HFGNN: Efficient Graph Neural Networks using Hub-Fringe Structures

Pak Lon Ip
SKL of Internet of Things for Smart City
University of Macau
Macau SAR, China
paklonip@um.edu.mo

Shenghui Zhang
SKL of Internet of Things for Smart City
University of Macau
Macau SAR, China
zhang.shenghui@connect.um.edu.mo

Xuekai Wei
SKL of Internet of Things for Smart City
University of Macau
Macau SAR, China
xuekaiwei@um.edu.mo

Tsz Nam Chan
College of Computer Science and Software Engineering
Shenzhen University
Shenzhen, China
edisonchan@szu.edu.cn

Leong Hou U
SKL of Internet of Things for Smart City
University of Macau
Macau SAR, China
ryanlhu@um.edu.mo

2024 IEEE International Conference on Data Mining (ICDM) | 979-8-3315-0668-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDM59182.2024.00022

***Abstract*—Existing message passing-based and transformer-based graph neural networks (GNNs) cannot satisfy requirements for learning representative graph embeddings due to restricted receptive fields, redundant message passing, and reliance on fixed aggregations. These methods face scalability and expressivity limitations from intractable exponential growth or quadratic complexity, restricting interaction ranges and information coverage across large graphs. Motivated by the analysis of long-range graph structures, we introduce a novel Graph Neural Network called Hub-Fringe Graph Neural Network (HFGNN). Our Hub-Fringe structure, drawing inspiration from the graph indexing technique known as Hub Labeling, offers a straightforward and effective approach for learning scalable graph representations while ensuring comprehensive coverage of information. HFGNN leverages this structure to enable selective propagation of relevant embeddings through a carefully designed message function. Theoretical analysis is presented to show the expressivity and scalability of the proposed method. Empirically, HFGNN exceeds standard GNNs on tasks including classification and regression, especially for large, long-range graphs where scalability and coverage matter. Ablation studies further confirm the benefits of our hub-fringe based graph neural network, including improved expressivity and scalability. The source codes is available at https://github.com/nick12340/HFGNN.**

***Index Terms*—Hub-Fringe, Graph Neural Networks, Expressivity, Indexing structure**

I. Introduction

Graph neural networks (GNNs) provide a powerful framework for learning structural and relational representations of nodes in graphs. The expressiveness of GNNs stems from their ability to efficiently aggregate neighborhood information for each node. These have been extensively used in many application domains to learn node embeddings that capture the structural and relational information in graphs, including social network analysis [1], recommendation systems [2], citation network analysis [3], and life science [4].

Two main GNN training methods are commonly used, which are the message passing [5] and the transformer-based [6], [7] approaches. The message passing approach iteratively aggregates messages (features) for each node from its neighbors to update its embedding in order to propagate information across the graph. The transformer-based approach applies the self-attention mechanism from transformers to learn node embeddings based on the contextual information of the entire graph. Both approaches aim to provide rich representations of all nodes in the graph to support various downstream tasks.

To learn the representation of each node by the message passing-based approach, the core idea is to iteratively obtain the information from its neighbors, e.g., r-hop neighbors ($r = 3$ for the red dashed circle in Figure 1). By setting r to be the diameter of the graph, this approach can fully obtain both the node and edge information of this graph for each node in order to increase the expressivity of the representation. Despite ensuring that information from each node is thoroughly considered, the message passing-based approach necessitates r iterations to effectively learn representations, with each iteration requiring the computation of messages for every edge in the graph E. This process may not be deemed *efficient* for large-scale (or long-range) graphs with a high diameter. To address this issue, research studies [8], [9], [10] combine the message passing-based approach with sampling and clustering techniques to improve efficiency. However, these solutions may not fully capture complete information for each node, potentially reducing the *expressivity* of the representation.

Another approach is to use transformers to learn node embeddings, without message passing. The core idea is to apply multi-head self-attention, allowing each node to obtain

This work was supported by the Science and Technology Development Fund Macau SAR (0003/2023/RIC, 0052/2023/RIA1, 0031/2022/A, 001/2024/SKL for SKL-IOTSC), the Research Grant of University of Macau (MYRG2022-00252-FST), the National Natural Science Foundation of China under Grant No. 62202401, Shenzhen-Hong Kong-Macau Science and Technology Program Category C (SGDX20230821095159012), and Wuyi University joint Research Fund (2021WGALH14). This work was performed in part at SICC which is supported by SKL-IOTSC, University of Macau.

2374-8486/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDM59182.2024.00022

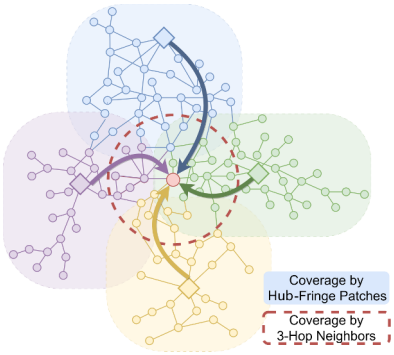


Fig. 1: While expanding the number of hops in a GNN poses challenges for the receptive field to encompass all nodes in the graph (red dashed circle), our method overcomes this limitation by achieving complete coverage through the partitioning of the graph into a Hub-Fringe structure.

information from all other nodes in the graph. Unlike message passing, the transformer strategy avoids the iterative process, but it requires $O(|V|^2)$ space to store the self-attention matrix, which limits its *efficiency* to large graphs. Additionally, it does not explicitly model the edge information (graph structure), relying solely on node co-occurrence for learning embeddings, which restricts its *expressivity* in capturing structural roles and properties of nodes. These issues constrain its performance from explicitly modeling edges between nodes and propagating information in a more structured way for applications that require capturing fine-grained structural node properties.

Hence, we ask a question in this paper. *Can we develop a new GNN structure that provides a highly expressive yet efficient solution?* To provide an affirmative answer to this question. We first propose a new perspective on graph learning problems by formulating the concept of coverage for graph learning tasks and then discuss the expressivity and efficiency issues of existing graph learning methods. Considering the pros and cons of existing methods, we propose a hub-fringe structure (see Figure 1) that aims to achieve high expressivity and high efficiency. Our hub-fringe structure draws inspiration from a graph indexing technique [11] commonly employed for efficient answering of shortest path queries on very large-scale graphs. A two-stage learning framework is then proposed based on the hub-fringe structure. The innovative approach we have employed in our research brings several notable benefits that set it apart from previous studies in the field as illustrated in Table I. (1) Our approach employs *full coverage of node features*, where each node on graph can obtain sufficient information from any its reachable neighbor. (2) We take into account the modeling of *path features along different hubs*, which leads to enhanced expressive capabilities compared to solely considering a single shortest path. For example, in a social network, we can learn about shared interests by analyzing the relationships between users and common follow celebrities. (3) Our method can obtain *exact positional information* (due to the shortest path information) whereas previous approaches can only yield approximate information. (4) Lastly, our approach offers *relatively lower complexity* compared with existing approaches due to the two-stage (hub-fringe) learning framework.

