Two-level Graph Neural Network

Xing Ai, Chengyu Sun, Zhihong Zhang*, and Edwin R Hancock, *Fellow, IEEE*

***Abstract*—Graph Neural Networks (GNNs) are recently proposed neural network structures for the processing of graph-structured data. Due to their employed neighbor aggregation strategy, existing GNNs focus on capturing node-level information and neglect high-level information. Existing GNNs therefore suffer from representational limitations caused by the Local Permutation Invariance (LPI) problem. To overcome these limitations and enrich the features captured by GNNs, we propose a novel GNN framework, referred to as the Two-level GNN (TL-GNN). This merges subgraph-level information with node-level information. Moreover, we provide a mathematical analysis of the LPI problem which demonstrates that subgraph-level information is beneficial to overcoming the problems associated with LPI. A subgraph counting method based on the dynamic programming algorithm is also proposed, and this has time complexity is $O(n^3)$, n is the number of nodes of a graph. Experiments show that TL-GNN outperforms existing GNNs and achieves state-of-the-art performance.**

***Index Terms*—Graph representation, Graph neural networks, Local permutation invariance, Attention mechanism.**

I. Introduction

GRAPH Neural Networks (GNNs) have attracted increasing interest in graph-structured data such as social networks, recommender systems, bioinformatics and combinatorial optimization. Scarselli et al. [1] first introduced the concept of the GNN by extending recursive neural networks. Veličković et al. [2] proposed the Graph Attention Network (GAT), which leverages masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions. Xu et al. [3] present a theoretical framework to analyze the representational capability of GNNs, and develop a simple neural architecture referred to as the Graph Isomorphism Network (GIN).

Although the existing neighborhood aggregation strategy used in GNNs is relatively efficient from the viewpoint of graph isomorphism classification, recent studies [4] [5] [6] show that such a procedure brings some inherent problems. Namely, most existing GNNs suffer from local permutation invariance (LPI), which leads them to confuse specific structures. In fact, invariance is very common in many learning tasks. Data can produce identical embeddings in a reduced low-dimensional space after symmetric transformations or rotations are applied [7], [8]. As for graph-structured data, Garg et al. [4] have found that existing GNNs have representational limitations caused by the translation of graph-structured data. To eliminate such an effect, Sato et al. [5] have exploited a local port ordering of nodes referred to as the Consistent Port Numbering GNN (CPNGNN). Moreover, Klicpera et al. [6] have proposed DimeNet, which is a directional message passing algorithm introduced in the context of molecular graphs. However, Garg V et al. [4] prove that all existing GNN variants have representational limits caused by LPI, and propose a novel graph-theoretic formalism.

In the meantime, studies show that complex networks can be succinctly described using graph substructures (also referred to as subgraphs, graphlets, or motifs). Subgraph methods have been well-studied and widely used in chemistry [9], biology [10], and social network graph tasks [11]. For example, specific patterns of atoms or modes of interaction can be discovered by identifying specific subgraph topologies. Bai et al. [12] propose a general subgraph-based training framework referred to as Ripple Walk Training (RWT). This can not only accelerate the training speed on large graphs but also solve problems associated with the memory bottleneck. Emily et al. [13] propose SUBGNN to propagate neural messages between the subgraph components and randomly sampled anchor patches. These methods extract node features and subgraph features separately. Moreover, they characterize only the number of different subgraphs, which ignore the learning of their representation.

For the sake of the above mentioned problems, we propose a novel model which merges subgraph-level information into the node-level representation. First, we merge subgraph-level information at the node-level to enrich the features. And secondly, we theoretically verify the model to demonstrate its performance with real-world datasets. The results show that our approach is significantly more effective than state-of-the-art baselines. Our main contributions are summarized as follows:

1. We propose a novel GNN approach, the Two-level GNN (TL-GNN), which captures both microscopic (small scale) and macroscopic (large scale) structural information simultaneously and thus enriches the representation of a graph.
2. We provide a mathematical definition and analysis of the effects of LPI on GNNs. Furthermore, we prove that subgraph-level information offers benefits in overcoming these limitations.
3. We verify our method on seven different benchmarks and a synthetic dataset. The results show that TL-GNN is more powerful than existing GNN's.
4. A subgraph counting method based on dynamic programming is also proposed. The time complexity and space complexity of this algorithm are $O(n^3)$ and $O(n^3)$ respectively, where n is the number of nodes in the graph.

Xing Ai and Chengyu Sun are with School of Informatics, Xiamen University, Xiamen, Fujian, China.
E-mail: 24320191152507@stu.xmu.edu.cn, 30920201153942@stu.xmu.edu.cn

Corresponding author: Zhihong Zhang is with School of Informatics, Xiamen University, Xiamen, Fujian, China.
E-mail: zhihong@xmu.edu.cn

Edwin R. Hancock is with University of York, York, UK.
E-mail: edwin.hancock@york.ac.uk

The remainder of this paper is organized as follows. Section. II provides an overview of the related work. Section. III introduces the proposed method, including theoretically proving the capacity of our model to solve the LPI problem. Section. IV describes our experimental setting and demonstrate empirically the performance of TL-GNN. Finally, Section. V concludes the paper and offers directions for future work.

II. Related Work

GNNs have achieved state-of-the-art results on graph classification, link prediction and semi-supervised node classification.However, recent studies [4] [5] [6] have demonstrated one of the severe representational limitations of GNNs, namely Local Permutation Invariance (LPI). In this section, we will briefly review these interrelated topics.

A. Graph Neural Networks.

Graph Neural Networks (GNNs) have proved to be an effective machine learning tool for non-Euclidean structure data for several years. Since the GNN was first presented in [1], a set of more advanced approaches have been proposed, including but not limited to GraphSAGE [14], Graph Attention Networks (GAT) [2], Graph Isomorphism Network (GIN) [3], edGNN [15]. These methods learn local structural information by recursively aggregating neighbor representations.

In a macroscopic view, the above models follow the same pattern. For each node $v \in V$ within a graph $G = (V, E)$, GNNS capture k-hop neighbor information $h^k_{\mathcal{N}(v)}$, and then learn a representational vector h^k_v after k layers of processing. On this basis, tasks such as graph classification can be accomplished. In fact, the critical difference between GNN variants is how they design and implement the neighbor aggregation function. Xu et al. [3] summarized some of the most common GNN approaches and proposed a general framework referred to as the Graph Isomorphism Network (GIN). The GIN approach defines the above steps as three related functions, namely AGGREGATE (AGG), COMBINE (COM), and READOUT (READ),

$$\begin{cases} h^k_{\mathcal{N}(v)} = AGG(\{h^{(k-1)}_{\mu}, \forall \mu \in \mathcal{N}(v)\}), \\ h^k_v = COM(h^{k-1}_v, h^k_{\mathcal{N}(v)}), \\ h_G = READ(\{h^k_v | v \in G\}), \end{cases} \tag{1}$$

The initialization is $h^0_v = X_v$, and X_v represents the initial features of the nodes. The quantity $\mathcal{N}(v)$ represents the set of nodes adjacent to v and h_G is the graph representation vector.

Xu et al. [3] indicate that what makes GNN so powerful is the injective aggregation strategy, which maps different nodes to different representational units. They also demonstrate that when the above three functions are all injective functions, for example a sum, then the GNN can be as powerful as the WL test [16] on the graph isomorphism problem.

B. Local Permutation Invariance

Even though the precise form of the aggregation or combination strategy varies across different GNN architectures, most share the same neighborhood aggregation concept. This characteristic leads to an underlying graph isomorphism problem, the so-called local permutation invariance (LPI). A more common LPI example in the real world is edge rewiring [17] [18]. For a graph $G = (V, E)$, edge rewiring operation alters the graph structure and leads to a new graph $G^{'} = (V^{'}, E^{'})$ by exchanging a pair of edges, for instance removing edges e_{AB}, e_{CD} between nodes A, B, C, D and adding edges e_{AC}, e_{BD}. After changing, G and $G^{'}$ are non-isomorphic graphs but have identical representation in GNNs: $h_G = h_{G'}$.

Garg et al. [4] analyze specific cases of LPI for the GNNs aggregation function and provide generalization bounds for message passing in GNNs. As shown in Fig. 1, the graphs are obviously non-isomorphic, but their node-level characteristics are identical. Unfortunately, existing GNNs focus on extracting node-level information and neglect high-level information, so they suffer representational limitations. Recent studies have attempted to overcome these limitations by providing additional information to the nodes. Consistent Port Numbering GNN (CPNGNN) [5] assigns port numbers to nodes and treats their neighbors differently. Klicpera et al. [6] propose DimNet for molecular graphs. This embeds whole atoms using a collection of edge embeddings and takes advantage of directional information by modifying messages based on their angle. However, Garg et al. [4] demonstrate that there are some graphs that both CPNGNN and DimeNet can not distinguish. This means that these approaches fail to overcome the representational limitations caused by LPI.

