

State of the Art and Potentialities of Graph-level Learning

ZHENYU YANG, GE ZHANG, JIA WU, JIAN YANG, QUAN Z. SHENG, SHAN XUE, Macquarie University, Australia

CHUAN ZHOU, Chinese Academy of Sciences, China

CHARU AGGARWAL, IBM T. J. Watson Research Center, USA

HAO PENG, Beihang University, China

WENBIN HU, Wuhan University, China

EDWIN HANCOCK, University of York, United Kingdom

PIETRO LIÒ, University of Cambridge, United Kingdom

Graphs have a superior ability to represent relational data, like chemical compounds, proteins, and social networks. Hence, graph-level learning, which takes a set of graphs as input, has been applied to many tasks including comparison, regression, classification, and more. Traditional approaches to learning a set of graphs heavily rely on hand-crafted features, such as substructures. But while these methods benefit from good interpretability, they often suffer from computational bottlenecks as they cannot skirt the graph isomorphism problem. Conversely, deep learning has helped graph-level learning adapt to the growing scale of graphs by extracting features automatically and encoding graphs into low-dimensional representations. As a result, these deep graph learning methods have been responsible for many successes. Yet, there is no comprehensive survey that reviews graph-level learning starting with traditional learning and moving through to the deep learning approaches. This article fills this gap and frames the representative algorithms into a systematic taxonomy covering traditional learning, graph-level deep neural networks, graph-level graph neural networks, and graph pooling. To ensure a thoroughly comprehensive survey, the evolutions, interactions, and communications between methods from four different branches of development are also examined. This is followed by a brief review of the benchmark data sets, evaluation metrics, and common downstream applications. The survey concludes with a broad overview of 12 current and future directions in this booming field.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → **Data mining**.

Additional Key Words and Phrases: graph-level learning, graph datasets, deep Learning, graph neural networks, graph pooling.

ACM Reference Format:

Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Shan Xue, Chuan Zhou, Charu Aggarwal, Hao Peng, Wenbin Hu, Edwin Hancock, and Pietro Liò. 2023. State of the Art and Potentialities of Graph-level Learning. 1, 1 (May 2023), 52 pages. <https://doi.org/XXXXXXXX.XXXXXXX>

Authors' addresses: Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Shan Xue, Macquarie University, Sydney, Australia, zhenyu.yang3@hdr.mq.edu.au, ge.zhang5@hdr.mq.edu.au, jia.wu@mq.edu.au, jian.yang@mq.edu.au, michael.sheng@mq.edu.au, emma.xue@mq.edu.au; Chuan Zhou, Chinese Academy of Sciences, Beijing, China, zhouchuan@amss.ac.cn; Charu Aggarwal, IBM T. J. Watson Research Center, New York, USA, charu@us.ibm.com; Hao Peng, Beihang University, Beijing, China, penghao@buaa.edu.cn; Wenbin Hu, Wuhan University, Wuhan, China, hwb@whu.edu.cn; Edwin Hancock, University of York, York, United Kingdom, edwin.hancock@york.ac.uk; Pietro Liò, University of Cambridge, Cambridge, United Kingdom, Pietro.Lio@c1.cam.ac.uk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Association for Computing Machinery.

XXXX-XXXX/2023/5-ART \$15.00

<https://doi.org/XXXXXXXX.XXXXXXX>

1 INTRODUCTION

Research into graph-structured data started with the Königsberg bridge problem [1] in the 18th century, that is: “How can we design a path among seven bridges in Königsberg city that crosses each bridge only once?” Through modeling seven bridges into a graph in which nodes represent the junctions between bridges and edges represent bridges, the Königsberg bridge problem is proved unsolvable. Since then, graph-structured data has become an indispensable tool for exploring the world. In reality, researchers can model millions of molecules in which each presents a graph to analyze molecular properties [2]. Such case learning the underlying semantics among a set of graphs is graph-level learning.

Mining the underlying rules among a set of graphs is tough hard as graphs are irregular with an unfixed number of disordered nodes and varied structural layouts. A long-standing challenge in graph-level learning, the graph isomorphism problem, is “How to determine whether two graphs are completely equivalent or isomorphic?” An enormous number of studies [3–5] focused on this question and concerned it as a candidate for NP-immediate until a quasi-polynomial-time solution was proposed in 2016 [6]. To tackle the struggle in this area, tremendous efforts have been made involving traditional methods and deep learning.

Generally, traditional graph-level learning builds the architecture upon handcrafted features (e.g., random walk sequences [7], frequently occurring substructure [8]) and classical machine learning techniques (e.g., support vector machine). This paradigm is human-interpretable but is usually restricted to simple small graphs rather than reality large networks. This is because traditional methods cannot bypass the graph isomorphism problem, the predefined features require to preserve the isomorphism between graphs, i.e., mapping isomorphism graphs to the same features. On the contrary, deep learning techniques break the shackles by training the network to automatically learn non-linear and low-dimensional features. This makes deep neural networks bring new benchmarks for state-of-the-art performance and support the ever-increasing size of graph data. The fly in the ointment is the black-box nature of deep learning, which leads to compromised trustworthiness. An emerging trend is to develop reliable graph-level learning techniques that own the advantages of neural networks and traditional methods.

Benefiting from these techniques, graph-level learning has applications and promise in many fields. Wang *et al.* [9] took graphs of molecules, where the nodes denote atoms and the edges represent chemical bonds, and performed graph regression as a way of predicting molecular properties to help discover more economical crystals. In another study, a graph generation task based on a series of protein graphs was used to produce graphs of proteins with specific functions to support drug discovery [10]. Likewise, graph classification with brain graphs has the potential to distinguish brain structures with neurological disorders from those of healthy individuals [11].