In the rest of this paper, we present a fresh perspective on graph learning problems by introducing the concept of information coverage for graph learning tasks in Section II-A. We then provide a comprehensive review of classic graph learning techniques in Section II-B, emphasizing the challenges pertaining to expressivity and efficiency. We present the hub-fringe structure in Section III-A, followed by an explanation of how we employ the hub labeling technique as the hub-fringe structure in Section III-B. The proposed method, HFGNN, is then introduced in Section III-C. A theoretical analysis of our method is presented in Section III-D, followed by an evaluation of empirical results in Section IV. Lastly, we conclude this paper in Section V.

II. Preliminaries and Related Work

A. Information Coverage

With the growing popularity of GNNs, capturing long-range interactions is crucial for various practical tasks, particularly those involving large graphs or long chain structures [12]. For instance, the chemical property of a molecule [13], [14] depends on the combination of atoms situated on opposite sides. To quantify the extent of expressive power in modeling long-range correlations in graphs, we assess the information coverage provided by graph neural networks using the concept called *receptive field* in deep learning, defined as the input size that produces a feature.

In graph learning, *receptive field* refers to the number of graph nodes contributing to the feature generation process [15], [16], [17]. For example, in a message passing-based GNN, the receptive field encompasses the 2-hop neighbors when the model iterates twice to compute node embeddings. We claim a *receptive field* is *full* for a node u in a graph G (Proposition 1) if the learning process covers all nodes reachable from u in G.

Proposition 1 (Information Coverage)**.** *The embedding representation of node u is considered to achieve information coverage if it incorporates context information from all of its neighboring nodes that are reachable via any simple path. Mathematically, we define this as follows:*

$$\boldsymbol{z}_u = \tau(V_{[u]}, \mathcal{X}_{[u]}) \tag{1}$$

where $V_{[u]}$ represents the induced node set that can be reached from node u in graph G. The embedding function, denoted as τ, maps each node v in $V_{[u]}$ along with its relevant features $\mathcal{X}_{[u]}$ to a representation.

As a remark, there are two representative types of graph features for $\mathcal{X}_{[u]}$, which are path feature and topological feature, The path feature is often collected along the message path during the process of message passing in GNN. The task of finding all paths among a pair of nodes in a graph is known to be a computationally expensive problem that

Properties	Message passing-based GNN	Transformer-based GNN	HFGNN
Node features	*local neighborhood*	*full cover*	*full cover* (Section III-D)
Path features	*neighborhood edges*	*the shortest path*	*multi shortest hub paths* (Section III-C3)
Positional information	*n/a*	*approximate*	*exact* (Section III-C2)
Learning framework	*iterative message passing*	*self attention*	*collect and distribute* (Section III-C1)
Computational complexity	$O(\ r\|V\|d^2 + r\|E\|d\ )$	$O(\ \|V\|d^2 + m\|V\|^2 d\ )$	$O(\ \|V\|d^2 + \ell\|V\|d\ )$ (Section III-D)

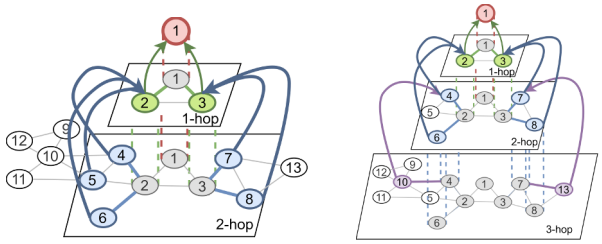
TABLE I: Comparison of the properties of different GNN architectures

requires exponential complexity. However, a specific subset of paths, such as the shortest path, only requires a polynomial cost. Besides, the topological features are employed to complement the structure information as feature enhancements. As an example, the topological feature may include the information that specifies the absolute position of a node inside the graph. It has been shown that models that use distances between nodes as features can provide more expressive power than the 1-WL algorithm [18].

B. Expressivity and Efficiency

Existing GNN methods often encounter practical limitations that hinder their effectiveness. As the scale of graphs increases, the computational demands and memory requirements of these methods become impractical [19]. This presents issues with both expressivity and efficiency. Increasing the expressive power of graph models requires considering more nodes and additional features, resulting in higher computational complexity. As a consequence, this increased complexity presents efficiency challenges for the model.

In this section, we conduct an analysis of two representative graph learning approaches to better illustrate the relationship between expressivity and efficiency.



(a) A 2-layer message passing can cover all information in 2-hop neighborhood of the root node in red.

(b) Sampling sacrifices completeness (v_5, v_{11} not sampled) but increases the coverage range (3-hop layer).

Fig. 2: An illustration of multi-hop message passing.

Message passing-based GNNs. The representative studies in this category include GCN [20], GraphSage [5], DeeperGNN [21], etc. As shown in Figure 2a, according to Proposition 1, to learn the representation of a node u with *complete* neighborhood information, the model should be able to receive information from other nodes at a distance of r, where r indicates the maximum hop distance from u to any nodes in the graph. We should at least stack the layers (the depth of the model) up to r, which results in a computation cost of $|E| \cdot r$. To allow GNNs to be used on large graphs, a practical solution is to sample partial messages from the r-hop neighbors as shown in Figure 2b, which plays a trade-off between computation resources and information coverage. Although many subgraph sampling techniques, such as neighbor sampling [5], layer sampling [22], and sub-graph sampling [8], have been proposed to tackle the neighbor explosion issue, all these approaches still encounter a certain degree of expressivity loss.

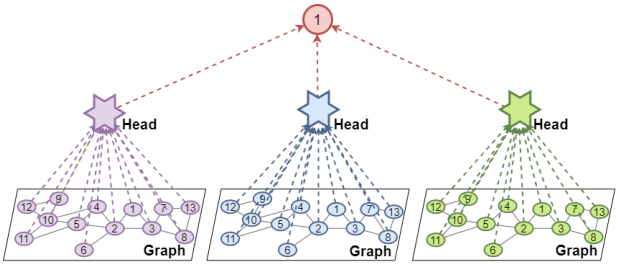


Fig. 3: Each attention head in a self-attention transformer model covers all nodes, causing the model to be costly.