C. Subgraph Methods.

Subgraphs can be regarded as the basic structural elements or building blocks of the larger graph, including paths [19] and subtrees [20]. Subgraph frequency was studied as a local feature in social networks by Ugander et al. [21], who discovered that it can provide unique insights for recognizing both social organization and graph structure in very large networks. Ugender et al. [21] propose a novel graph homomorphism algorithm based on subgraph frequencies. They define a coordinate system that is beneficial both to the representation and understanding of large sets of dense graphs. Grochow et al. [22] introduce a symmetry-breaking technique into the motif (subgraph) discovery process and develop a novel motif discovery algorithm that can achieve an exponential speed-up. More recently, significant effort has been expended in the design of subgraph-based GNNS for graph classification. The role of substructures has been explored empirically by Ying et al. [23] and used to interpret the predictions made by GNNs.

In this paper, we propose an architecture which we refer to as the Two-level Graph Neural Network (TL-GNN), which is designed to make the association between the neighbor node and subgraph structural information. Our model mitigates the negative impacts of the LPI problem and is verified to be efficient on real-world datasets.

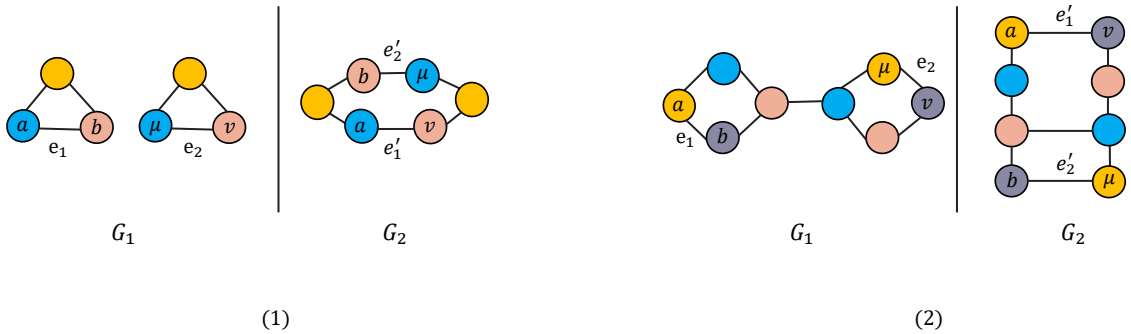


Fig. 1. Two LPI examples, G_1 and G_2 have identical node-level information but different structures. In other words, they are non-isomorphic graphs. The nodes of the same colour have the same features. The numbers of nodes and edges number together with the node features are identical. The only difference between them is that a pair of edges of G_1: $e_1 = (a, b)$, $e_2 = (\mu, v)$, change their nodes and transform the edges of G_2: so that $e_1^{'} = (a, v)$, $e_2{}^{\prime} = (b, \mu)$.

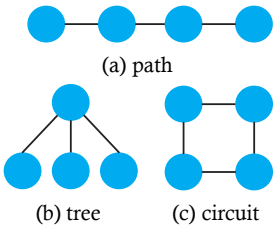


Fig. 2. Subgraphs example

III. Proposed Method

In order to extract subgraph-level information from a graph, we first count all subgraphs within a certain size range of a graph and then generate a new graph, namely generated graph, whose nodes (or supernodes) represent subgraphs in the original graph. We develop a novel subgraph counting algorithm for this purpose. After the generated graph is to hand, we develop a novel GNN framework, that significantly extends the existing GNN framework. We also develop two key operators for the novel framework that facilitate the effective merging of subgraph and node information. This Section will introduce the proposed subgraph counting algorithm and the new GNN framework.

A. Constructing the Generated Graph

Consider an undirected graph $G = (V, E)$, where V and $E \subseteq (V \times V)$ respectively denote the set of nodes and the set of edges. The element (v_i, v_j) in E is an unordered pair of nodes v_i and v_j, $i, j = 1, 2, 3...N$, where N is the number of nodes in the graph, i.e. the size of the network.

1) Subgraph Counting Algorithm : We aim to identify three different types of subgraph structures, namely trees, paths, and circuits (cycles). These represent the different basic classes of subgraph structure, and structures such as triangles or quadrilaterals can be regarded as different specific cases as shown in Fig.2. Finding all subgraphs within a graph is an NP-hard problem. To implement our method, it is unnecessary to find all subgraphs. For each node in turn, we identify all subgraphs that are contained within its D-hop neighbors.

Algorithm 1 Tree counting

Input: input original graph $G(V, E)$ and its adjacency matrix A, hyper parameter $tree\_threshold$.
Output: output subgraph set S
Initialize subgraphset S, tree set $Tree$, neighbor set $\{N(v)|t = 0, 1, 2...2^T, \forall v \in V\}$, and path from v to $\mu : \{P_v(\mu)|\forall v, \mu \in V\}$
for $v \in V$ **do**
  for $\mu \in V$ **do**
    if $A_{v\mu} = 1$ **then**
      add μ into $N(v)$
      add v into $N(\mu)$
      $P_v(\mu) \leftarrow v$
      $P_\mu(v) \leftarrow \mu$
    end if
  end for
  if $|N(v)| > tree\_threshold$ **then**
    add $(N(v) + [v])$ into $Tree$
  end if
end for
add $Tree$ into S
return S

We design a dynamic programming algorithm to achieve this goal. Our algorithm consists of three steps for tree, path, and circuit location. Firstly, we store the neighbor set of each node $v \in V$ and select those tree-shaped subgraphs which have more than three nodes ($tree\_threshold = 3$). A 3-node tree or a 2-node tree is essentially a 3-node path or a 2-node path. This step is shown in Algorithm. 1. Secondly, we find all path-shaped and circuit-shaped subgraphs based on the dynamic programming algorithm. This step is realized using Algorithm. 2 and Algorithm. 3. If the node μ is one of the 2^d-hop neighbors of the node v and the node a is one of the i-hop neighbors of the node μ ($i \leq 2^d$), then the node a is one of the $(2^d + i)$-hop neighbors of the node v. After Algorithm. 1 locates the 1-hop neighbors of each node, Algorithm. 2 and Algorithm. 3 can locate the 2-hop neighbors of each node by two sequential 1-hop searches. After locating the 2-hop neighbors of each node, the 3-hop neighbors and

Algorithm 2 Path and circuit counting

Input: input original graph $G(V, E)$, depth D, subgraph set S and d-hop adjacency matrix set $\{A^d \in \mathbb{R}^{|V|\times|V|} | d = 2...2^D\}$ and path from v to $\mu : \{P_v(\mu)|\forall v, \mu \in V\}$
Output: output subgraph set S
for $d = 0, 1, 2...D$ **do**
  for $v \in V$ **do**
    for $\mu \in V$ **do**
      for $a \in V$ **do**
        $Circuit, Path$=Algorithm3
      end for
    end for
  end for
end for
add $Circuit, Path$ to S
return S

Algorithm 3 Path and subgraph sifting

Input: input original graph $G(V, E)$, adjacency matrix A, subgraph set S, path from v to $\mu : \{P_v(\mu)|\forall v, \mu \in V\}$ and $\{A^d | d = 2...2^D\}$
Output: output circuit set $Circuit$ and path set $Path$
for $i = 1, 2, ..., 2^d$ **do**
  if $A_{v\mu}^{(2^d)} = 1$ and $A_{\mu a}^{(i)} = 1$ **then**
    if $a \notin P_\mu(v)$ and $P_v(a) \neq \varnothing$ **then**
      add $P_v(\mu) + P_\mu(a) + P_a(v)$ to $Circuit$
    end if
    if $a \notin P_{\mu v}$ and $P_{va} = \varnothing$ **then**
      $A_{av}^{(2^d+i)} = 1$
      $A_{va}^{(2^d+i)} = 1$
      $P_a(v) = P_a(\mu) + P_\mu(v)$
      $P_v(a) = P_v(\mu) + P_\mu(a)$
      add $P_v(a)$ to $Path$
    end if
  end if
end for
return $Circuit$, $Path$

4-hop neighbors of each node can be found. Besides, we store all nodes on the path from the node v to the node μ. If the node μ's i-hop neighbor a is not on the path from the node v to the node μ and the node v has a path to the node a, a circuit can be found. Otherwise, a path can be obtained.

The main advantage of the proposed algorithm is low complexity. Early subgraph counting algorithms [24] [25] [20] required exponential time complexity. The current best-known algorithm [26] for exact subgraph counting, which requires $O(n^{\frac{\omega D}{3}})$, where ω and D are exponent of fast matrix multiplication and nodes number of subgraphs. Due to ω can be neglected, the time complexity of this algorithm is $O(n^D)$. However, the proposed method only requires $O(n^3)$. As for the proposed methods, the time complexity of Algorithm. 1 is $O(n^2)$, $n = |V|$. The time complexity of Algorithm. 2 and Algorithm. 3 is $O(2^D n^3)$. Due to the fact that D is a scalar no larger than 10, the factor of 2^D can be neglected. The time complexity of the proposed method is therefore $O(n^3)$.

As for space complexity, space complexities of $P_u(v)$ and A_d are $O(n^3)$ and $O(2^D n^2)$ respectively. As a result the space complexity of the proposed subgraph counting method is $O(n^3)$.

2) Generating Graphs: After the subgraph counting is complete, we generate a new graph to represent the subgraph relationships. Given a network $G(V, E)$, the generated graph $G^*(V^*, E^*)$ consists of the set of supernodes representing the detected subgraphs V^* and the set of edges $E^* \subseteq (V^* \times V^*)$ representing the relationships between them. Two subgraphs are connected if they share common nodes or links in the original network. The features associated with the supernodes are represented by a two-dimensional vector, whose components are the node counts and the subgraph type respectively. Fig. 3 provides an example.