The success of applications qualifies the huge potential of graph-level learning, which raises a practical demand to comprehensively survey this field spanning both traditional and deep learning within the vast amount of literature. There are surveys on learning graph-structured data. However, these reviews suffer from two main disadvantages. First, most existing surveys concentrate on articles that explore the node/edge/substructures in a single graph, such as network embedding [12], community detection [13, 14], anomaly detection [15], and graph neural networks [16, 17]. Graph-level learning is treated as a by-product taking up a subsection or less. The differences between graph learning on a single graph and graph-level learning are illustrated in Fig. 1. Second, graph-level learning is only investigated from a single perspective, such as graph kernels [18] or graph pooling [19]. As such, the surveys have not covered a broad width and overlook the

¹Two graphs \mathcal{G}_1 and \mathcal{G}_2 are isomorphic if the following two conditions are met: (1) There exists matching between nodes in \mathcal{G}_1 and \mathcal{G}_2 ; (2) Two nodes are linked by an edge in \mathcal{G}_1 iff the corresponding nodes are linked by an edge in \mathcal{G}_2 .

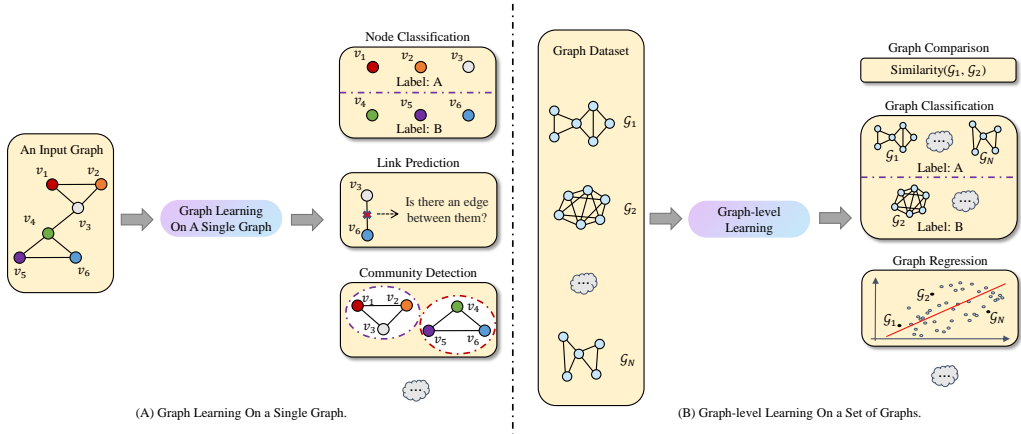


Fig. 1. Toy examples of graph learning on a single graph and graph datasets.

interactions between different graph-level learning techniques, e.g., adopting traditional techniques to empower GL-GNNs (see sections 6.2 and 6.3).

To the best of our knowledge, this is the first comprehensive survey of graph-level learning that spans both traditional methods and deep learning-based techniques (i.e. GL-DNNs, GL-GNNs, and graph pooling). This article exhaustively depicts the mainstream techniques in different periods of graph-level learning (see Fig. 2), and further discusses the evolutions, interactions, and communications between them. Thus, the contributions of this survey include:

- **A comprehensive taxonomy:** We propose a comprehensive taxonomy for graph-level learning techniques. Specifically, our taxonomy covers graph-level learning through both traditional and deep learning methods.
- **An in-depth review:** Over four categories, we summarize the representative algorithms, make comparisons, and discuss the contributions and limitations of existing methods.
- **Abundant resources:** This survey provides readers with abundant resources of graph-level learning, including information on the state-of-the-art algorithms, the benchmark datasets for different domains, fair evaluation metrics for different graph-level learning tasks, and practical downstream applications. The repository of this article is available at <https://github.com/ZhenyuYangMQ/Awesome-Graph-Level-Learning>.
- **Future directions:** We identify 12 important future directions in the graph-level learning area.

2 DEFINITIONS

This section, provides some definitions that are essential to understanding this paper. Bold lowercase characters (e.g., \mathbf{x}) are used to denote vectors. Bold uppercase characters (e.g., \mathbf{X}) are used to denote matrices. Plain uppercase characters (e.g., \mathcal{V}) are used to denote mathematical sets, and lowercase italic characters (e.g., n) are used to denote constants.

Definition 2.1. (Graph): A graph can be denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set \mathcal{V} having n nodes (also known as vertices) and the edge set \mathcal{E} having m edges. In an undirected graph, $\mathcal{E}_{u,v} = \{u, v\} \in \mathcal{E}$ represents that there is an edge connecting nodes u and v , where $u \in \mathcal{V}$ and $v \in \mathcal{V}$. If \mathcal{G} is unweighted, we use an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ to describe its topological structure, where $\mathbf{A}_{u,v} = 1$ if $\mathcal{E}_{u,v} \in \mathcal{E}$, otherwise, 0. If \mathcal{G} is weighted, the value of $\mathbf{A}_{u,v}$ refers to the

