ρ cannot represent the function as the function depends on pair-wise relationships of node representations as well as graph structure. For permutation pooling, we set the permutation distribution as the uniform distribution and the pooling function **Pool** is set as

$$\textbf{Pool}(\boldsymbol{H}_{\boldsymbol{\pi}}, \boldsymbol{W}_{\boldsymbol{\pi}}) = \boldsymbol{W}_{\boldsymbol{\pi}}(1,2)\left\|\boldsymbol{h}_1 - \boldsymbol{h}_2\right\|$$

Then, the corresponding permutation invariant pooling function **PermPool** can represents the function in 16. □

F. Training and Inference

We consider the graph classification problem: given a set of graph samples $\mathcal{D} = \{(\mathcal{G}_1, \boldsymbol{y}_1), (\mathcal{G}_2, \boldsymbol{y}_2), ..., (\mathcal{G}_N, \boldsymbol{y}_N)\}$ where

$\boldsymbol{y}_i \in \mathbb{Y}$ is the label of graph $\mathcal{G}_i$. The objective is to minimize the empirical loss as,

$$\begin{aligned}\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D};\boldsymbol{\theta}) &= \frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{y}_i, \mathbf{PermPool}(\boldsymbol{H}_i^{(k)}, \boldsymbol{W}_i^{(k)})) \\ &= \frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{y}_i, \rho(\mathbb{E}_{\boldsymbol{\pi}}[\mathbf{Pool}(\boldsymbol{H}_i^{(k)}, \boldsymbol{W}_i^{(k)}, \boldsymbol{\pi})]))\end{aligned} \tag{17}$$

where $\boldsymbol{\theta}$ denotes the set of parameters in the **GNN** model and L is the loss function such as the cross entropy loss.

Training The objective is a standard stochastic optimization with learnable parameters $\boldsymbol{\theta}$ and random variable $\boldsymbol{\pi}$. We use the stochastic gradient descent to find the optimal parameters θ^* [18], [35], [36]. At step t, we uniformly sample a mini-batch of example graphs as $\mathcal{B} = \{(\mathcal{G}'^{(1)}, \boldsymbol{y}'^{(1)}), (\mathcal{G}'^{(2)}, \boldsymbol{y}'^{(2)}), ..., (\mathcal{G}'^{(b)}, \boldsymbol{y}'^{(b)})\}$ from the training set and the gradient is computed as

$$\begin{aligned}\boldsymbol{g}_t &= \frac{1}{b}\sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}_i', \mathbf{GNN}_k(\mathcal{G}_i')) \\ &= \frac{1}{b}\sum_{i=1}^{b} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}_i, \rho(\mathbf{Pool}(\boldsymbol{H}_i^{(k)}, \boldsymbol{W}_i^{(k)}, \widetilde{\boldsymbol{\pi}_i})))\end{aligned} \tag{18}$$

where $\widetilde{\boldsymbol{\pi}_i}$ is the random permutation sampled with Gumbel-max techniques in Section IV-C.

We update the parameters by the following,

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \boldsymbol{g}_t$$

where $\eta_t \in (0,1)$ is the learning rate at step t with $\lim_{t\to\infty} \eta_t = 0$, $\sum \eta_t = \infty$ and $\sum \eta_t^2 < \infty$. Note that the algorithm is a standard stochastic optimization algorithms used in training neural networks.

The above stochastic gradient descent essentially optimizes the following modified objective with expectation outside the function L and ρ as [18],

$$\min_{\boldsymbol{\theta}} \tilde{\mathcal{L}}(\mathcal{D};\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{\pi}}[L(\boldsymbol{y}_i, \rho(\mathbf{Pool}(\boldsymbol{H}_i^{(k)}, \boldsymbol{W}_i^{(k)}, \boldsymbol{\pi})))] \tag{19}$$

When the loss function L is convex and ρ is the identity function, the modified objective function is an upper bound of the original loss function following Jensen's inequality [18]. We denote $\boldsymbol{\theta}^*$ as the optimal parameters of the optimization problem 19. Similar to the analysis of stochastic gradient descent, the parameters $\boldsymbol{\theta}_t$ converges to $\boldsymbol{\theta}^*$ with probability 1 under mild conditions [18], [37].

Inference Assuming $\boldsymbol{\theta}^*$ are the optimal parameters, the output $\hat{\boldsymbol{y}}$ is estimated as the average of the predicted sample outputs as,

$$\hat{\boldsymbol{y}} = \frac{1}{n'}\sum_{i=1}^{n'} \mathbf{GNN}_k(\mathcal{G}) = \frac{1}{n'}\sum_{i=1}^{n'} \rho\left(\mathbf{Pool}(\boldsymbol{H}^{(k)}, \boldsymbol{W}^{(k)}, \boldsymbol{\pi}_i)\right) \tag{20}$$

where n' is the number of inference samples with $\boldsymbol{\pi}_i$ as the random permutation sampled from distribution $\boldsymbol{p}_\theta(\boldsymbol{\pi}|\boldsymbol{H}, \boldsymbol{W})$.