We also record the appearances of each node in V^* by constructing a transformation matrix $T \in \mathbb{R}^{N\times M}$, $|V| = N, |V^*| = M$. The transformation matrix T indicates the correspondences between nodes and subgraphs (i.e. supernodes), i.e. which supernodes subsume each node. The elements of the transformation matrix are defined as follows:

$$T_{ij} = \begin{cases} 1, & if\ node\ i\ in\ subgraph\ j \\ 0, & else \end{cases} \tag{2}$$

B. The LPI problem and subgraphs

Garg et al. [4] indicate the limitations of GNNs caused by Local Permutation Invariance (LPI). Specifically, changing a pair of edges in a graph leads to structure changing, while node-level information of the graph is maintained. The existing aggregation strategy, which extracts node-level information only is unable to distinguish structure change. Garg et al. [4] provide several example structures which can not be distinguished by existing methods, as illustrated in Fig. 1. Specifically, two non-isomorphic graphs, G and $G^{'}$ have identical node-level information which confuses GNNs to distinguish them. G and $G^{'}$ can be transformed into each other by exchanging a pair of edges. It means their only difference is endpoints of the pair of edges.

However, Garg et al. [4] provide examples only, without giving mathematical definitions for the ambiguities encountered.

Without loss of generality, we firstly indicate the general characteristics of graphs that existing GNNs can not distinguish due to the LPI ambiguity. Secondly, we demonstrate that for those graphs that GNNs can not distinguish due to ambiguities, our generated graphs behave in a different and useful manner. Finally, we demonstrate mathematically the effectiveness of our method in Section III-C.

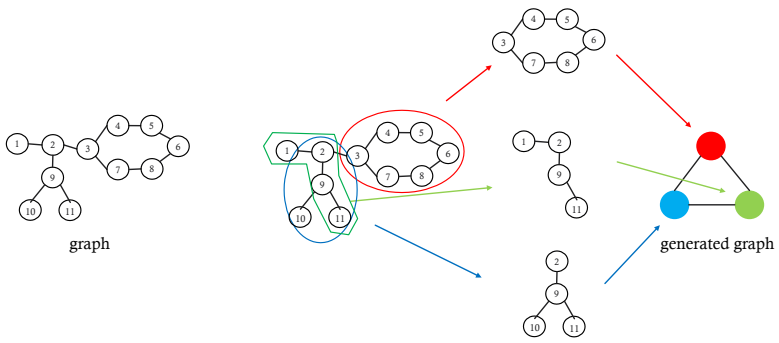


Fig. 3. Example of the graph generation process. The red supernode corresponds to a 6-node circuit in the graph. The green and blue supernodes correspond to a 4-node path and a 4-node tree in the graph respectively.

These results concerning LPI lead us to propose a new neighborhood aggregation strategy for GNNs. While the existing neighborhood aggregation strategy is effective, it sometimes fails to distinguish non-isomorphic graphs. To demonstrate this we give a definition of Permutation Non-isomorphic Graphs(PNG):

Definition 1. Permutation Non-isomorphic Graphs (PNG) are non-isomorphic graphs that have the same node set and the same node features but swap the nodes of two edges.

Assume $G = \{V, E\}$ and $G^{'} = \{V^{'}, E^{'}\}$ are PNG. $v \in V, v^{'} \in V^{'}, e_1, e_2 \in E, e_1^{'}, e_2^{'} \in E^{'}$, then they must satisfy the following conditions:

$$\begin{cases} |V| = |V^{'}|, & (a) \\ |E| = |E^{'}|, & (b) \\ h_v^0 = h_{v^{'}}^0 \quad \forall v \in V, & (c) \\ N(v) = N(v^{'}), \quad \forall v \in V, & (d) \\ e_1 = \{a, b\}, e_2 = \{i, j\}, e_1^{'} = \{a^{'}, j^{'}\}, e_2^{'} = \{i^{'}, b^{'}\}, & (e) \end{cases} \tag{3}$$

The above conditions indicate that the only difference between $G = \{V, E\}$ and $G^{'} = \{V^{'}, E^{'}\}$ is a single pair of edges. The remaining characteristics of $G = \{V, E\}$ and $G^{'} = \{V^{'}, E^{'}\}$ are identical.

Due to LPI, the GNNs based neighbor aggregation strategy can not distinguish permutation non-isomorphic graphs, which we state in Theorem 2:

Theorem 2. GNNs based on Eq(1) can not distinguish permutation non-isomorphic graphs.

We give a proof of this theorem as follows. For two permutation non-isomorphic graphs $G = (V, E)$ and $G^{'} = (V^{'}, E^{'})$, their neighbourhood aggregation from the 0-th GNN layer are:

$$h_{\mathcal{N}(v)}^0 = AGG(\{h_\mu^{(0)} | \mu \in \mathcal{N}(v)\}), \forall v \in V, \tag{4}$$

$$h_{\mathcal{N}(v^{'})}^0 = AGG(\{h_\mu^{(0)} | \mu \in \mathcal{N}(v^{'})\}), \forall v^{'} \in V^{'}. \tag{5}$$

Due to the conditions (d) given in Definition 1:

$$h_{\mathcal{N}(v)}^0 = h_{\mathcal{N}(v^{'})}^0, \quad h_v^1 = h_{v^{'}}^1, \tag{6}$$

According to the above equations and the mathematical inductive. the representations of $v \in V, v^{'} \in V^{'}$ of the l-th layer meet the condition:

$$h_v^l = h_{v^{'}}^l, \tag{7}$$

For a GNN with K layers, the representations of G and $G^{'}$ are identical:

$$h_G = READ(\{h_v^k | v \in G\}), h_{G^{'}} = READ(\{h_{v^{'}}^k | v^{'} \in G^{'}\}). \tag{8}$$

$$h_G = h_{G^{'}}. \tag{9}$$

Obviously, existing GNN provides identical representation for a pair of PNG: $h_G = h_{G^{'}}$. The reason that GNNs are confused by PNGs is due to their adopted neighbor aggregation strategy, which captures node-level information only. Unfortunately, PNGs share identical characteristics at the node-level but have different global structures. As a result, GNNs can not distinguish PNGs effectively.

Garg et al. [4] have indicated that the LPI problem limits the representational power of GNNs. Theorem 2 further indicates how the LPI problem affects the graph classification performance of GNNs from the perspective of graph isomorphism. Another interesting question is how the LPI problem influences the node classification task. Unlike the graph structure which is implicated in the LPI problem, the node classification task is unaffected by global structural information from the whole graph. Therefore, to what extent and precisely how the LPI problem influences node classification needs deeper mathematical analysis and associated proofs, which are beyond the scope of this paper we will we hope to investigate in more detail in further work. In this paper, we simply focus on the LPI problem for the graph classification task.

However, although PNGs have identical node-level characteristics, their subgraph-level characteristics are different. The following lemmas demonstrate that the structural differences between PNGs can be learned from their generated graphs. In other words, the subgraph-level information is helpful in distinguishing PNGs.

Lemma 3. For two permutation non-isomorphic graphs $G = (V, E)$ and $G^{'} = (V^{'}, E^{'})$, their generated graphs $Gg = (Vg, Eg)$ and $Gg^{'} = (Vg^{'}, Eg^{'})$ are structurally distinct.

We provide a proof of Lemma 3. by analyzing the relationship between edges and subgraphs. For the edges:

$$e_i \in E, \quad e_i^{'} \in E^{'}, \quad (i = 1, 2), \tag{10}$$

$$e_1 = (a, b), \quad e_2 = (i, j), \quad e_1^{'} = (a, i), \quad e_2^{'} = (b, j), \tag{11}$$

there are four types of relations between them.

Relation 1:

As shown in Fig. 4 (1), for the subgraphs S_i and their nodes V_i and edges E_i:

$$S_i = (V_i, E_i), \quad S_i^{'} = (V_i^{'}, E_i^{'}), \quad (i = 1, 2). \tag{12}$$

Relation 1 can be written as:

$$e_i \in E_i, e_i^{'} \notin E_i^{'}, i = 1, 2. \tag{13}$$

Due to the features of supernodes (subgraphs) include node counts and the subgraph type respectively (Section. III-A2), the edges change must lead to node counts or subgraph change of subgraphs. We thus have:

$$E_i \neq E_i^{'}, h_{S_1}^0 \neq h_{S_1^{'}}^0, \tag{14}$$

and so, $Vg \neq Vg^{'}$, Gg and $Gg^{'}$ are different.

Relation 2:

As shown in Fig. 4 (2), consider the subgraphs:

$$S_i = (V_i, E_i), S_i^{'} = (V_i^{'}, E_i^{'}), (i = 1, 2, 3). \tag{15}$$

Relation 2 can be written as:

$$e_1 \in E_1, e_i^{'} \notin E_j^{'}, e_2 \notin E_j, \tag{16}$$

$$i = 1, 2 \quad j = 1, 2, 3. \tag{17}$$

Due to the relation:

$$e_1 \in E_1, e_1^{'} \notin E_1^{'}. \tag{18}$$

We have:

$$V_i = V_i^{'}, (i = 1, 2, 3), \tag{19}$$

$$E_1 \neq E_1^{'}, E_j = E_j^{'}, (j = 2, 3). \tag{20}$$

So, $S_1 \neq S_1^{'}$, $h_{S_1}^0 \neq h_{S_1^{'}}^0$. Gg and $Gg^{'}$ are different.