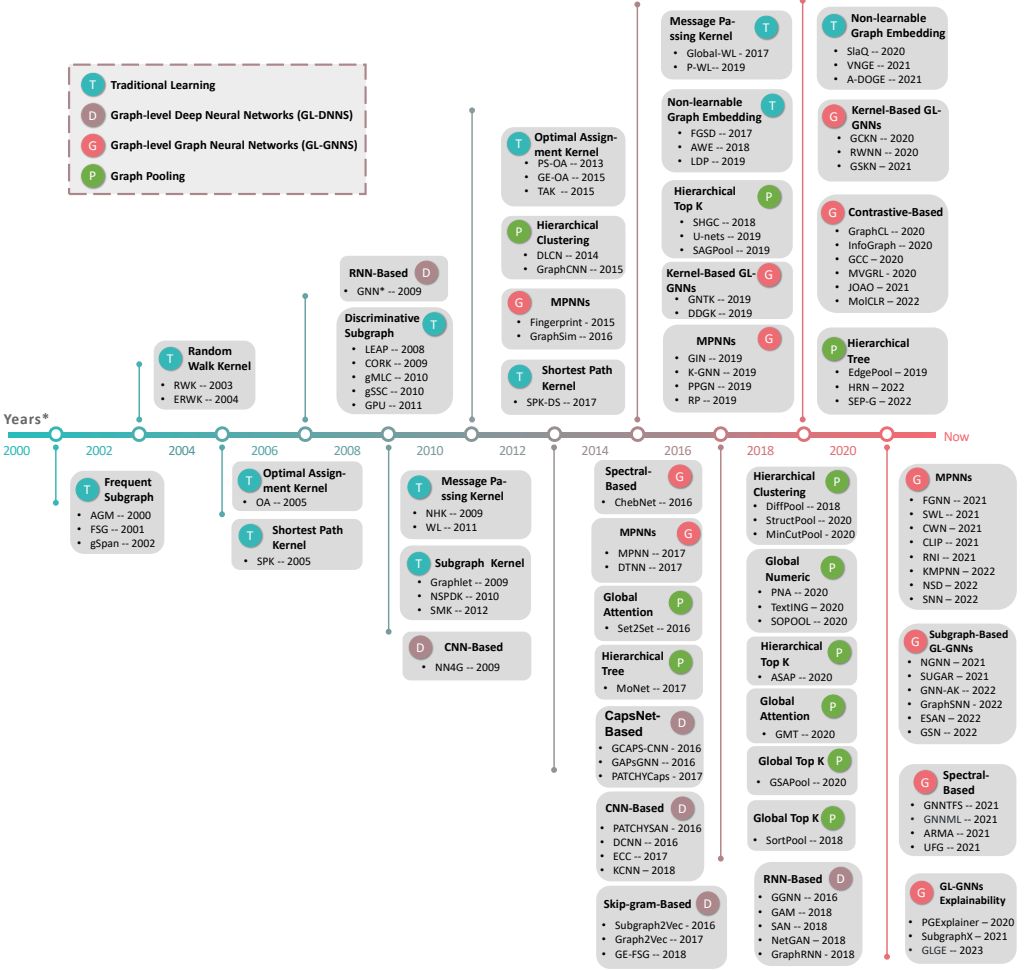


Fig. 2. The timeline of graph-level learning in terms of four mainstream techniques.

weight value of $\mathcal{E}_{u,v}$. $\mathbf{X} \in \mathbb{R}^{n \times f}$ is the node attribute matrix and a node $u \in \mathcal{V}$ can be described by an attribute vector $\mathbf{x}_u \in \mathbb{R}^f$. Similarly, the edge feature matrix is denoted as $\mathbf{S} \in \mathbb{R}^{m \times d}$, where $\mathbf{s}_{u,v} \in \mathbb{R}^d$ describes the edge $\mathcal{E}_{u,v} \in \mathcal{E}$. Unless otherwise specified, the graphs in this paper are undirected attributed graphs.

Definition 2.2. (Graph Dataset): A graph dataset \mathbb{G} is composed of N graphs, where $\mathbb{G} = \{\mathcal{G}_1, \dots, \mathcal{G}_N\}$.

Definition 2.3. (Subgraph/Substructure): A graph $g_m = (\mathcal{V}_{g_m}, \mathcal{E}_{g_m})$ can be regarded as the subgraph/substructure of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ iff there exist an injective function $\phi : \mathcal{V}_{g_m} \rightarrow \mathcal{V}$ s.t. $\{u, v\} \in \mathcal{E}_{g_m}$ and $\{\phi(u), \phi(v)\} \in \mathcal{E}$.

Definition 2.4. (Graph-level Learning): Graph-level learning takes a graph dataset $\mathbb{G} = \{\mathcal{G}_1, \dots, \mathcal{G}_N\}$ consisting of N graphs as inputs and returns a function $f(\cdot)$ which maps a graph \mathcal{G}_i to some output

$f(\mathcal{G}_i)$. For instance, in the graph classification task, for any graph \mathcal{G}'_i isomorphic to \mathcal{G}_i , we have $f(\mathcal{G}_i) = f(\mathcal{G}'_i)$. In other words, $f(\cdot)$ is permutation-invariant².

3 TAXONOMY OF GRAPH-LEVEL LEARNING TECHNIQUES

This section provides a taxonomy of graph-level learning techniques. Its categories include traditional learning, graph-level deep neural networks (GL-DNNs), graph-level graph neural networks (GL-GNNs), and graph pooling. Each category is briefly introduced next. The taxonomy tree describing these four branches of graph-level learning with selected algorithms can be found in Fig. 8 in Appendix A.

Traditional Learning. As the historically dominant technique, traditional learning tries to solve the fundamental problem that is lacking feature representations of graphs, by manually defined features. Given well-designed features (e.g., random walk sequences [7], frequently occurring substructure [8]), off-the-shelf machine learning models were used to tackle graph classification tasks, in a non-end-to-end fashion. The form of traditional learning is less applicable to reality complex networks due to the computational bottlenecks, yet, it still provides great valuable insights, such as better interpretability and better ability to model irregular structures [20].