Transformer-based GNNs. The representative studies in this category include SAN [23], GraphTrans [24], Graphormer [25], etc. The self-attention module of a transformer possesses a global receptive field that allows each input token to attend to and process the representation of information at any position; thus, each node in the graph can attend to all other nodes in this model.

According to Proposition 1, graph learning should encompass not only the features of the nodes but also the topological patterns within the graph to provide additional information. Previous studies [26], [25], [27], [28] share the same intuition and have attempted to explicitly encode the correlated topological information for each node to supplement the topological structures and positional encoding as a feature enhancement. Graphormer [26] is one of the transformer-based models that guarantee the information coverage (Proposition 1). It explicitly encodes all edge encoding that encodes all edge features along the shortest path between each pair of nodes.

As the number of graph nodes increases, these methods encounter a quadratic growth in training time and space complexity, resulting in a complexity of $O(|V|^2)$. Besides, the design of multi-head attention [29] allows the model to simultaneously focus on various aspects of the input sequence, enabling it to capture greater nuance and complexity in the data. However, existing transformers on graph [25], [23], [24] do not fully leverage this characteristic, as all the heads receive inputs from all the nodes (see Figure 3).

Several attempts have been made to address the limitations of full attention models and explore linear-time attention approaches. Notably, BigBird [30] and Performer [31] have sought to approximate full attention by employing sparse attention or lower-dimensional matrices. Performers [31] utilize a softmax kernel to approximate attention computation, but this approach often results in a loss of accuracy. Additionally, BigBird [30] was originally designed for sequential inputs, and it faces challenges in effectively generalizing to non-sequential inputs such as graphs. As a result, when applied to graph data in GraphGPS[25], its performance tends to deteriorate (see Table III, V). Exphormer [32] addressed this challenge by introducing modifications tailored to graph inputs, incorporating local attention that utilizes the graph's edge-defined local connectivity. However, Exphormer sacrifices the ability to achieve complete information coverage.

Summary. To the best of our knowledge, no existing method successfully addresses both the information coverage (expressivity) and the computational complexity (efficiency) challenges simultaneously. This drawback motivates us to develop a novel approach to achieve both objectives. We assert that a robust method should possess the potential for expressive power, enabling capturing long-range interactions (for expressivity) while maintaining reasonable complexity for computing and graph embeddings (for efficiency). It is crucial for such a method to operate within the limitations of available GPU memory and computational resources, ensuring practical feasibility in real-world applications.

III. Hub-Fringe Graph Neural Network

A. Hub-Fringe Structure

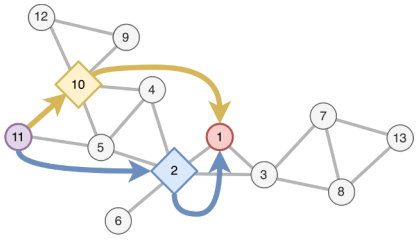


Fig. 4: Hub nodes facilitate the transmission of long range messages. In this case, v_1 obtains the message from v_{11} by utilizing two hub nodes, v_2 and v_{10}, as intermediaries.

One approach to alleviate the complexity of multi-head attention is through utilizing sparse attention mechanisms [31]. It motivates us to allow different attention heads to focus more selectively on various parts of the graph. We expect that the attention head will initially focus on localized information and then progressively transition to incorporating global information, rather than distributing attention equally among all nodes from the outset. To achieve this division strategy, we employ representative nodes in the graph as *hub* nodes and designate their neighboring nodes as the *fringe* (see Figure 4). By adopting this approach, we can significantly reduce computational costs, as the attention mechanism only needs to consider a small local neighborhood surrounding each hub rather than the entire graph. Their respective hub nodes initially collect the fringe messages, ensuring that each hub-fringe patch contains partial graph information.

A *hub* node should hold a significant position within the graph structure, facilitating the aggregation of information from its *fringe* nodes. In the context of graph learning, node properties such as degree can be used to identify the graph hub. For instance, celebrities with many followers can be treated as hubs in a social network, while in a citation network, a groundbreaking paper can be identified by its citation degree. Note that a hub may also serve as the fringe of another hub.

Properties of hub-fringe structure. In addition to selection criteria for hubs, the hub-fringe structure should satisfy certain properties to ensure the efficient recomposition of the full receptive field. These properties guarantee that our model achieves the desired information coverage and preserves the necessary relationships between nodes.

(1) Full receptive field. Every graph node must be a fringe of at least one hub-fringe structure. Therefore, the union of the fringe set of the hub-fringe structure should be equivalent to the graph node set V. This property guarantees that the hub-fringe patches encompass all nodes in the graph to obtain complete information coverage.

(2) Highly concentrated features. We argue that concentrating topological features is crucial, as features tend to experience information decay during a long message passing process. By facilitating messages from the fringe to the hubs along the shortest paths, we can mitigate this decay and preserve important information [33]. This concept aligns with certain transformer-based solutions, such as Graphormer [26], which ensures that edges along the shortest path between each pair of nodes are retained.

(3) Low redundancy. The issue of information redundancy has been discussed in recent works, such as [33], [34], [35], where it is highlighted that redundancy message passing can lead to over-squashing issue. Thus, the selection of the hub aim to minimize redundancy in the fringe nodes, ensuring that each hub captures unique information from the graph.

B. Bridging Graph Learning and Indexing

In this study, we introduce a novel approach that incorporates indexing techniques from graph query answering to construct the hub-fringe structure, aiming to fulfill the aforementioned properties. Graph query answering and graph learning share a common objective of accomplishing tasks with minimal resource utilization. While graph query answering focuses on ensuring query correctness, graph learning aims to achieve learning efficiency, such as information coverage.

Specifically, we construct the hub-fringe structure using the Hub Labeling (HL) technique [11]. HL is a graph indexing technique that optimizes shortest path-finding algorithms, trading space for time compared to Dijkstra's algorithm [36].