Relation 3:

As shown in Fig. 4 (3), for the subgraphs:

$$S_i = (V_i, E_i), S_i^{'} = (V_i^{'}, E_i^{'}), (i = 1, 2, 3, 4). \tag{21}$$

Relation 3 can be written as:

$$e_i \notin E_j, e_i^{'} \notin E_j^{'}, E_j = E_j^{'}. \tag{22}$$

$$i = 1, 2 \quad j = 1, 2, 3, 4. \tag{23}$$

$Gg = Gg^{'}$ if and only if:

$$\begin{cases} h_{S_1}^0 = h_{S_1^{'}}^0 = h_{S_3}^0 = h_{S_3^{'}}^0 \\ h_{S_2}^0 = h_{S_2^{'}}^0 = h_{S_4}^0 = h_{S_4^{'}}^0 \\ \exists e_3 = (S_1, S_4), \exists e_4 = (S_2, S_3), \\ \exists e_5 = (S_1, S_2), \exists e_6 = (S_3, S_4) \end{cases} \tag{24}$$

Obviously, $G = G^{'}$ when $Gg = Gg^{'}$. Because swapping the nodes constituting edges does not change the connection structure of the subgraphs and their nodes, the structures are identical.

Else, $Gg \neq Gg^{'}$, because of the super-node features or connections of super-nodes.

Relation 4:

As shown in Fig. 4 (4), for the subgraphs:

$$S_1 = (V_1, E_1), S_1^{'} = (V_1^{'}, E_1^{'}), \quad e_1 \in S_1, e_1^{'} \in S_1^{'}. \tag{25}$$

Obviously, $S_1, S_1^{'} \notin Tree$. If $S_1 \in Path, S_1^{'} \in Cir^{'}$ or $S_1 \in Cir, S_1^{'} \in Path^{'}$:

$$h_{S_1}^0 \neq h_{S_1^{'}}^0. \tag{26}$$

There is a special case of Lemma 3 which deserves comment. If S_1 is a Path or a Cir, changing a pair of edges would divide S_1 into two subgraphs or lead to a subgraph identical to S_1 but with a different node sequence. We will discuss this situation in Lemma 4.

Lemma 4. GNNs can distinguish the structural variance caused by the node sequence.

In summary, because GNN adopts a hierarchical aggregation strategy, different node sequences will lead to different information aggregated by GNN in different layers.

As shown in Fig. 5. If u, v are i-hop and j-hop neighbors of node a respectively, then $u^{'}, v^{'}$ are respectively j-hop and i-hop neighbors of node $a^{'}$. Then as a result $h_a^i \neq h_{a^{'}}^i$ and $h_a^j \neq h_{a^{'}}^j$. Therefore, GNNs can distinguish them.

Lemma 5. As for Lemma 3, if $Gg = (Vg, Eg)$ and $Gg^{'} = (Vg^{'}, Eg^{'})$ are different in terms of their supernode features or subgraph connections, then a GNN can distinguish the generated graphs.

Lemma 3 and Lemma4 lead to Lemma 5. Due to the properties of GNNs, Lemma 5 can be demonstrated easily.

Suppose $S^{'}$ is the permuted subgraph of $S \in V_g$. Under relations 1,2 and 4, $\exists S \in Vg, S^{'} \in Vg^{'}$ and let the equation below be tenable:

$$h_S^0 \neq h_{S^{'}}^0. \tag{27}$$

Similar to the proof of Theorem. 2, we can demonstrate $h_{G_g} \neq h_{G_g^{'}}$ via the mathematical inductive.

For relation 3, we have:

$$\mathcal{N}(S) \neq \mathcal{N}(S^{'}), \tag{28}$$

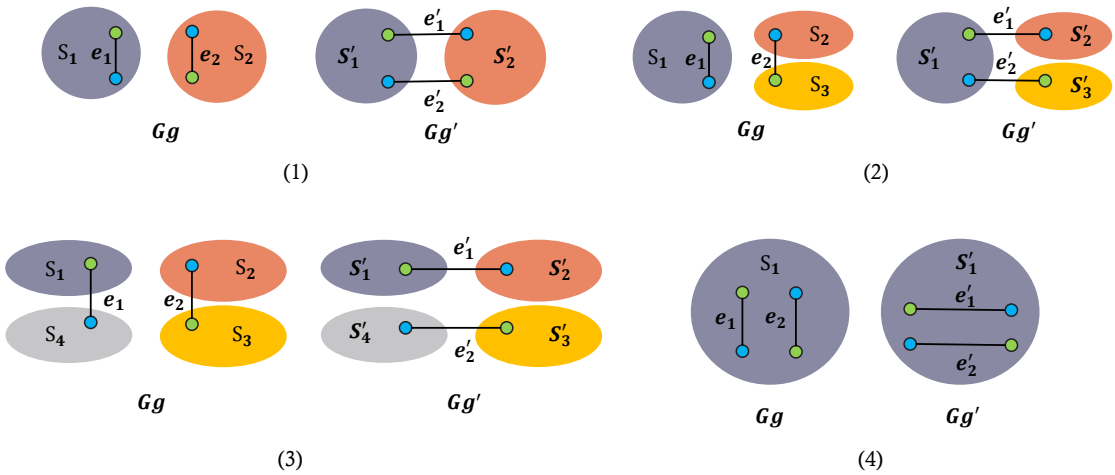


Fig. 4. Four relations between edges and subgraphs. S_i is the subgraph with the index i. Blue and green points are nodes. The same color nodes share the same feature.

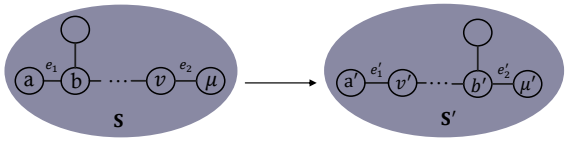


Fig. 5. A special case. Changing a pair of edges, e_1 and e_2, leads to different node sequences. Subgraphs S and $S^{'}$ have identical structure but different node sequences.

which means:

$$h_{\mathcal{N}(S)} \neq h_{\mathcal{N}(S')}, \tag{29}$$

After combining operations, we have:

$$h_S^1 = COM(h_S^0, h_{N(S)}), \quad h_{S'}^1 = COM(h_{S'}^0, h_{N(S')}). \tag{30}$$

Due to COM is an injective function, different inputs will be mapped into different points in the feature space. It means that graph representations after iterations are different:

$$h_S^1 \neq h_{S'}^1, \quad h_{G_g} \neq h_{G_g'}. \tag{31}$$

Eq. (31) shows that PNGs can be distinguished by the differences between subgraphs, which proves the Lemma 5. According to the above theoretical analysis, we propose the framework of TL-GNN and discuss it in the next subsection.

C. The TL-GNN Framework

As noted above the LPI of GNNs lead to serious representational limitations. Existing GNNs are therefore sometimes compromised in their performance by PNGs. Fortunately and as we have shown, the subgraph-level information can be beneficial in distinguishing the PNGs. To overcome the limitations caused by the LPI problem and enrich the representational capacity of GNNs, we propose a novel GNN approach which we refer to as the Two-level Graph Neural Network(TL-GNN). In this section, we present details of the TL-GNN framework.

Due to the neighbor aggregation strategy, existing GNNs focus on capturing node-level information and ignore higher level structural arrangement information. Unlike existing GNNs, the TL-GNN can capture both node-level information and subgraph-level arrangement information simultaneously. This is achieved by merging subgraph-level information into supernode-level information.

Specifically, the TL-GNN accepts both a graph and its generated graph (as described in Section III-A) as inputs. In each layer, the two separate GNN components capture information concerning the graph (node-level) and generated graph (subgraph-level) respectively, as illustrated in Fig.6.

The node-level propagation process can be described as:

$$\begin{cases} h_{\mathcal{N}(v)}^k = AGG(\{h_\mu^{(k-1)}, \forall \mu \in \mathcal{N}(v)\}), \\ \widetilde{h_v^k} = COM(h_v^{k-1}, h_{\mathcal{N}(v)}^k), \end{cases} \tag{32}$$

On the other hand, the subgraph-level propagation can be described as:

$$\begin{cases} h_{\mathcal{N}(s)}^k = AGG(\{h_\gamma^{(k-1)}, \forall \gamma \in \mathcal{N}(s)\}), \\ h_s^k = COM(h_s^{k-1}, h_{\mathcal{N}(s)}^k), \end{cases} \tag{33}$$

Here $\mathcal{N}(v)$ and $\mathcal{N}(s)$ are the neighbor sets of node v and supernode s respectively. These two information propagation processes are discrete and do not share parameters.

1) The AGG_SUB and MERG functions: With a two-level representation to hand, we merge the subgraph-level representation into the node-level representation. We design two functions AGG_SUB and MERG for this process.