Graph-Level Deep Neural Networks (GL-DNNs). Towards the deep learning era, neural networks achieved wide success in representing Euclidean data (e.g., images and texts). Thus, researchers try to apply deep neural networks to graph data, the tentative explorations include Skip-gram, RNNs, CNNs, and CapsNet. These four types of deep neural networks were not initially designed to learn non-Euclidean data like graphs. Hence, one of the important issues with GL-DNNs is how to enable these deep neural networks to learn graph-structured data that varies in size and has irregular neighborhood structures.

Graph-Level Graph Neural Networks (GL-GNNs). GL-GNNs use graph convolution operations specifically proposed for graphs as the backbone for performing graph-level learning [16]. Most GL-GNNs use the graph convolutions MPNNs frameworks because they are simple, easy to understand, and have linear complexity [21]. GL-GNNs condense the most fruitful achievements of graph-level learning. In addition, some practitioners integrate the advantages of MPNN-based GL-GNNs with other techniques, particularly traditional learning techniques, to improve graph-level learning.

Graph Pooling. GL-DNNs and GL-GNNs always encode graph information into node representations that cannot be directly applied to graph-level tasks, graph pooling fills this gap. Graph pooling is a kind of graph downsizing technology where compact representations of a graph are produced by compressing a series of nodes into a super node [17, 19]. It is worthy to be recorded as a significant graph-level technique, as it is unique for graph-level learning without appearing in node-level and edge-level tasks. In addition, graph pooling has great power to preserve more information (e.g., hierarchical structure) for graph-level tasks, resulting in an abundant literature of related methods.

4 TRADITIONAL LEARNING

Traditional graph-level learning algorithms work in a deterministic way, encoding graphs using handcrafted features. Traditional graph-level learning methods can be divided into three main types: i.e., those based on graph kernels (GKs, Section 4.1), subgraph mining (Section 4.2), and graph embedding (Section 4.3). We summarize all discussed traditional graph-level learning models in Table 2 in Appendix B.

²The prediction results of a graph-level learning algorithm are invariant to any permutations of the order of nodes and/or edges of each input graph.

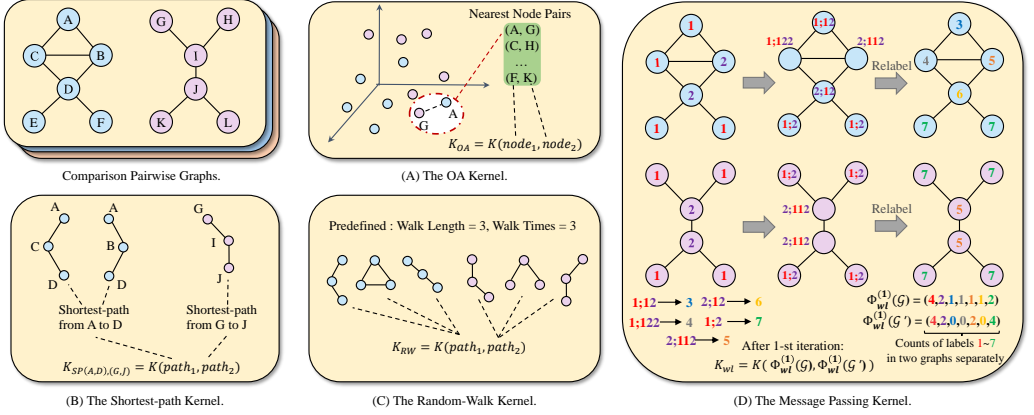


Fig. 3. Different mechanisms of four graph kernels in decomposing and comparing pairwise graphs.

4.1 Graph Kernels (GKs)

GKs perform graph-level learning based on kernel values (i.e., pair-wise graph similarities) []. Given a graph dataset \mathbb{G} , GKs decompose each graph \mathcal{G} into a bag-of-graphs $S^{\mathcal{G}} = \{g_1, \dots, g_I\}$, where $g_i \subseteq \mathcal{G}$ and g_i can be a node or a subgraph. Most GKs are based on the paradigm of an R -Convolution kernel [22] that obtains the kernel value $K_{R-conv}(\mathcal{G}, \mathcal{G}')$ of two graphs \mathcal{G} and \mathcal{G}' by:

$$K_{R-conv}(\mathcal{G}, \mathcal{G}') = \sum_{i=1}^I \sum_{j=1}^J K_{parts}(g_i, g'_j), \quad (1)$$

where $K_{parts}(g_i, g'_j)$ is the kernel function that defines how to measure the similarity between g_i and g'_j . A kernel matrix that packages all kernel values is then fed into an off-the-shelf machine learning model, such as a support vector machine (SVM), to classify the graphs.