Definition III.1 (Hub Label)**.** *For each node $v \in V$, we define the hub label of a node $L(v)$ as a set of pairs $(h, dist)$, where $dist$ is the distance from v to the hub vertex h.*

The fundamental idea behind hub labeling is to pre-compute and store the information about the distances between each vertex and a small set of hubs, nodes with a high degree and many connections to other vertices in the graph. The distance information is stored in a data structure, namely a hub label. To compute the result of the shortest path query from source u to destination v, a sort-merge join is performed between its hub labels $L(u)$ and $L(v)$. The purpose of the sort-merge join is to find a common hub in the shortest path from u to v that is labeled by both u and v.

Property 1 (2-hop cover)**.** *For any pair of reachable nodes $u, v \in V$ of $G(V, E)$, there exists at least one common hub $h \in SP_{u\to v}$ in both label sets, $L(u)$,$L(v)$, such that the shortest path $SP_{u\to v}$ is the result of merging $SP_{u\to h}$ and $SP_{h\to v}$.*

$$dist(SP_{u\to v}) = \min_{h\in L(u)\cap L(v)} \{dist(SP_{u\to h}) + dist(SP_{h\to v})\} \tag{2}$$

In order to ensure the correctness of the shortest path finding, which guarantees the ability to reply to distance requests for every pair of nodes, the hub label must satisfy the 2-hop cover property (see Property 1). An optimization goal in hub label construction is typically to minimize the number of labels, i.e., $\min \sum_{v\in V} |L(v)|$. In [37], it shows that finding a hub labeling with the minimum total label size, while maintaining the 2-hop cover property, is a formulation of the NP-hard weighted set-cover problem [38]. Similar to the idea of solving set-cover problems, many greedy heuristic construction algorithms have been shown to provide small index size in practice [38], [39], [40]. Note that the index size only impacts the query processing time without compromising the accuracy of the shortest path calculation.

Construction of hub labeling. To construct a hub labeling index [39], we follow a sequential process where each node in the graph propagates its distance information based on a predefined node order $\mathcal{O}$. This algorithm initially starts from the highest-order node and iteratively (1) propagates the distance information to its neighbors and (2) assigns the label from the higher-order node u to its lower-order neighbor v. During the propagation of hub node u, the propagation is stopped at node v if the existing label sets $L(u) \cap L(v)$ are already sufficient for computing the shortest path between nodes u and v. This ensures that redundant labels are pruned properly during the propagation process.

Figure 5 illustrates how the construction works. Note that the importance orders $\mathcal{O} = \{v_{10}, v_2, v_3, ...\}$ is given by the degree of the nodes in this example. The propagation process starts with the first hub v_{10}, and the corresponding distance information is added to the label set of nodes with lower ranks. We use small yellow diamonds attached to all nodes to represent the hub v_{10}. For instance, a new hub label $(v_{10}, 3)$ is inserted into $L(v_6)$ as the distance from v_{10} to v_6 is 3. During the propagation of the next hub v_2, it stops at v_{10} since existing label sets $L(v_2) \cap L(v_{10})$ are sufficient to answer the shortest path distance between v_2 and v_{10}.

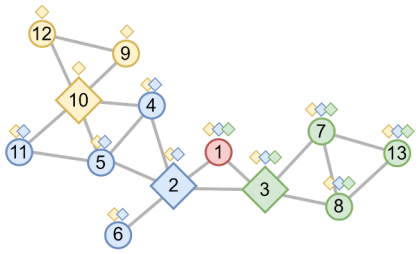


Fig. 5: $L(v_1)$ has three labels from v_{10}, v_2, and v_3. To learn the embedding $\boldsymbol{z}_{v_1}$, we set v_{10}, v_2, and v_3 as hubs, and their corresponding fringe nodes are highlighted with colors.

Following the construction of the hub labeling, the node information can now be propagated from node v_{11} to node v_1 through the intermediate hubs v_{10} and v_2 via a two-step process. One of these hub paths is guaranteed to be the shortest path, as indicated by Property 1. Moreover, it is noteworthy that every node in the graph has the potential to serve as an intermediate hub for other nodes. The determination of which nodes act as hubs primarily relies on the importance order $\mathcal{O}$ and the underlying structure of the graph.

From hub labeling to hub-fringe structure. The hub labeling has the following properties that correspond to the three properties described in Section III-A which make it a good candidate to produce the hub-fringe structure.

(1) 2-hop cover. This property from HL ensures that every shortest path between any node pairs must pass through at least one common hub node. This is crucial for maintaining information completeness.

(2) Shortest path dedication. The main objective of the HL index is to efficiently answer shortest path queries by eliminating unnecessary paths and ensuring that the label set contains the shortest and most unique paths. Such direct message passing enhances the concentration of features and facilitates more effective information exchange between nodes.

(3) Minimality. One common objective in HL algorithms is to minimize redundant information, i.e., minimizing the label size. These algorithms naturally prioritize the preservation of nodes commonly chosen as hubs by many node pairs. This not only ensures the selection of effective hubs but also reduces the level of redundancy in the label set.

C. Hub-Fringe based Graph Neural Network

1) Hub-Fringe based Learning Framework: To provide a clear contrast between our model and the other frameworks for graph learning, we present graph learning in a simplified matrix multiplication format. The r-layer message passing methods on node neighbors can be written as follows.

$$Z = \rho(\mathbb{A}_N ... (\mathbb{A}_N (\mathbb{A}_N \mathcal{X} W_1) W_2) ... W_r) \tag{3}$$

where $\mathbb{A}_N$ is the neighborhood adjacency matrix of a graph, typically provided by the graph of the dataset. The message passing-based methods involve r iterations of matrix multiplication, enabling a node to receive messages from r-hop neighbors (full receptive field). The variable $\mathcal{X}$ denotes the relevant feature of nodes.