For a graph containing N nodes, assume that its generated graph contains M supernodes. The outputs of the $k-$th layer

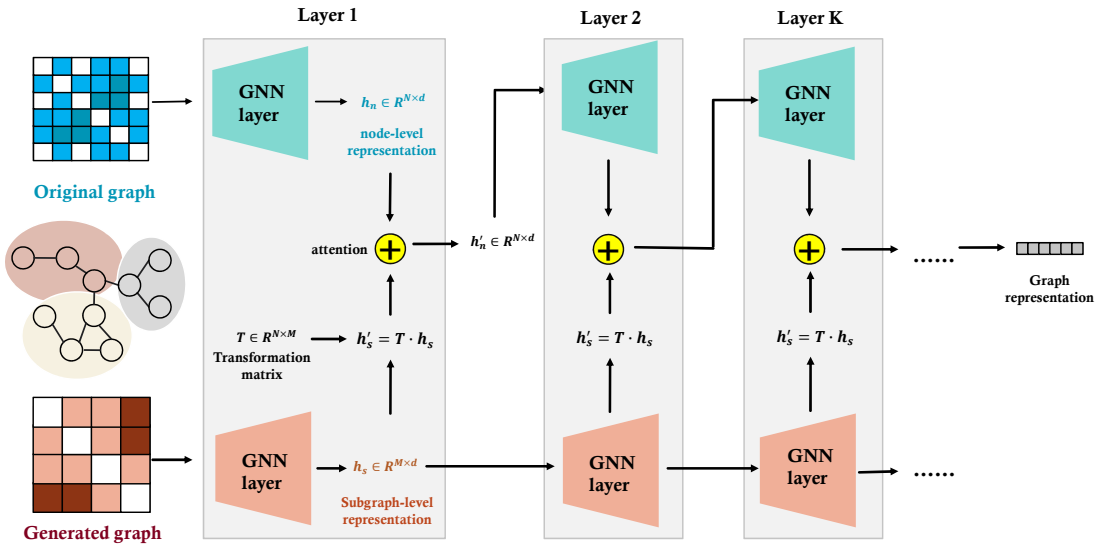


Fig. 6. Visual illustration of the TL-GNN framework. The model takes both the original graph and generated graph as input, then aggregates them with the same strategy. Each TL-GNN layer has two separate GNN blocks for graph and generated graph to capture node-level and subgraph-level information respectively. At the end of each layer, the subgraph-level representation h_s would be converted into h'_s via the transformation matrix T, which could be aligned with node-level representation h_n. On this basis, two independent representations can be merged together with the attention mechanism. The summation h'_n would be the input of the next layer.

of TL-GNN are the node-level representation $\widetilde{h}_N^k$ and the subgraph-level representations h_S^k respectively:

$$\widetilde{h}_N^k \in \mathbb{R}^{N\times d}, \quad h_S^k \in \mathbb{R}^{M\times d}, \tag{34}$$

Here $\widetilde{h}_N^k$ and h_S^k are matrices that contain the node and supernode (subgraph-level) representations respectively. Each row of h_N^k or h_S^k corresponds to the representation of an individual node or supernode.

We merge (or concatenate) the matrices h_N^k and h_S^k into a single matrix $\widetilde{h_N^k}$ using the MERG function. Then $\widetilde{h_N^k}$ and h_S^k become the inputs into the next layer of the GNN. Each layer has an identical matrix structure. After applying the merging step, the representations of the nodes are merged with the available subgraph-level information.

Next, we define two more functions for the TL-GNN, namely AGG_SUB and MERG:

$$H_s^k = AGG\_SUB(h_S^k), \quad h_N^k = MERG(\widetilde{h_N^k}, H_S^k), \tag{35}$$

The AGG_SUB operator aggregates all of the subgraph representations which contain a given node v. The output of the MERG function is the matrix h_N^k, which merges the node-level and subgraph-level information into a single representation.

Finally, the representation of the graph G is obtained from the representations of its nodes.

$$h_G = READ(\{\widetilde{h_v^k} | v \in G\}), \tag{36}$$

In order to distinguish PNGs, the AGG_SUB and MERG need to fulfill conditions specified in Lemma 6:

Lemma 6. If AGG_SUB and $MERG$ are both injective multiset functions, then the representation of G is distinct from that of $G^{'}$.

To demonstrate Lemma 6, We define $C(v)$, which is a set of subgraphs that contain node v. According to Lemma 5:

$$\exists S \in C(v), S^{'} \in C(v'), \quad h_S^1 \neq h_{S'}^1, \tag{37}$$

As

$$h_v^0 = h_{v'}^0, \quad h_{c(v)}^0 \neq h_{c(v')}^0, \tag{38}$$

$$h_v^1 = MERG(h_v^0, h_{c(v)}^0), \quad h_{v'}^1 = MERG(h_{v'}^0, h_{c(v')}^0), \tag{39}$$

$$h_v^1 \neq h_{v'}^1, \tag{40}$$

The above equations are satisfied if and only if $MERG$ is an injective function. When the above equations are satisfied, we can obtain $h_G \neq h_{G'}$, which means G and G' can be distinguished. Lemma 6 is proved.

There are several choices available for the AGG_SUB and MERG functions, which include summation and concatenation. We chose summation for AGG_SUB of TL-GNN.

Due to the fact that the graph size and generated graph size are different, the subgraph-level representations need to be transformed into node-level representations of identical size.

The translation matrix mentioned in Section III-A translates the matrix h_S^k into the matrix $H_S^k \in \mathbb{R}^{N\times d}$:

$$H_S^k = T \cdot h_S^k, \tag{41}$$

Due to the definition of T, the i-th row of H_S^k is the summation of representations of those supernodes that contain node i. Assume i-th line of H_S^k is $(H_s^k)_i$:

$$(H_s^k)_i = \sum_{s\in\mathcal{C}(i)} h_s^k, \tag{42}$$

where $\mathcal{C}(i)$ is the set of supernodes (subgraphs) that contain node i.

As for the MERG operation, we use an attention mechanism to define the MERG function. The attention mechanism for TL-GNN is essentially a weighted summation. Therefore,the MERG operation can be rewritten as:

$$h_v^k = \alpha^k \cdot \widetilde{h_v^k} + \beta^k \cdot H_s^k, \tag{43}$$

$$\alpha^k = \frac{exp(\hat{\alpha}^k)}{exp(\hat{\alpha}^k) + exp(\hat{\beta}^k)}, \quad \beta^k = \frac{exp(\hat{\beta}^k)}{exp(\hat{\alpha}^k) + exp(\hat{\beta}^k)}, \tag{44}$$

where $\hat{\alpha}^k$ and $\hat{\beta}^k$ are randomly initialized scales. α and β satisfy the condition $\alpha^k + \beta^k = 1$. The parameters α^k and β^k can be learned during training. The parameter α^k is large if the node-level representation is more important for the classification of graphs, and vice versa.

Finally, the effectiveness of TL-GNN in distinguishing PNGs can be stated mathematically.

Theorem 7. TL-GNN has the ability to distinguish the PNG.

We provide the proof of Theorem 7 using the above Lemmas.

For permutation non-isomorphic graphs $G = (V, E)$ and $G^{'} = (V^{'}, E^{'})$, According to Lemma 6, we have:

$$h_v^1 \neq h_{v'}^1 \; v \in \; V \; v^{'} \in \; V^{'}. \tag{45}$$

According to mathematical induction, the representations of $v \in V, v^{'} \in V^{'}$ of l-th layer meet condition:

$$h_v^l = h_{v'}^l. \tag{46}$$

For a GNN with K layers, the representations of G and $G^{'}$ are different:

$$h_G = READ(\{h_v^k | v \in G\}), \quad h_{G'} = READ(\{h_{v'}^k | v^{'} \in G^{'}\}), \tag{47}$$

$$h_G \neq h_{G'}. \tag{48}$$

So, TL-GNN can distinguish permutation non-isomorphic graphs G and $G^{'}$.

Theorem 7 indicates that the TL-GNN can distinguish those ambiguous graphs that confuse existing GNNs. In other words, TL-GNN is more powerful than GNNs.

IV. Experiments

In this section, we perform experimental evaluations of our TL-GNN method on the graph classification task. We compare the TL-GNN to several state-of-the-art deep learning and graph kernel methods, and conduct experiments on seven standard graph classification benchmarks together with synthetic data.

A. Datasets

Datasets of this paper include MUTAG [27], PTC [28], NCI1 [29], PROTEINS [30], COX2 [31], IMDB_ M [32] and IMDB_B [33]. The IMDB_ M and IMDB_ B datasets have no node features. The remaining datasets have categorical node features. In order to verify the ability to distinguish PNGs, we have prepared a synthetic PNG dataset named SPNG. The details of these datasets are shown in Appendix.

B. Baselines for Comparison

The baselines used for comparison include state-of-the-art methods which are applied to the graph classification task:

(1) The kernel based methods: Weisfeiler-Lehman(WL) [34] and subgraph Matching Kernel (CSM) [35], Deep Graph Kernel (DGK) [36].

(2) The state-of-the-art GNNs: Graph convolution network (GCN) [37], Deep Graph CNN (DGCNN) [38], Graph Isomorphism Network (GIN) [3], Random Walk Graph Neural Network (RW-GNN) [39], Graph Attention Network (GAT) [2], Motif based Attentional Graph Convolutional Neural Network (MA-GCNN) [40].