4.1.1 Message Passing Kernels (MPKs). MPKs perform message passing on neighborhood structures to obtain graph representations. The 1-dimensional Weisfeiler-Lehman (1-WL) algorithm³ [4, 23] is one of the most representative MPKs. 1-WL updates a node's label (or color) iteratively. An illustration of 1-th iteration is shown in Fig. 3 (D). At the h -th iteration, 1-WL aggregates node v 's label $l^{(h-1)}(v)$ and its neighbor's labels $l^{(h-1)}(u), u \in \mathcal{N}(v)$ to form a multi-set⁴ of labels $\{l^{(h-1)}(v), \text{sort}(l^{(h-1)}(u) : u \in \mathcal{N}(v))\}$. Subsequently, 1-WL employs an injective hash function $\phi(\cdot)$ to map the $\{l^{(h-1)}(v), \text{sort}(l^{(h-1)}(u) : u \in \mathcal{N}(v))\}$ into a new label $l^{(h)}(v)$. Formally:

$$l^{(h)}(v) = \phi\left(l^{(h-1)}(v), \text{sort}(l^{(h-1)}(u) : u \in \mathcal{N}(v))\right). \quad (2)$$

When $\phi(\cdot)$ no longer changes the labels of any nodes, 1-WL stops iterating and generates a vector $\phi_{wl}(\mathcal{G})$ that describes \mathcal{G} . That is,

$$\phi_{wl}(\mathcal{G}) = [c^{(0)}(l_1^{(0)}), \dots, c^{(0)}(l_{I_0}^{(0)}); \dots; c^{(H)}(l_1^{(H)}), \dots, c^{(H)}(l_{I_H}^{(H)})], \quad (3)$$

where $l_i^{(h)}$ is the i -th label generated at the h -th iteration, and $c^{(h)}(l_i^{(h)})$ counts the occurrences of nodes labeled with $l_i^{(h)}$ in the h -th iteration. The kernel value of 1-WL between \mathcal{G} and \mathcal{G}' is the

³1-WL is also a well-known algorithm for graph isomorphism test.

⁴In a multiset, multiple elements are allowed to be the same instance.

inner product of $\phi_{wl}(\mathcal{G})$ and $\phi_{wl}(\mathcal{G}')$:

$$K_{WL}(\mathcal{G}, \mathcal{G}') = \langle \phi_{wl}(\mathcal{G}), \phi_{wl}(\mathcal{G}') \rangle. \quad (4)$$

The followed upgrading of 1-WL mainly focuses on aggregation and relabeling steps. Hido and Kashima [24] replaced the hash function with a binary arithmetic giving rise to a faster $\phi(\cdot)$. Morris *et al.* [25] used the idea of k -WL to relabel node groups consisting of k nodes that could form a connected graph. Theoretically, k -WL is more powerful than 1-WL for distinguishing between graph structures. Further, Neumann *et al.* [26] proposed a random label aggregation process based on node label distributions that only considers labels of part of neighbors. Random label aggregation saves time and computational resources making work on large-scale graphs more efficient. Persistent Weisfeiler–Lehman (P-WL) [27] is the recent enhancement to MPKs that adds weighted edges into the aggregation process. To calculate the edge weight, P-WL measures the distance between the continuous iterative updated labels of two end nodes. Additionally, P-WL can track changes in substructures that cannot be identified by 1-WL, such as cycles.

4.1.2 Shortest-path Kernels (SPKs). SPKs denote the kernel value as a comparison between pairwise node sequences (see Fig. 3 B). For example, the shortest-path kernel [28] determines the shortest path between the vertices v and u via the Floyd-Warshall [29] or Dijkstra's [30] algorithms. The distance between the pairwise shortest paths from \mathcal{G} and \mathcal{G}' is defined as the kernel value between them. Formally,

$$K_{SP}(\mathcal{G}, \mathcal{G}') = \sum_{\substack{v, u \in V \\ v \neq u}} \sum_{\substack{v', u' \in V' \\ v' \neq u'}} K_{Parts}((v, u), (v', u')) := \begin{cases} K_D(P(v, u), P(v', u')) & \text{if } l(v) \equiv l(v') \wedge l(u) \equiv l(u') \\ 0 & \text{otherwise,} \end{cases} \quad (5)$$

where $l(v)$ is the label of node v , $P(v, u)$ is the length of shortest path between vertices v and u , and $K_D(\cdot, \cdot)$ is a kernel comparing the shortest path lengths. Nikolentzos [31] proposed a variant of SPKs that draws on more information in a shortest path, such as node and edge labels, to calculate the distance of any two paths.

4.1.3 Random Walk Kernels (RWKs). RWKs are another kernel method guided by node sequences. Gärtner *et al.* [7] was the first to propose a random walk kernel. This technique counts the same random walk sequences that pair-wise graphs both own. Performing random walks on $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ simultaneously is the same as conducting random walks on a direct product graph $\mathcal{G}_\times = (\mathcal{V}_\times, \mathcal{E}_\times)$, where

$$\mathcal{V}_\times = \{(v, v') : v \in \mathcal{V} \wedge v' \in \mathcal{V}' \wedge l(v) \equiv l(v')\}, \quad \mathcal{E}_\times = \{(v, v'), (u, u') \in \mathcal{V}_\times : \mathcal{E}_{v,u} \in \mathcal{E} \wedge \mathcal{E}'_{v',u'} \in \mathcal{E}'\}. \quad (6)$$

Given \mathcal{G}_\times , the kernel function is defined as:

$$K_{RW}(\mathcal{G}, \mathcal{G}') = \sum_{i=1}^{|\mathcal{V}_\times|} \sum_{j=1}^{|\mathcal{V}_\times|} \left[\sum_{p=0}^P \lambda_p \mathbf{A}_\times^p \right]_{ij}, \quad (7)$$

where \mathbf{A}_\times is the adjacency matrix of \mathcal{G}_\times , P is the predefined max length of random walking sequences, and λ_p are the weights given to different P . $K_{RW}(\mathcal{G}, \mathcal{G}')$ counts the occurrences of common walk paths in \mathcal{G} and \mathcal{G}' with lengths equal to or less than P .