The transformer-based methods only require one step since the self-attention affinity matrix $\mathbb{A}_V$ encompasses all pairs of nodes.

$$Z = \rho(\mathbb{A}_V \mathcal{X} W) \tag{4}$$

In contrast to these two classic methods, the hub-fringe structure allows us to achieve Proposition 1 with only a two-stage model as illustrated in Figure 6. The matrix multiplication format of the process can be presented as follows.

$$Z = \rho(\overbrace{\mathbb{A}_L^T(\underbrace{\mathbb{A}_L \mathcal{X} W_1}_{\mathcal{F}\to\mathcal{H}})W_2}^{\mathcal{H}\to\mathcal{F}}) \tag{5}$$

where $\mathbb{A}_L$ indicates the affinity matrix of hub labels (see Definition III.1), i.e.,

$$\mathbb{A}_L(u,h) = \begin{cases} dist, & \text{if } (h, dist) \in L(u) \\ 0, & \text{otherwise} \end{cases}$$

For clarity, the mathematical equations of the two-stage process are shown in message passing fashion as follows. Note that $\mathcal{F}(h)$ and $\mathcal{H}(f)$ are the set representations of the hub label L. Specifically, $\mathcal{H}(f)$ includes all the hub nodes in $L(f)$, whereas $\mathcal{F}(h)$ consists of all the fringe nodes labeled with h in $L(f)$.

$$\begin{aligned} &\text{Fringe} \to \text{Hub} \\ \text{Message:} \quad & m_{f\to h}^k = \phi(\boldsymbol{z}_h^{k-1}, \boldsymbol{z}_f^{k-1}), \forall f \in \mathcal{F}(h) \\ \text{Update:} \quad & \boldsymbol{z}_h^k = \rho(\boldsymbol{z}_h^{k-1}, \oplus(\{m_{f\to h}^k\})) \\ &\text{Hub} \to \text{Fringe} \\ \text{Message:} \quad & m_{h\to f}^k = \phi(\boldsymbol{z}_f^{k-1}, \boldsymbol{z}_h^{k-1}), \forall h \in \mathcal{H}(f) \\ \text{Update:} \quad & \boldsymbol{z}_f^k = \rho(\boldsymbol{z}_f^{k-1}, \oplus(\{m_{h\to f}^k\})) \end{aligned} \tag{6}$$

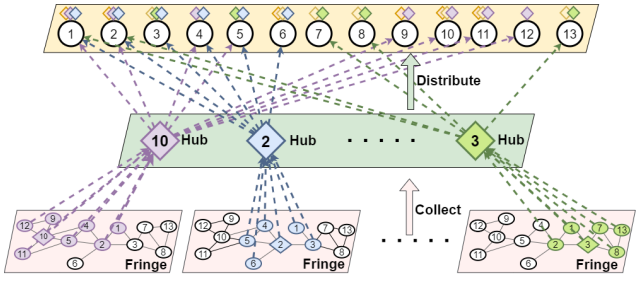


Fig. 6: Hubs are considered transit hubs of the message passing. Each of these hubs first collects the messages from fringes and updates its representation, and then distributes the message back to its fringes.

Collect and Distribute. As depicted in Equation 5, these two stages are a pair of mirroring processes. In the first stage, hubs *collect* information from its fringes using $\mathbb{A}_L$, while in the second stage, hubs *distribute* the aggregated information to its fringes based on the transposed matrix $\mathbb{A}_L^T$.

In the first stage, messages are collected in a message passing fashion. Hubs collect and aggregate messages from all their fringes to update their embeddings. This is achieved by computing an attention-based message embedding for each fringe node via the message function and then aggregating them using a differentiable and permutation-invariant function.

In the second stage, the representation of the hubs contains all the information from their fringes (upper part of Figure 6). It is worth noting that the union of the fringe set is equivalent to the node set due to the 2-hop cover property (Property 1). In the subsequent stage, a *mirror* process is used to update the representation of the fringes, which distributes the messages from the hubs back to the fringes. Each fringe will receive messages from its hub set and update its representation, similar to how the previous stage updated the hubs. The pseudocode of our framework is given in Algorithm 1.

Algorithm 1 Collect and Distribute framework of HFGNN

Input: Hub set $\mathcal{H}$; Fringe set $\mathcal{F}$; Shortest path encoding SPE; Hub label based Positional encoding PE^{HL}; Node input features $\{x_v\}$; Message computation function ϕ; Message aggregation functions $\oplus$; Non-linearity update function σ; Network Layer $l \in [1, K]$;
Output: Embedding z_v for all $v \in \mathcal{V}$.
Hidden layer embedding $\boldsymbol{z}_v \leftarrow x_v$;
for $l = 1,\ldots,K$ **do**
  for $f \in \mathcal{F}, \forall h \in \mathcal{H}(f)$ **do** ▷ Message collecting
    $\phi_{f\to h} = \phi(\boldsymbol{z}_f, \boldsymbol{z}_h, SPE_{f\to h}, PE_f^{HL}, PE_h^{HL})$;
  for $h \in \mathcal{H}$ **do**
    $\phi_h = \oplus(\{\phi_{f\to h}, \forall f \in \mathcal{F}(h)\})$;
    $\boldsymbol{z}_h = \rho(\boldsymbol{z}_h, \phi_h)$;
  for $h \in \mathcal{H}, \forall f \in \mathcal{F}(h)$ **do** ▷ Message distributing
    $\phi_{h\to f} = \phi(\boldsymbol{z}_h, \boldsymbol{z}_f, SPE_{h\to f}, PE_f^{HL}, PE_h^{HL})$;
  for $f \in \mathcal{F}$ **do**
    $\phi_f = \oplus(\{\phi_{h\to f}, \forall h \in \mathcal{H}(f)\})$;
    $\boldsymbol{z}_f = \rho(\boldsymbol{z}_f, \phi_f)$;

Similar to other GNN frameworks [25], our approach also considers several features to enhance expressive power, including (1) node features obtained from data, (2) relative distances of the node pairs, (3) HL-based positional encoding that reflects exact position of each node on the graph denoted as PE^{HL}, and (4) shortest path encoding which contains the edge features from each node pair denoted as SPE. While the first two features can be directly obtained from the input data and query answering in HL, we will focus on the latter two features in the following subsections.

2) HL-based Positional Encoding, PE^{HL}: According to the observation in JK-Net [41], the message passing process differs depending on its position and neighborhood characteristics. This idea has been used in DEGNN [18], Graphormer [26], GraphiT [7], PEG layer [42], and SAN [23].

The basic idea behind PE in graph transformers is similar to that in natural language processing tasks. A fixed vector is added to the embedding of each node in the graph, which encodes the node position on the graph. In this work, we use the hub label to form an encoding vector for each node. For a node u in the graph, its positional encoding is represented by a vector with a length equal to the total number of hubs in the hub-fringe structure. The vector records the distance from u to the labeling hub of $L(u)$. Since we intend to map close nodes on the graph to similar embeddings, a simple normalization is applied to map it to a $[0, 1]$ range.

$$PE_u^{HL}(h) = \begin{cases} \frac{1}{1+dist}, & \text{if } (h, dist) \in L(u), \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Similar to the sort-merge join used in answering shortest path query, the relative distance can be calculated as follows:

$$dist(SP_{u->v}) = \min(PE_u^{HL^{\circ-1}} + PE_v^{HL^{\circ-1}}) - 2$$

where $\cdot^{\circ-1}$ is the Hadamard inverse operator [43] that indicates applying a reciprocal to each vector element.