C. Experimental Setup

For our experimental comparison, we set the GNN layers of GIN so as to have the same structure but with no parameter sharing. These layers aggregate and combine the original graph and its generated graph. There are independent attention parameter pairs for merging operations between the GNN layers. Each GNN layer has several MLP layers. More details about experimental setup are shown in Appendix.

D. Results and Discussion

Comparison with existing GNNs on real-world datasets: The results in Table I indicate that TL-GNN achieves the best results on 6 out of 7 benchmarks, often with a clear improvement over alternative GNN methods studied. The performances for the classical GNNs are quoted from their indicated reference. We perform 10-fold cross-validation to compute the GIN, GCN and RW-GNN accuracies on COX2. The parameters for the deep learning methods are as suggested by their authors. For fairness, all the methods run on the same computing device. For cases where accuracy cannot be obtained, we use the "-" tag in Table I.

TABLE I
CLASSIFICATION ACCURACY (IN % ± STANDARD ERROR)

Datasets	MUTAG	PTC	NCI1	IMDB-M	IMDB-B	COX2	PROTEINS
WL	90.4 ± 5.7	59.9 ± 4.3	$\mathbf{86 \pm 1.8}$	50.9 ± 3.8	73.8 ± 3.9	83.2 ± 0.2	75.0 ± 3.1
CSM	85.4	63.8	65.5	63.3	58.1	80.7 ± 0.3	-
DGK	87.4 ± 2.7	60.1 ± 2.6	80.3 ± 0.5	43.9 ± 0.4	65.9 ± 1.0	-	71.7 ± 0.6
GCN	85.6 ± 5.8	64.2 ± 4.3	80.2 ± 2.0	51.9 ± 3.8	74.0 ± 3.4	-	76.0 ± 3.2
DGCNN	85.8 ± 1.7	58.6 ± 2.5	74.4 ± 0.5	47.8 ± 0.9	70.0 ± 0.9	-	70.9 ± 2.8
GIN	89.4 ± 5.6	64.6 ± 7.0	82.7 ± 1.7	52.3 ± 2.8	75.1 ± 5.1	83.3 ± 5.3	76.2 ± 2.8
FDGNN	88.5 ± 3.8	63.4 ± 5.4	77.8 ± 2.6	50.0 ± 1.3	72.4 ± 3.6	83.4 ± 2.9	76.8 ± 2.9
RW-GNN	89.2 ± 4.3	61.6 ± 9.5	–	47.8 ± 3.8	70.8 ± 4.8	81.6 ± 4.7	74.7 ± 3.3
GAT	89.4 ± 6.1	66.7 ± 5.1	75.2 ± 3.3	47.8 ± 3.1	70.5 ± 2.3	-	74.7 ± 2.2
HA-GCNN	93.9 ± 5.2	71.8 ± 6.3	81.8 ± 2.4	53.8 ± 3.1	77.2 ± 3.0	-	79.4 ± 1.7
TL-GNN_sm	90.9 ± 6.4	68.1 ± 5.0	81.9 ± 3.3	54.3 ± 4.7	77.5 ± 2.0	86.7 ± 3.5	77.3 ± 2.6
TL-GNN_ms	91.2 ± 3.9	67.0 ± 7.9	82.1 ± 4.2	53.4 ± 4.7	77.5 ± 3.5	86.2 ± 4.6	78.9 ± 2.3
TL-GNN_mm	90.8 ± 5.4	66.3 ± 7.6	81.0 ± 3.6	52.2 ± 3.8	76.6 ± 3.3	85.6 ± 2.3	77.5 ± 2.6
TL-GNN(w/o S)	92.4 ± 6.3	70.0 ± 7.9	82.2 ± 4.9	54.4 ± 3.0	77.8 ± 2.1	87.8 ± 2.7	79.4 ± 3.0
TL-GNN	$\mathbf{95.7 \pm 3.4}$	$\mathbf{74.4 \pm 4.8}$	83.0 ± 2.1	$\mathbf{55.1 \pm 3.2}$	$\mathbf{79.7 \pm 1.9}$	$\mathbf{88.6 \pm 2.7}$	$\mathbf{79.9 \pm 4.4}$

We found that TL-GNN always achieves the best results on datasets containing sparse graphs. For PTC and COX2, TL-GNN achieves 2.6% and 5.2% margins of improvements over the second-best method. The accuracies of TL-GNN on MUTAG and IMDB-M are 95.7% and 55.1% respectively. This represents a slight but consistent improvement over the alternative methods studied. TL-GNN also achieves the best performance on PROTEINS. Although TL-GNN gives only a slight improvement compared to the second-best method. The average degree of PROTEINS is more than 3. which is dense compared with the remaining datasets. For this kind of dense graph, TL-GNN can capture a large number of subgraphs and enrich the learned information. Even in the cases where TL-GNN does not achieve the best performance, its accuracy is close to that of the best performing method. For NCI1, TL-GNN achieves 0.3% more accuracy than the second best deep learning method.

GIN and TL-GNN have identical GNN layers. However, TL-GNN achieves better performances on most of the datasets. For example, TL-GNN achieves 6.3% and 9.8% improvements on MUTAG and PTC compared to GIN. Moreover, the standard errors for TL-GNN on MUTAG and PTC are lower than those for GIN. NCI1 is the only dataset on which TL-GNN can not surpass the performance of WL. It is worth noting that all deep learning methods are also do not outperform WL on NCI1.

It is worth noting that both GAT and TL-GNN apply the attention mechanism. The difference is that GAT assigns attention weights to the neighbors of nodes. This means that GAT pays different attention to node-level information. Unlike GAT, TL-GNN assigns attention weights to different levels of information (both node-level and subgraph-level). The results show that TL-GNN outperforms GAT on all datasets, and it also achieves significant improvements on several datasets. For example, on NCI and IMDB, TL-GNN achieves $6-9\%$ improvement compared to GAT.

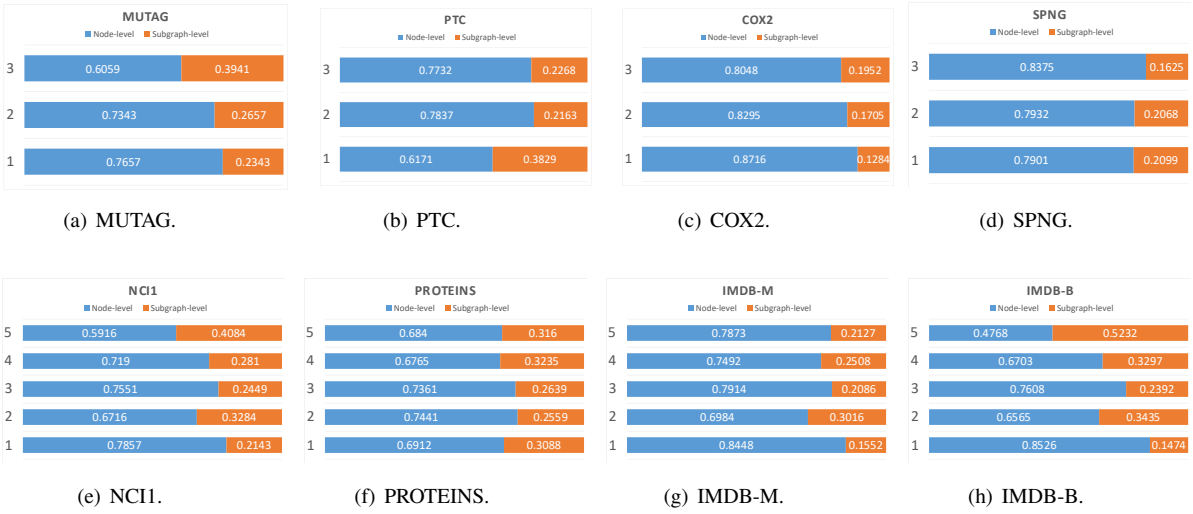
Performance comparison on SPNG

An interesting observation that can be made from Table. II is that the TL-GNN easily achieves 100% training accuracy while the alternative methods studied do not. None of the methods can perfectly fit the training data with the exception of TL-GNN. Due to the fact that both GIN and GCN with K layers can capture $K-$hop information, they do offer some robustness to the LPI problem and can achieve 95.6% and 91.1% training accuracy respectively. However, the remaining GNN training accuracies are no more than 80%. The results on SPNG show that the TL-GNN can distinguish PNGs perfectly. As shown in Fig. 10, the TL-GNN training accuracy after 50 epochs reaches 90%, which represents the fastest convergence rate. Finally, the TL-GNN converged at a stable 100% training accuracy after 220 iterations. The training accuracies of RW-GNN, DGCNN, DAGCN no longer increase after 100 epochs. The reason for this is that these methods focus on capturing the local neighborhood information, which leads to less robustness to the LPI problem.

As for GIN, its convergence rate is slower than the TL-GNN. According to the curve shown in Fig. 10, as the training proceeds, GIN saturated at 95.6%. We believe that the TL-GNN is benefited significantly from the subgraph merging strategy described in Section III-C because both GIN and TL-GNN have the same GNN layers, but TL-GNN's overall performance is better.