The random walk kernel in Eq. (7) assumes a uniform distribution for the beginning and ending probabilities of the walks across two graphs. However, Vishwanathan *et al.* [32] proposed a generalized version of RWKs. Specifically, they defined \mathbf{p} and \mathbf{q} as the beginning and ending probability

vectors in \mathcal{G} , respectively. In addition, they used the Kronecker product operation \otimes to derive \mathbf{A}_\times , that is $\mathbf{A}_\times = \mathbf{A} \otimes \mathbf{A}'$. Formally, the kernel value is:

$$K_{RW}(\mathcal{G}, \mathcal{G}') = \sum_{l=0}^{\infty} \mu_l (\mathbf{q} \otimes \mathbf{q}')^\top (\mathbf{A}_\times)^l (\mathbf{p} \otimes \mathbf{p}'), \quad (8)$$

where μ_l is the convergence coefficient.

RWGs suffer from a problem called tottering, where a random walk sequence traverses v to u and immediately returns to v via the same edge. To address tottering, Mahé *et al.* [33] employed a second-order Markov random walk that considers the last two steps in the current random walk sequence when deciding the next step.

4.1.4 Optimal Assignment Kernels (OAKs). Fröhlich *et al.* [34] was the first to propose OAKs. OAKs consider nodes as a basic unit for measuring kernel values. Of all the GKs introduced in this paper, OAKs are the only family of GKs that do not belong to R -Convolution paradigm. Specifically, given a fixed i in Eq. (1), OAKs only add in the maximum similarity value between g_i and g'_j where $j \in \{1, \dots, J\}$. Formally, OAKs are defined as:

$$K_{OA}(\mathcal{G}, \mathcal{G}') = \begin{cases} \max_{\pi \in \Pi_I} \sum_{i=1}^I K_{parts}(g_i, g'_{\pi[i]}), & \text{if } J \geq I \\ \max_{\pi \in \Pi_I} \sum_{j=1}^J K_{parts}(g_{\pi[j]}, g'_j), & \text{otherwise} \end{cases} \quad (9)$$

where Π_I represents all permutations of the indexes of a bag-of-graphs $\{1, \dots, I\}$, and π is the optimal node permutation to reach maximum similarity value between two graphs.

Searching for a pair-wise element with the maximum similarity tends to be a highly time-consuming process. Hence, to reduce the time requirement of this task, Johansson *et al.* [35] mapped the graphs in geometric space and then calculated the Euclidean distance between pair-wise nodes. This method enables OAKs to use approximate nearest neighbors algorithms in Euclidean space as a way to speed up the process. Transitive Assignment Kernels (TAKs) [36, 37] are variants of OAKs. Unlike OAKs that search for the optimal assignment among pair-wise graphs, TAKs identify node permutations that with the most similar node pairs among three or more graphs. OAKs have been confined to node similarity measurement, although they can be extended to measure subgraph similarities so as to capture a graph's topological information [38]. As discussed next, we introduce the GKs with subgraph information.

4.1.5 Subgraph Kernels (SGKs). SGKs calculate the similarity between two graphs by comparing their subgraphs. For example, the representative SGK – Graphlet Kernel [39] uses either depth-first search (DFS) or sampling to identify the subgraphs. With these subgraphs, the vector $\phi_{SG}(\mathcal{G}) = [c_{\mathcal{T}_1}^{(\mathcal{G})}, \dots, c_{\mathcal{T}_N}^{(\mathcal{G})}]$ is then used to describe the graph \mathcal{G} , where \mathcal{T}_i means the i -th isomorphism type of subgraphs, N is the total number of subgraphs' types, and $c_{\mathcal{T}_i}^{(\mathcal{G})}$ counts the occurrences of the \mathcal{T}_i category subgraphs in graph \mathcal{G} . Graphlet's kernel value is then defined as the inner product of $\phi_{SG}(\mathcal{G})$ and $\phi_{SG}(\mathcal{G}')$:

$$K_{SG}(\mathcal{G}, \mathcal{G}') = \langle \phi_{SG}(\mathcal{G}), \phi_{SG}(\mathcal{G}') \rangle. \quad (10)$$

There are several different implementations of SGKs kernel functions. For instance, Wale *et al.* [40] employed a min-max kernel $\frac{\sum_{i=1}^N \min(c_{\mathcal{T}_i}^{(\mathcal{G})}, c_{\mathcal{T}_i}^{(\mathcal{G}')})}{\sum_{i=1}^N \max(c_{\mathcal{T}_i}^{(\mathcal{G})}, c_{\mathcal{T}_i}^{(\mathcal{G}')})}$ to measure the distance between two graphs. Subgraph Matching Kernels (SMKs) [41] calculate the similarity between two subgraphs by counting the number of nodes with the same labels. Then the similarities between all pairwise subgraphs sourced from the two graphs are summed as the kernel value of the SMKs. Methods of identifying the

subgraphs in SGKs have also been explored. For example, Neighborhood Subgraph Pairwise Distance Kernels (NSPDK) [42] denotes the subgraphs as the first-, second-, and third-hop neighborhoods of pairwise vertices with the shortest path of a predefined length. However, the main contributions of SGKs lie in assessing the similarity of graphs in terms of a set of selected subgraphs, not how the subgraphs are chosen. More detailed and sophisticated subgraph mining methods are demonstrated next.