V. Experiments

In the experiment, we evaluate the proposed method **PermPool** and compare with state-of-the art algorithms on real-world graph classification tasks.

A. Experiment Setup

Datasets The benchmark graph datasets contain 5 bioinformatics datasets and 5 social network datasets which are commonly used [7], [38]. The biological datasets (ENZYMES, NCI1, MUTAG, PROTEIN, PTC) contain graphs over a variety of biological applications, which are node-attributed with categorical graph labels. The social network datasets (COLLAB, IMDB-B, IMDB-M, REDDIT-B, REDDIT-M5K) contains graphs over social network applications with no explicit node features. The detailed dataset statistics are in the supplementary materials.

Baseline models We compare with the following state-of the art graph pooling functions used in graph neural network models: **DiffPool** [20], **SortPool** [11], **SAGPool** [13], **GIN** [7].

Model Configuration We use 10-fold cross validation and report the average classification accuracy and standard deviation. We use LSTM as the pooling function. The message passing layers are defined using GIN models. The detailed configuration are in the supplementary material.

B. Main Results

Table I and Table II summarize the classification results on the biological and social graph datasets. For most datasets, the proposed permutation pooling achieves superior or comparable performance compared with other state-of-the art pooling functions. The empirical results indicate that the permutation pooling functions are able to learn more effective graph representations for graph classification tasks.

C. Expressive Power of Pooling Functions

In the experiment, we compare the model performance of permutation pooling and **GIN** pooling under the same number of convolutional layers.

Figure 7 shows the plots of classification accuracy under the number of layers from 1 to 4 for the ENZYMES dataset. For both models, the classification accuracy improves as the number of the convolutional layer increases. However, under each setting, the permutation pooling constantly outperforms **GIN** pooling. In particular for $k = 1$, the **GIN** pooling function cannot learn much meaningful information due to its limited expressive power while permutation pooling can achieve 40% classification accuracy.

TABLE I
CLASSIFICATION ACCURACY ON BIOLOGICAL GRAPH DATASETS.

Datasets	MUTAG	ENZYMES	NCI1	PROTEINS	PTC
DiffPool [20]	85.5	62.53	-	76.25	58.45
SortPool [11]	85.8 $\pm$ 1.7	-	74.4 $\pm$ 0.5	75.5 $\pm$ 0.9	58.6 $\pm$ 2.5
SAGPool [13]	84.6 $\pm$ 8.3	45.3 $\pm$ 5.6	74.2 $\pm$ 1.2	70.0 $\pm$ 1.4	60.5 $\pm$ 4.1
GIN [7]	87.8 $\pm$ 5.3	52.2 $\pm$ 5.9	80.9 $\pm$ 2.0	**76.2 $\pm$ 2.8**	64.6 $\pm$ 7.0
PermPool	**88.5 $\pm$ 7.4**	**64.1 $\pm$ 6.1**	**81.7 $\pm$ 1.1**	74.8 $\pm$ 2.6	**65.1 $\pm$ 3.6**

TABLE II
CLASSIFICATION ACCURACY ON SOCIAL GRAPH DATASETS.

Datasets	COLLAB	IMDB-B	IMDB-M	RDT-B	RDT-M5K
DiffPool [20]	75.5	-	-	-	-
SortPool [11]	73.8 $\pm$ 0.5	70.0 $\pm$ 0.9	47.8 $\pm$ 0.8	-	-
GIN [7]	80.2 $\pm$ 1.9	**75.1 $\pm$ 5.1**	52.3 $\pm$ 2.8	92.4 $\pm$ 2.5	57.5 $\pm$ 1.5
PermPool	**80.6 $\pm$ 1.2**	74.3 $\pm$ 3.4	**52.8 $\pm$ 2.9**	**92.5 $\pm$ 2.7**	**58.2 $\pm$ 1.6**

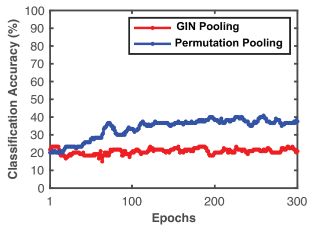

Fig. 3. $k = 1$

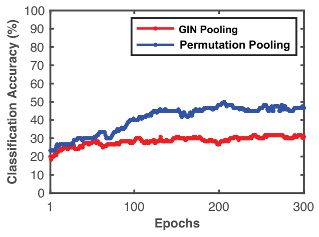

Fig. 4. $k = 2$

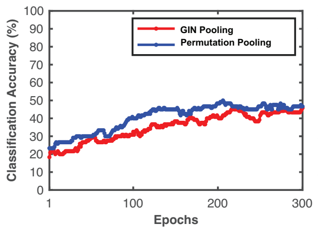

Fig. 5. $k = 3$

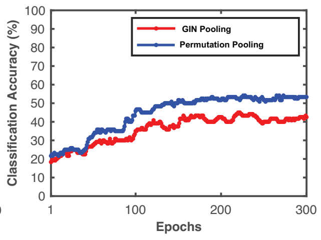

Fig. 6. $k = 4$

Fig. 7. Classification accuracy of GNN models under different numbers of convolutional layers.