Variants of supernode-based subgraph representation

To demonstrate the effectiveness of supernode-based subgraph-level representation used by the proposed methods, we provide a similar TL-GNN architecture, TL-GNN(w/o S) that accepts separated subgraphs as the input sequentially and regards each subgraph as a separated graph to capture its feature, instead of a generated graph containing supernodes. The results of this experiment are shown in Table. I. It is obvious that TL-GNN achieves better performance than TL-GNN (w/o



(a) MUTAG. (b) PTC. (c) COX2. (d) SPNG.

(e) NCI1. (f) PROTEINS. (g) IMDB-M. (h) IMDB-B.

Fig. 7. Variation of the two-level attention parameters on real-world datasets. The number of histograms represents the weight coefficient.

TABLE II
TRAINING ACCURACY ON SYNTEHTIC DATASET(IN % ± STANDARD ERROR)

Model	Training Accuracy
DGCNN	64.0
DAGCN	72.0
GIN	95.6
GCN	91.1
RW-GCN	70.37
TL-GNN	**100**

S) because TL-GNN(w/o S) neglects connections between subgraphs. The reason for this observation is supernode-based subgraph-level representation provides connections between subgraphs, which is beneficial to capture subgraph-level information. However, TL-GNN (w/o S) achieves better performances than other baselines on most datasets, because of the subgraph-level information captured by TL-GNN (w/o S).

Variants of AGG_SUB and MERG

In order to verify the theory described in Section. III-C, we select both non-injective function and injective functions for the AGG_SUB and MERG operators and then compare their performance. We chose the sum function as the injective function. For the non-injective function, we chose the max function. Although there is a large potential choice for the non-injective function, the max is the most convenient for use with our method. The specific TL-GNN variants used are summarised in Table. III. The performances of the four TL-GNN variants are shown in Table. I. As shown in Table. I, TL-GNN achieves a better performance than the remaining variants of TL-GNN on most of the datasets studied. For TL-GNN_mm, on all the datasets except PROTEINS, TL-GNN_mm is the weakest performing TL-GNN variant. This observation demonstrates the correctness of Lemma 6. When AGG_SUB and MERG are not injective functions, the TL-GNN can not effectively capture and retain subgraph-level information. For TL-GNN_ms and TL-GNN_sm, the performance is better than TL-GNN_mm but poorer than TL-GNN. We believe this is because a non-injective function leads to information loss.

TABLE III
DETAILED CONFIGURATION OF TL-GNN VARIANCES

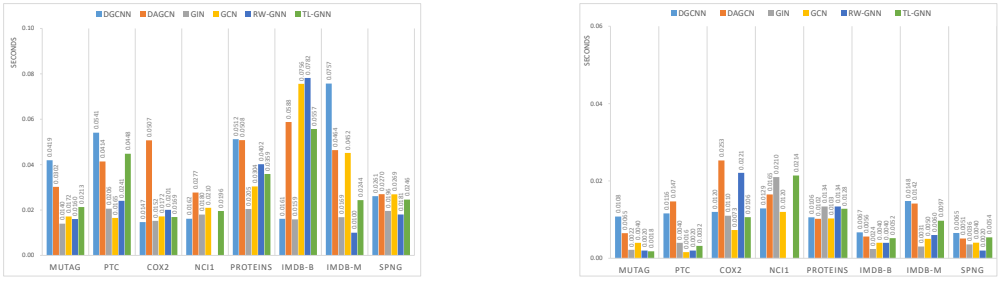
TL-GNN variances	AGG_SUB	MERG
TL-GNN	sum	sum
TL-GNN_sm	sum	max
TL-GNN_ms	max	sum
TL-GNN_mm	max	max

Attention weight visualization and analysis

We provide visualization results of each layer on each dataset in Fig. 7. The blue and orange represent attention weights of node-level and subgraph-level respectively. Numbers of the vertical axis are layers. We applied 3 layers on MUTAG, PTC, COX2, SPNG and 5 layers on NCI1, PROTEINS, IMDB-B, IMDB-M, PROTEINS. An observation is that TL-GNN pays more attention to node-level on all datasets. However, in most cases, the attention weights of subgraph-level increased with the increasing of the layer. A possible reason is that node features become identical with the layer increasing, namely the oversmoothing problem. So, the TL-GNN pays more attention to subgraph-level in high layers. The ratios of node-level attention weight to subgraph-level attention weight are around 7:3. This observation shows that node-level information is more important to the graph classification task and little subgraph-level information is beneficial to graph classification.

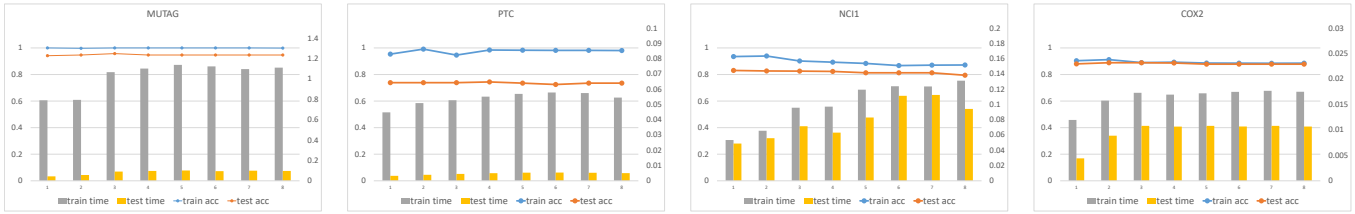
Comparison of computational complexity

We report the average running times per iteration after training or test of TL-GNN and the baselines Fig. 8. For a fair comparison, all of the methods are run on a system with

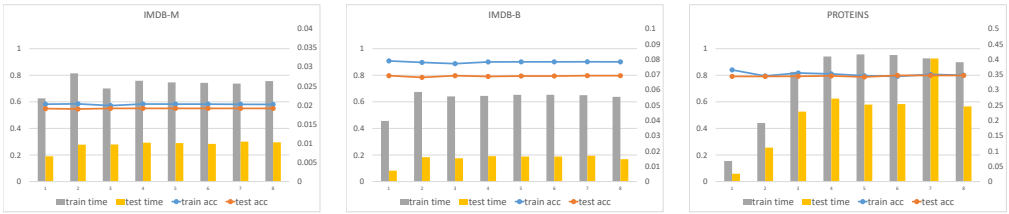


(a) Training time. (b) Test time.

Fig. 8. Running time comparison.



(a) MUTAG. (b) PTC. (c) NCI1. (d) COX2.



(e) IMDB-M. (f) IMDB-B. (g) PROTEINS.

Fig. 9. Variation of the parameter D on real-world datasets. The horizontal axis shows number of D.

an Intel Xeon CPU E3-1270 v5 processor. Due to RW-GNN cannot accept NCI1 as input [39], the running time of RW-GNN on NCI1 dataset is vacant. Obviously, TL-GNN requires more time than GIN because of the extra computation of subgraph-level and attention mechanisms. Although GIN and TL-GNN have identical GNN layers, TL-GNN costs more time on both training and test than GIN. The main reason is that TL requires additional subgraph convolution operations. However, the training and test time of TL-GNN is less than several baselines such as DGCNN and RWGNN, because of the simple but effective architecture of TL-GNN.

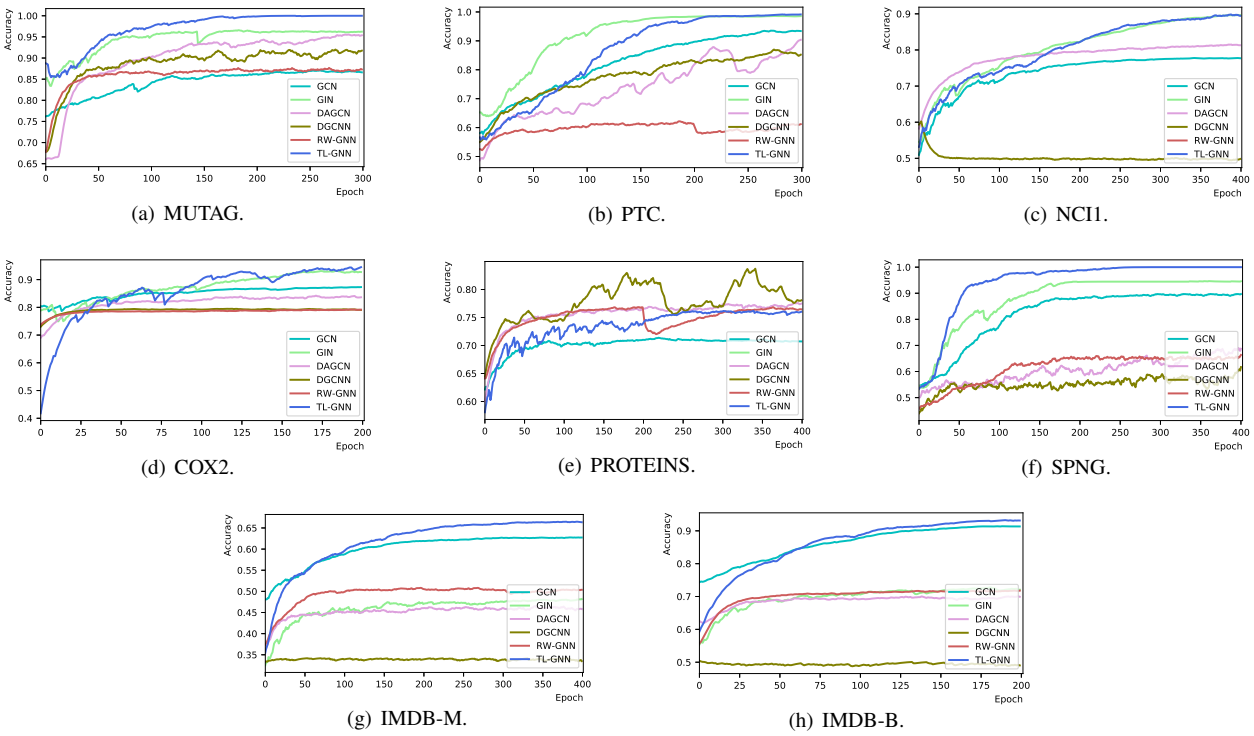
Variation of the depth

In order to find the optimal value of D, we have carried out experiments on all of the real-world datasets, and compute time and accuracies for both training and test are presented. From Fig.9 the compute time consumed in training and testing increases with an increasing value of D on most of the datasets studied. A large value of D leads to a large number of subgraphs and thus increases the compute time required.