PE^{HL} captures the information of relative network distance between two nodes with the concrete position on the graph, providing additional expressive power[28]. In addition, PE^{HL} takes a relatively low complexity due to the HL techniques.

3) Shortest Path Encoding, SPE: Edge encoding has been widely used to enhance the performance of node embedding [44], [26], [25]. Our SPE method works by extracting the shortest paths between pairs of nodes in the graph using our hub labeling structure and then encoding these paths as fixed-sized feature vectors. Each path is represented as a sequence of node and edge features, where the node features correspond to the nodes along the path, and the edge features correspond to the edges connecting them.

To effectively incorporate edge features into our graph neural network, we apply a trainable weight when computing the message between node pairs to sum up all the edge features along the shortest path. Note that we only compute and store the shortest path between the hub and the fringe nodes, which can be extracted from hub labeling. This ensures that the SPE process remains efficient across a variety of graphs.

It is important to highlight that our method effectively models the dependencies between node pairs by leveraging the messages from their common labeled hubs. Property 1 ensures that there is always at least one hub positioned along the shortest path connecting the node pairs. This guarantees that our method incorporates at least one SPE of the shortest path between the node pair. Moreover, our approach goes beyond considering a single shortest path by incorporating multiple paths that bypass different hubs. This multi-path perspective provides a richer and more comprehensive understanding of the connectivity between nodes. By considering various paths and their associated hubs, our method captures multiple perspectives and factors into the learning process. This allows us to gain deeper insights into the intricate connectivity patterns and relationships within the graph.

D. Analysis

Information coverage. We formally show that hub-fringe structure can achieve comprehensive information coverage.

Lemma. *HL-based hub-fringe fulfills Proposition 1.*

Proof. As per the 2-hop cover property (Property 1), it is important to note that the union of the fringe set $\mathcal{F}$ from the hub set $\mathcal{H}$ of node v is equivalent to the entire node set V. Mathematically,

$$V \equiv \mathcal{F} = \cup_{h \in \mathcal{H}(v)} \mathcal{F}(h), \forall v \in V \tag{8}$$

This secures that the HL-based hub-fringe structure is reachable to all nodes. Besides, the hub-fringe structure identifies at least one shortest path between any pair of nodes, thus representing their relationship, as stated in Property 1. □

Expressivity. With reduced complexity, it becomes more feasible to incorporate additional feature encodings. Moreover, the tuning space for the hyper parameter is expanded such as hidden size, number of layer and others, allowing for a wider range of choices. In addition to ensuring information coverage from all reachable nodes, our hub-fringe structure empowers the learning process to concentrate on highly informative features, facilitating effective learning while reducing redundancy in the information.

Over-squashing. As discussed in Topping et al. [33], refers to the distortion in message passing caused by the repeated involvement of numerous nodes during the long-range message propagation process. The main conclusion is that, as the distance between two nodes increases, the issues of distortion and over-squashing become more pronounced. One approach to mitigate over-squashing is to *connect all nodes in the graph*. However, this would result in the loss of topological information, which is crucial for message passing in classification tasks, especially for prediction tasks that heavily rely on remote interactions. Achieving *a balance between reducing over-squashing and preserving topological information* presents a significant challenge in graph representation learning. The message passing steps of HFGNN between two nodes are always fixed at two hops, which is much shorter than that of traditional message passing-based GNNs. This proximity between nodes results in a significantly lower distortion compared to neighboring message passing methods.

Efficiency. In message passing-based GNNs, each node collects messages from its one-hop neighbors N, and the affiliation of every node is represented by a matrix $\mathbb{A}_N$ which is the adjacency matrix provided by the graph data. To achieve information coverage (Proposition 1), this process is repeated r times, where r indicates the length of the full receptive field. On the other hand, transformer-based GNNs collect messages from all nodes in one step, and the affiliation is represented by a full matrix $\mathbb{A}_V$. Given the hub-fringe structure, each node u collects messages from its hub labels $L(u)$. Achieving full information coverage requires running the message passing exactly two times, as supported by the 2-hop cover property.

The sparsity between the matrices of message passing, hub-fringe, and transformer can be generally summarized as follows: $|\mathbb{A}_N| \leq |\mathbb{A}_L| \leq |\mathbb{A}_V|$, with $\mathbb{A}_V$ being the superset of the other two matrices (see Equation 8). It is crucial to emphasize that the computation in the learning process is sensitive to the sparsity since we utilize sparse matrix calculations in the learning framework. Interestingly, in our experimental studies, the sparsity of $\mathbb{A}_L$ in "COCO-SP" is even better than that of $\mathbb{A}_N$, showcasing the hub-fringe structure's advantage. Furthermore, the numbers of iterations for these three approaches are as follows: r (message passing) > 2 (hub-fringe) > 1 (transformer).

Obviously, the hub-fringe structure excels when the average size of the hub label set ($\mathbb{A}_L$) is small. As reported in numerous studies [40], [39], the label size ℓ of a node is much smaller than $|V|$ in practice. For instance, there are typically hundreds of labels per node, even in large-scale graphs containing millions of nodes [39].

Complexity Analysis. We denote the number of nodes as $|V|$, the number of edges as $|E|$, the hub label size per node as ℓ, the size of feature encoding vectors as d, the number of attention heads as m.

The computation of a GNN consists of two parts: (1) the linear transformation on the input embeddings and (2) the message-aggregation framework to generate the output embeddings. The first part is a fundamental operation in neural networks that aligns the dimensions of different features for each node so the complexity is $O(|V|d^2)$. The second part involves learning the relationship between the input embeddings and output embeddings through the interaction of graph nodes. In this work, our main contribution lies in reducing the complexity of the second part.

In HFGNN, each hub collects information from all corresponding fringes and subsequently distributes the aggregated information back to all fringes. It is important to highlight that attention, similar to transformer-based GNNs, is employed to learn the embedding relationship between a hub and its fringes. Considering that the total number of hubs is bounded by $|V|$ and the number of fringes per hub is bounded by ℓ, the complexity of our HFGNN approach is $O(\ell|V|d)$, as it only runs the same learning process twice. Note that a space with a complexity of $O(\ell|V|)$ is required to store the sparse hub-fringe relationship. The overall complexity of the system can be expressed as the sum of two components, resulting in a total complexity of $O(|V|d^2 + \ell|V|d)$.