4.2 Subgraph Mining

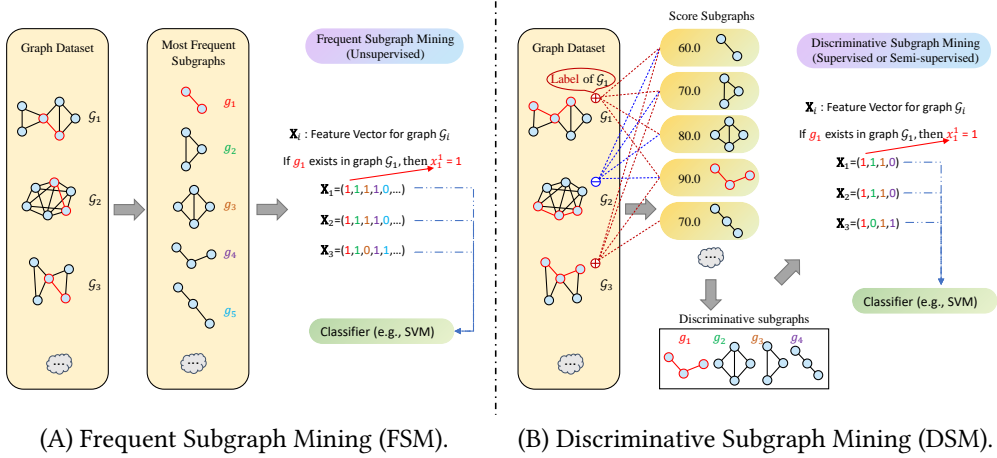


Fig. 4. Different subgraph extraction methods of FSM and DSM.

Subgraph mining is similar to SGKs, where the vector $\mathbf{x}_i = [x_i^1, \dots, x_i^M]^\top$ is taken as a graph-level representation of the graph G_i . Here, $x_i^m \in \{0, 1\}$, $x_i^m = 1$ if $g_m \subseteq G_i$, otherwise, $x_i^m = 0$. The established graph-level representation is then directly input into an off-the-shelf machine learning model, such as SVM classifier, for downstream tasks. What is different about subgraph mining algorithms is that they place particular emphasis on how to extract the optimal subgraph set $\mathcal{S}^* = \{g_1, \dots, g_T\}$ from the subgraph set $\{g_1, \dots, g_M\}$, where g_1, \dots, g_M denote all possible subgraphs of $\mathbb{G} = \{G_1, \dots, G_N\}$. Techniques for extracting subgraphs can be divided into two branches depending on how the supervision information is used. Frequent subgraph mining is the unsupervised method, as illustrated in Fig. 4 A, while discriminative subgraph mining is the supervised or semi-supervised method (see Fig. 4 B).

4.2.1 Frequent Subgraph Mining (FSM). FSM identifies the subgraphs whose frequency of occurrence in \mathbb{G} sits over a predefined threshold δ . These subgraphs are then added to \mathcal{S}^* . Apriori-like algorithms, such as AGM [8] and FSG [43], enumerate subgraphs from size one to a predefined largest size as candidates for \mathcal{S}^* . In the enumeration, these apriori-like algorithms pick up the candidates that occur more frequently than δ and add them to \mathcal{S}^* . Others subgraphs are dropped and expansions based on those subgraphs are no longer considered. Testing for subgraph isomorphism with vast numbers of candidate subgraphs can mean apriori-like algorithms suffer from computation bottlenecks. To address this issue, gSpan [44] employs a depth-first-search (DFS) strategy to search subgraphs, while assigning a unique DFS code of minimum length for each subgraph searched. gSpan can then do a quick check for isomorphism by simply comparing the DFS codes of pairwise subgraphs.

4.2.2 Discriminative Subgraph Mining (DSM). DSM extracts discriminative subgraphs from a set of all possible subgraphs of \mathbb{G} based on the label information. Given a binary graph classification task, Thoma *et al.* [45] defined an evaluation criterion called CORK which describes the discriminative score of a subgraph set \mathcal{S} , $\mathcal{S} \subseteq \{g_1, \dots, g_M\}$. Formally,

$$\text{CORK}(\mathcal{S}) = -1 \times \text{num}(\mathcal{G}_i, \mathcal{G}_j), \quad \text{s.t. } \mathcal{G}_i \subset \mathbb{G}_+ \wedge \mathcal{G}_j \subset \mathbb{G}_- \wedge \forall g_m \in \mathcal{S} : x_i^m = x_j^m, \quad (11)$$

where $\text{num}(\cdot)$ counts the number of pairs of graphs $(\mathcal{G}_i, \mathcal{G}_j)$ satisfying the specific conditions. \mathbb{G}_+ is the set of graphs with positive labels, while \mathbb{G}_- is the set of graphs with negative labels. The optimal subgraph set \mathcal{S}^* has the highest CORK score among all possible subgraph sets \mathcal{S} containing T subgraphs, denoted as:

$$\mathcal{S}^* = \underset{\mathcal{S} \subseteq \{g_1, \dots, g_M\}}{\text{argmax}} \text{CORK}(\mathcal{S}) \quad \text{s.t. } |\mathcal{S}| \leq T. \quad (12)$$

The CORK score can also be used to prune the subgraph search space of gSpan, mentioned in Section 4.2.1. More specifically, if replacing any existing element of \mathcal{S}^* with a subgraph g_m does not \mathcal{S}^* 's CORK score, gSpan will no longer perform DFS along g_m . To speed up DSM based on discriminative scores and gSpan, Yan *et al.* [46] proposed LEAP, which initializes an optimal subgraph set \mathcal{S}^* with frequent subgraphs. In this way, LEAP prunes gSpan's search space right at the beginning. In addition, Kong *et al.* [47] and Wu *et al.* [48] expanded DSM to the multi-label⁵ and multi-view⁶ scenarios, respectively. Note, however, that all the DSM methods discussed are supervised methods. In terms of semi-supervised subgraph mining, Kong and Yu [49] proposed gSSC which maps each graph into a new feature space by \mathcal{S}^* . Unlabeled graphs are separated from each other in the new feature space. In the labeled group, graphs with the same label are close, whereas graphs with different labels remain distant. In addition, Zhao *et al.* [50] only used the positively labeled graphs and unlabeled graphs to select \mathcal{S}^* when performing binary graph classification tasks. This is because sometimes the real-world data is composed of an incomplete set of positive instances and unlabeled graphs.