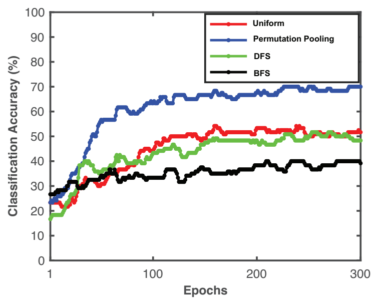

Fig. 8. Classification accuracy of GNN models under different permutation sampling strategies.

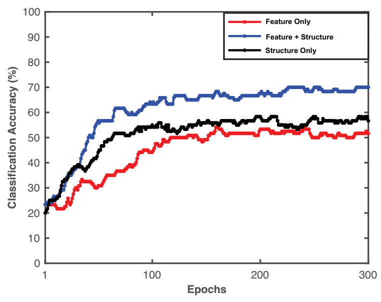

Fig. 9. Classification accuracy of GNN models under different definitions of random walk matrices.

D. Ablation Study: Permutation Sampling

We discuss the effect of permutation sampling on the model performance. We first compare with the following permutation sampling strategies used in previous work.

- **Uniform** permutations used in *Janossy Pooling* to sample node sequences [18], [19].
- **BFS/DFS** are used in Niepert et al.'s work to map graph nodes to sequences [29].

Figure 8 shows the classification performance for the above permutation sampling methods. It is shown that our proposed permutation sampling outperforms other sampling strategies by a large margin.

In the second experiment, we experiment with different input features that depends on different graph information as

- **Structure only** $\boldsymbol{X} = \boldsymbol{W}$.
- **Feature only** $\boldsymbol{X} = \boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^T$ where $\boldsymbol{D}$ is the learnable parameters.
- **Structure + Feature** $\boldsymbol{X} = \boldsymbol{W} + \boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^T$.

Figure 9 shows the classification results with permutation sampling based on the different definitions of DSMs. It is shown that the permutation distribution based on both graph structure and node representations performs the best compared with the definitions only relying on partial graph information. The experiment justifies our selection of input matrix in Equation 11 to approximate the permutation distributions.

VI. Conclusion

In this work, we propose a new graph pooling function that provably improves the performance of graph neural network models. The newly proposed permutation pooling is formulated as the expectation of a general pooling function over a distribution of node permutations. We propose novel ways to model the distribution of node permutations based on parameterized formulation of node features and graph structural information. With the advanced sampling techniques, we are able to build an end-to-end graph neural network that incorporates the permutation modeling and the model parameter learning. We provably show that GNN with permutation pooling is strictly more powerful than GNN with standard pooling functions. Empirical results shows that the proposed method is superior or comparable with other pooling functions on real-world graph classification tasks.

References

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *arXiv preprint arXiv:1611.08097*, 2016.

[3] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[4] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International conference on machine learning*, 2016, pp. 2702–2711.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *ICML*, 2017.

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

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

[8] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in neural information processing systems*, 2017, pp. 3391–3401.

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

[10] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 260–13 271, 2020.

[11] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] H. Gao and S. Ji, "Graph u-nets," *arXiv preprint arXiv:1905.05178*, 2019.

[13] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.

[14] R. Sato, M. Yamada, and H. Kashima, "Random features strengthen graph neural networks," in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 2021, pp. 333–341.

[15] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz, "The surprising power of graph neural networks with random node initialization," *arXiv preprint arXiv:2010.01179*, 2020.

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

[17] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[18] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," *arXiv preprint arXiv:1811.01900*, 2018.

[19] ——, "Relational pooling for graph representations," *arXiv preprint arXiv:1903.02541*, 2019.

[20] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018, pp. 4800–4810.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 6000–6010.

[22] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, "A survey on visual transformer," *arXiv e-prints*, pp. arXiv–2012, 2020.

[23] R. P. Adams and R. S. Zemel, "Ranking via sinkhorn propagation," *arXiv preprint arXiv:1106.1925*, 2011.

[24] G. Mena, D. Belanger, S. Linderman, and J. Snoek, "Learning latent permutations with gumbel-sinkhorn networks," *arXiv preprint arXiv:1802.08665*, 2018.

[25] A. Grover, E. Wang, A. Zweig, and S. Ermon, "Stochastic optimization of sorting networks via continuous relaxations," *arXiv preprint arXiv:1903.08850*, 2019.

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

[27] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

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

[29] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[30] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[31] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[32] M. Balog, N. Tripuraneni, Z. Ghahramani, and A. Weller, "Lost relatives of the gumbel trick," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 371–379.

[33] G. Papandreou and A. L. Yuille, "Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 193–200.

[34] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[35] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[36] L. Bottou and Y. L. Cun, "Large scale online learning," in *Advances in neural information processing systems*, 2004, pp. 217–224.

[37] A. L. Yuille, "The convergence of contrastive divergences," in *Advances in neural information processing systems*, 2005, pp. 1593–1600.

[38] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1365–1374.