IV. Experiments

This section evaluates the effectiveness of the HFGNN for graph tasks, and our implementation is developed based on the GraphGym [45] module of Pytorch-Geometric [46]. A computation resource is a single machine with an NVIDIA RTX3090 GPU with 24GB GPU memory and an AMD Ryzen Threadripper 3960X CPU with 24 cores and 64GB RAM. The source codes and relevant settings can be found at https://github.com/nick12340/HFGNN. The evaluation of baselines in this study encompasses a range of popular GNN methods, categorized into message-passing based models and transformer-based models.

A brief description of these benchmark datasets is introduced as follows. Table II summarizes the statistics of the datasets. It should be noted that the indexing time refers to the cumulative duration required for building the hub labeling for each individual graph inside a given dataset. It has been demonstrated that the computing cost associated with preprocessing the graph index is relatively inconspicuous in comparison to the time required for model training.

TABLE II: Statistics and description of the evaluated datasets

Dataset	$\|G\|$	$\|V\|$	$\|E\|$	ℓ	indexing (sec.)	prediction
ZINC	12,000	23.2	24.9	4.64	46.4	graph
PATTERN	10,000	118.9	6,098.9	29.13	428.4	inductive node
CLUSTER	10,000	117.2	4,303.9	25.03	394.2	inductive node
ogbg-molhiv	41,127	25.5	27.5	5.10	11.4	graph
ogbg-molpcba	437,929	26	28.1	4.93	127.1	graph
ogbg-ppa	158,100	243.4	2,266.1	15.85	1532.5	graph
ogbg-code2	452,741	125.2	124.2	4.84	499.3	graph
PascalVOC-SP	11,355	479.4	2,710.5	20.727	566.5	inductive node
COCO-SP	123,286	476.9	2,693.7	21.32	6194.4	inductive node
Peptides-func	15,535	150.94	307.3	17.83	258.9	graph
Peptides-struct	15,535	150.94	307.3	17.83	258.9	graph

The selected baselines from the message-passing based category include GCN [20], GAT [6], GIN [47], PAN [48], and DiffPool-GCN [9]. The transformer-based baselines consist of SAN [23], Graphormer [26], GraphGPS [25], Exphormer [32], and GraphTrans [24].

The best results are reported from the original papers for some methods, as some of the hyperparameter configurations in previous works are not publicly available. In the tables, we highlight the methods with a star * to indicate that the results are from the original papers. We also include information about the hardware used in all the experiments, including the available GPU memory, to provide a reference for the readers.

TABLE III: GNN Benchmark performance(mean ± std%. Best results are colored in **first**, **second**, **third**).

Method	Hardware	ZINC	PATTERN	CLUSTER
		MAE ↓	Accuracy ↑	Accuracy ↑
*GCN	1 × V100 32GB	0.367±0.011	71.892±0.334	68.498±0.976
*GAT	1 × V100 32GB	0.384±0.007	78.271±0.186	70.587±0.447
*GIN	1 × V100 32GB	0.526±0.051	85.387±0.136	64.716±1.553
*PAN	1 × V100 32GB	0.188±0.004	-	-
*SAN	1 × V100, 32GB	0.139±0.006	86.581±0.037	76.691±0.650
*Graphormer	8 × V100 32GB	**0.122±0.006**	-	-
*GraphGPS	1 × A100 40GB	**0.070±0.004**	**86.685±0.059**	**78.016±0.180**
GraphGPS(BigBird)	1 × 3090 24GB	0.130±0.012	86.049±0.014	76.827±0.180
*Exphormer	1 × A100 40GB	-	**86.734±0.008**	**78.070±0.037**
HFGNN	1 × 3090 24GB	**0.090±0.001**	**86.821±0.026**	**78.863±0.032**

A. Experimental Analysis

Table III compares the performance of graph neural networks on three datasets: ZINC, PATTERN, and CLUSTER. The evaluation metrics are mean absolute error (MAE) for ZINC and accuracy for PATTERN and CLUSTER. The proposed HFGNN model achieves a low Mean Absolute Error (MAE) of 0.09 on the ZINC dataset, surpassing all baseline methods except GraphGPS, which achieves a slightly better MAE of 0.070. For PATTERN and CLUSTER, HFGNN

TABLE IV: Open Graph Benchmark performance(mean ± std%. Best results are colored in **first**, **second**, **third**).

Method	ogbg-molhiv AUROC ↑	ogbg-molpcba Avg. Precision ↑	ogbg-ppa Accuracy ↑	ogbg-code2 F1 score ↑
*GCN	0.759±0.006	0.232±0.003	0.690±0.007	0.159±0.002
*GAT	0.761±0.003	0.217±0.008	0.684±0.009	0.157±0.001
*GIN	0.771±0.015	0.270±0.002	0.704±0.011	0.158±0.003
*PAN	0.774±0.009	0.262±0.006	**0.768±0.012**	0.164±0.022
DiffPool-GCN	0.766±0.009	0.244±0.008	0.725±0.008	0.144±0.004
*SAN	0.779±0.247	0.277±0.004	-	-
*GraphTrans	-	0.276±0.003	-	**0.183±0.002**
*Graphormer	**0.805±0.530**	**0.314±0.320**	-	-
*GraphGPS	**0.788±0.010**	**0.291±0.003**	**0.802±0.003**	**0.189±0.002**
HFGNN	**0.797±0.046**	**0.295±0.004**	**0.788±0.005**	**0.192±0.001**

TABLE V: Long Range Graph Benchmark performance(mean ± std%. Best results are colored in **first**, **second**, **third**).

Method	PascalVOC-SP F1 score ↑	COCO-SP F1 score ↑	Peptides-func AP ↑	Peptides-struct MAE ↓
*GCN	0.127±0.006	0.084±0.001	0.593±0.002	0.350±0.0011
*GatedGCN	0.287±0.022	0.264±0.005	0.586±0.008	0.342±0.001
*SAN	0.323±0.004	0.259±0.016	0.638±0.012	0.268±0.004
*TransF.+LapPE	0.269±0.010	0.262±0.003	0.633±0.013	0.253±0.002
*GraphGPS	**0.375±0.011**	**0.341±0.004**	**0.654±0.004**	**0.250±0.001**
GraphGPS(BigBird)	0.276±0.007	0.263±0.007	0.585±0.008	0.253±0.003
*Exphormer	**0.398±0.004**	**0.346±0.001**	**0.656±0.004**	**0.248±0.001**
HFGNN	**0.385±0.027**	**0.313±0.008**	**0.668±0.003**	**0.247±0.002**

achieves the highest accuracy of 86.821% and 78.863%, respectively, surpassing all models. Compared to more resource-intensive models, such as Graphormer, which runs on multiple high-end GPUs, HFGNN efficiently achieves better or comparable results on a single consumer-grade GPU. The proposed hub-fringe framework demonstrates its advantages by efficiently learning graph representations while ensuring sufficient information coverage.