When D reaches a critical value, the compute time consumed in both training and testing ceases to increase. This is because when D is large enough, then 2^D is greater than the number of nodes in the graph. This means that all subgraphs within a graph have been located. For example, on the MUTAG dataset, the training becomes stable when D is greater than 5, and this is because the mean node degree of MUTAG is less than 18. For most of the datasets studied, the maximum test accuracy is reached when D is 3 or 4. This indicates that aggregating the subgraph information for the 8-hop or 16-hop neighbors of a node adds the greatest benefits to the graph classification task. It is clear that too small a D value leads to an insufficient number of subgraphs, thus TL-GNN can not capture the subgraph-level information well. Conversely, too large a value of D leads to subgraphs that are distant from the node in question being included. The correlation between these subgraphs and the node in question is low, and this is not conducive to improved performance.

Comparison of training performance

We compare the effect of training for several GNNs for which the authors have made their code available. This study is summarised in Fig. 10. From the table, it is clear that TL-GNN can better fit all of the datasets studied than the



(a) MUTAG. (b) PTC. (c) NCI1.

(d) COX2. (e) PROTEINS. (f) SPNG.

(g) IMDB-M. (h) IMDB-B.

Fig. 10. Training performances comparison.

alternative GNNs. For example, TL-GNN is the only method which achieves 100% training accuracy on MUTAG. The same observation applies to SPNG. As for the NCI1 and PTC datasets, GIN and TL-GNN are the best performing methods, and their training accuracies are roughly identical. On PTC, the convergence speed of TL-GNN is slower than that of GIN, but faster than that of the remaining methods. On the COX2 dataset, the convergence rates of TL-GNN and GIN are close, but the final training accuracy of TL-GNN is slightly higher than that of GIN. On the PROTEINS dataset, the training accuracy of TL-GNN does not outperform that of DGCNN, which is equal to that of the other methods studied. On the two variants of IMDB, TL-GNN and GCN achieve the best training performance. However, TL-GNN is slightly, but consistently better than GCN. We also observe that TL-GNN achieves a higher training accuracy than GIN on many of the datasets studied. Due to the fact that they have the same GNN layer structure, these improvements come from the richer sources of information exploited by TL-GNN. It is worth noting that the convergence speed of TL-GNN on the SPNG dataset is much faster than the remaining deep learning methods. The above observations demonstrate that the subgraph-level information captured by TL-GNN is not only helpful in distinguishing PNGs, but also beneficially enriches the graph representation.

V. Conclusion

In this paper, we presented a novel deep learning method for graph-structured data, the so-called Two-level Graph Neural Network (TL-GNN). Considering the representational limitations on GNNs caused by the LPI problem, we introduce subgraph-level information into our framework and propose two novel operators, AGG_SUB and MERG to implement our method. Moreover, we provide distinct mathematical definitions for permutation non-isomorphic graphs (PNGs) and also provide a theoretical analysis of the role of subgraphs in solving the LPI problem. A novel subgraph counting algorithm is also proposed, which can locate all subgraphs of a $n-$node graph within a $D-$hop neighborhood of each node; this has $O(Dn^3)$ time complexity and $O(Dn^3)$ space complexity. Experimental results show that TL-GNN achieves state-of-the-art performance on most real-world datasets and distinguishes all the data perfectly on the synthetic PNG dataset. As for further work, we plan to extend the method to heterogeneous graphs and further enrich the macroscopic information captured by GNNs. Specifically, we will treat the different types of subgraphs as heterogeneous nodes within a heterogeneous graph and then exploit a discriminative attention mechanism to differentiate between these nodes. Besides, we will further discuss and explore the influence of the LPI problem on the node classification task.

References

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[2] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[3] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *International Conference on Learning Representations (ICLR)*, 2018.

[4] V. Garg, S. Jegelka, and T. Jaakkola, "Generalization and representational limits of graph neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3419–3430.

[5] R. Sato, M. Yamada, and H. Kashima, "Approximation ratios of graph neural networks for combinatorial problems," *In Neural Information Processing Systems (NeurIPS)*, 2019.

[6] J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," *International Conference on Learning Representations (ICLR)*, 2020.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[8] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen, "3d steerable cnns: Learning rotationally equivariant features in volumetric data," *arXiv preprint arXiv:1807.02547*, 2018.

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

[10] M. Koyutürk, A. Grama, and W. Szpankowski, "An efficient algorithm for detecting frequent subgraphs in biological networks," *Bioinformatics*, vol. 20, no. suppl_1, pp. i200–i207, 2004.

[11] C. Jiang, F. Coenen, and M. Zito, "Finding frequent subgraphs in longitudinal social network data using a weighted graph mining approach," in *International Conference on Advanced Data Mining and Applications*. Springer, 2010, pp. 405–416.

[12] J. Bai, Y. Ren, and J. Zhang, "Ripple walk training: A subgraph-based training framework for large and deep graph neural network," *arXiv preprint arXiv:2002.07206*, 2020.

[13] E. Alsentzer, S. G. Finlayson, M. M. Li, and M. Zitnik, "Subgraph neural networks," in *Proceedings of Neural Information Processing Systems, NeurIPS*, 2020.

[14] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[15] G. Jaume, A.-p. Nguyen, M. R. Martınez, J.-P. Thiran, and M. Gabrani, "Edgnn: Asimple and powerful gnn for di-rected labeled graphs," *arXiv preprint arXiv:1904.08745*, 2019.

[16] N. Shervashidze, P. Schweitzer, E. Jan, V. Leeuwen, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 1, pp. 1–48, 2010.

[17] M. Zhou and J. Liu, "A memetic algorithm for enhancing the robustness of scale-free networks against malicious attacks," *Physica A: Statistical Mechanics and its Applications*, vol. 410, pp. 131–143, 2014.

[18] L. Rong and J. Liu, "A heuristic algorithm for enhancing the robustness of scale-free networks based on edge classification," *Physica A: Statistical Mechanics and its Applications*, vol. 503, pp. 503–515, 2018.

[19] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 2005, pp. 8–pp.

[20] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial intelligence and statistics*. PMLR, 2009, pp. 488–495.

[21] J. Ugander, L. Backstrom, and J. Kleinberg, "Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 1307–1318.

[22] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Annual International Conference on Research in Computational Molecular Biology*. Springer, 2007, pp. 92–106.

[23] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," *Advances in neural information processing systems*, vol. 32, p. 9240, 2019.

[24] T. Kloks, D. Kratsch, and H. Müller, "Finding and counting small induced subgraphs efficiently," *Information Processing Letters*, vol. 74, no. 3-4, pp. 115–121, 2000.

[25] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 3, no. 4, pp. 347–359, 2006.

[26] F. Eisenbrand and F. Grandoni, "On the complexity of fixed parameter clique and dominating set," *Theoretical Computer Science*, vol. 326, no. 1-3, pp. 57–67, 2004.

[27] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[28] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[29] C. Gallicchio and A. Micheli, "Ring reservoir neural networks for graphs," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[30] R. Lab, "Protein function prediction via graph kernels," *Oral Radiology*, vol. 6, no. 2, pp. 29–35, 1990.

[31] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[32] Q. Zhao and Y. Wang, "Learning metrics for persistence-based summaries and applications for graph classification," *CoRR*, vol. abs/1904.12189, 2019. [Online]. Available: http://arxiv.org/abs/1904.12189

[33] C. Cai and Y. Wang, "A simple yet effective baseline for non-attributed graph classification," *arXiv preprint arXiv:1811.03508*, 2018.

[34] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[35] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, vol. 2, 06 2012.

[36] 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.

[37] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations (ICLR)*, 2017.

[38] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[39] G. Nikolentzos and M. Vazirgiannis, "Random walk graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 211–16 222, 2020.

[40] H. Peng, J. Li, Q. Gong, Y. Ning, S. Wang, and L. He, "Motif-matching based subgraph-level attentional convolutional network for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5387–5394.