Furthermore, Table IV presents a comparison of GNN models on four Open Graph Benchmark datasets. On ogbg-molhiv, Graphormer achieves the best AUROC of 0.805. However, the proposed HFGNN obtains a competitive AUROC of 0.797, surpassing all baseline GNNs. For ogbg-molpcba, Graphormer again shows the top average precision of 0.314, while HFGNN attains a close second at 0.295, outperforming other models. On ogbg-ppa, HFGNN obtains an accuracy of 0.788, surpassing strong baselines like GIN and PAN. Finally, for ogbg-code2, HFGNN achieves the highest F1 score of 0.192, significantly exceeding GNN baselines and GraphTrans. It is worth noting that the resources required by Graphormer on larger datasets are not detailed in [26], even though it shows top performance on chemical datasets. The resource efficiency of HFGNN is advantageous in handling larger graphs.

We also analyze the performance over long-range graph benchmarks shown in Table V. Again, HFGNN outperforms or remains competitive with state-of-the-art techniques on these long-range graph tasks. On PascalVOC-SP, the proposed HFGNN achieves an F1 score of 0.385, significantly outperforming prior works, including SAN, Transformer + LapPE, and GraphGPS. For COCO-SP, Exphormer shows the best F1 of 0.341, but HFGNN obtains a competitive 0.313, exceeding other methods. For Peptides-func, HFGNN has the top AP of 0.668, while Exphormer achieves a close second at 0.656, surpassing SAN and Transformer baselines. Finally, on Peptides-struct, HFGNN attains the lowest MAE of 0.247.

V. Conclusion

This work introduces a novel graph learning framework called HFGNN, based on the hub-fringe graph structure. Experimental results demonstrate that HFGNN represents an advancement in graph neural networks. In particular, HFGNN addresses GNN expressivity and efficiency challenges by introducing an efficient message passing framework for graph learning that employs the hub-labeling method. By attaining full coverage in message passing, HFGNN enables more expressive graph representations, resulting in more accurate predictions. The theoretical and empirical analysis of the proposed model demonstrates its ability to outperform other state-of-the-art models, indicating its potential as a fundamental solution to the fundamental problem of expressivity and efficiency in graph neural networks. HFGNN has a wide range of potential applications, and its ability to manage large-scale graphs efficiently makes it a promising solution for various real-world problems. In the future, we will investigate how the structure-driven GNN performs in configurations with large memory, enabling training of large-scale models.

References

[1] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] W. Fan, Y. Ma, Q. Li, Y. He, Y. E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, Eds. ACM, 2019, pp. 417–426.

[3] L. Waikhom and R. Patgiri, "A survey of graph neural networks in various learning paradigms: methods, applications, and challenges," *Artif. Intell. Rev.*, vol. 56, no. 7, pp. 6295–6364, 2023.

[4] Z. Wang, V. N. Ioannidis, H. Rangwala, T. Arai, R. Brand, M. Li, and Y. Nakayama, "Graph neural networks in life sciences: Opportunities and solutions," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 4834–4835.

[5] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 1024–1034.

[6] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[7] G. Mialon, D. Chen, M. Selosse, and J. Mairal, "Graphit: Encoding graph structure in transformers," *CoRR*, vol. abs/2106.05667, 2021.

[8] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[9] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *Advances in neural information processing systems*, vol. 31, 2018.

[10] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[11] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[12] V. P. Dwivedi, L. Rampášek, M. Galkin, A. Parviz, G. Wolf, A. T. Luu, and D. Beaini, "Long range graph benchmark," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 326–22 340, 2022.

[13] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, no. 1, pp. 1–7, 2014.

[14] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[15] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[16] X. Ma, J. Wang, H. Chen, and G. Song, "Improving graph neural networks with structural adaptive receptive fields," in *Proceedings of the Web Conference 2021*, 2021, pp. 2438–2447.

[17] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4424–4431.

[18] P. Li, Y. Wang, H. Wang, and J. Leskovec, "Distance encoding: Design provably more powerful neural networks for graph representation learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4465–4478, 2020.

[19] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "OGB-LSC: A large-scale challenge for machine learning on graphs," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, J. Vanschoren and S. Yeung, Eds., 2021.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[21] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 338–348.

[22] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[23] 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, pp. 21 618–21 629, 2021.

[24] Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 266–13 279, 2021.

[25] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 501–14 515, 2022.

[26] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 877–28 888, 2021.

[27] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3469–3489.

[28] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[31] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[32] H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop, "Exphormer: Sparse transformers for graphs," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 613–31 632.

[33] J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[34] R. Chen, S. Zhang, L. H. U, and Y. Li, "Redundancy-free message passing for graph neural networks," in *NeurIPS*, 2022.

[35] Q. Sun, J. Li, B. Yang, X. Fu, H. Peng, and S. Y. Philip, "Self-organization preserved graph structure learning with principle of relevant information," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 4643–4651.

[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, pp. 269–271, 1959.

[37] M. Babenko, A. V. Goldberg, H. Kaplan, R. Savchenko, and M. Weller, "On the complexity of hub labeling," in *Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II 40*. Springer, 2015, pp. 62–74.

[38] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings 20*. Springer, 2012, pp. 24–35.

[39] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 349–360.

[40] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 445–457, 2017.

[41] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[42] H. Wang, H. Yin, M. Zhang, and P. Li, "Equivariant and stable positional encoding for more powerful graph neural networks," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[43] R. Reams, "Hadamard inverses, square roots and products of almost semidefinite matrices," *Linear Algebra and its Applications*, vol. 288, pp. 35–43, 1999.

[44] D. Mesquita, A. Souza, and S. Kaski, "Rethinking pooling in graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2220–2231, 2020.

[45] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," in *NeurIPS*, 2020.

[46] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[47] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[48] 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.