4.3 Non-learnable Graph Embedding

Graph embeddings are the compression of graphs into a set of lower-dimensional vectors. Some non-learnable graph embedding methods extract graph-level representations from the inherent properties of graphs, e.g., their topologies and eigenspectrums.

Local Degree Profile (LDP) [51] summarizes the degree information of each node and its 1-hop neighbors as node features. LDP constructs graph representations by building an empirical distribution or histogram of the hand-crafted node features. In addition to node degree, non-learnable graph embedding can also leverage anonymous random walk sequences to describe a graph's topological information. Specifically, anonymous random walks record the status change of node labels. Two anonymous random walk sequences $A \rightarrow B \rightarrow A$ and $B \rightarrow A \rightarrow B$ can be both written as $1 \rightarrow 2 \rightarrow 1$. Anonymous Walk Embeddings (AWE) [52] encodes a graph via an n -dimensional vector in which each element represents the occurrence frequency of a specific anonymous random walk sequence.

In spectral graph theory [53], the spectrum of a graph is determined by its topology. Based on this theory, the Family of Graph Spectral Distances (FGSD) method [54] proposes that the distance between the spectrums of two graphs can be used to test whether the graphs are isomorphic.

⁵Each graph owns more than one label, such as a drug molecular can own different labels to represent anti-cancer effects for various cancers, e.g., breast cancer (+) and lung cancer (-).

⁶An object has different views, where each view can represent a separate graph, e.g., a scientific publication network is shown as two graphs, an abstract graph demonstrating the keywords correlations in the abstract of papers, and a reference citation graph about citation relationships.

Thus, the histogram of the spectrum is used to construct a graph-level representation. Analogously, A-DOGE [55] depicts a graph by computing the spectral density across its eigenspectrum. However, these methods are limited to use with small graphs given the prohibitive costs of computing eigenspectrum decompositions with large-scale graphs. As a possible solution to this limitation, SlaQ [56] uses stochastic approximations as a way of quickly calculating the distance between two graphs' spectral densities. More specifically, these authors employed von Neumann graph entropy (VNGE) [57, 58] as a way of approximately representing the spectral properties of the graphs. In turn, this approximation supports fast computation by tracing a Laplacian matrix of the graph. Liu *et al.* [59] proposed another fast approximation method involving VNGE, which is based on deriving the error bound of the approximation estimation.

5 GRAPH-LEVEL DEEP NEURAL NETWORKS (GL-DNNs)

GL-DNNs form the basis of a pioneering set of works that employ deep learning techniques to achieve graph-level learning. Researchers have explored graph-level learning techniques based on classic deep neural networks including skip-gram neural network, recurrent neural networks (RNNs), convolution neural networks (CNNs), and capsule neural networks (CapsNets) to achieve Skip-gram-based (see Section 5.1), RNN-based (see Section 5.2), CNN-based (see Section 5.3), and CapsNets-based (see Section 5.4) GL-DNNs, respectively. The representative GL-DNNs mentioned in this section are summarized in Table 3 in Appendix C.

5.1 Skip-gram-Based GL-DNNs

Skip-gram [60] is a widely used unsupervised neural networks, to predict the context words for the target word. Initially, the researchers built a skip-gram model based on the relationship between two adjacent subgraphs, namely subgraph2vec [61]. Subgraph2vec first takes the $(d-1)$ -, d -, $(d+1)$ -hop neighborhoods of the v th selected node in the graph \mathcal{G}_i as three subgraphs g_{v-1}^i , g_v^i , g_{v+1}^i , respectively, where $d \geq 1$ is a predefined value. $\{\mathbf{w}_{1-1}^1, \dots, \mathbf{w}_{V+1}^1; \dots; \mathbf{w}_{1-1}^N, \dots, \mathbf{w}_{V+1}^N\}$ are the randomly initialized embeddings of all sampled subgraphs $\{g_{1-1}^1, \dots, g_{V+1}^1; \dots; g_{1-1}^N, \dots, g_{V+1}^N\}$ respectively, where N represents the total number of graphs, and V is the number of selected nodes in each graph. Then, the Skip-gram model is used to update the subgraph embeddings. The Skip-gram model takes \mathbf{w}_v^i as its input, and predicts the context of \mathbf{w}_v^i (i.e., \mathbf{w}_{v-1}^i and \mathbf{w}_{v+1}^i). Then the prediction results are back-propagated to update \mathbf{w}_v^i . To summarize, subgraph2vec's learning objective is to maximize the following log-likelihood:

$$\sum_{i=1}^N \sum_{v=1}^V \log \Pr(\mathbf{w}_{v-1}^i, \dots, \mathbf{w}_{v+1}^i | \mathbf{w}_v^i). \quad (13)$$

Another method, Graph2vec [62] was designed to tackle graph representation tasks. By establishing a semantic association between a graph and its sampled subgraphs, Graph2Vec employs the idea of Skip-gram to learn a graph embedding. Following this work, Dang *et al.* [63] replaced the sampled subgraphs in Graph2vec with frequent subgraphs that have more discriminative features for graph classification tasks.

5.2 RNN-Based GL-DNNs

RNNs are particularly good at learning sequential data, such as text and speech. There are two main types of algorithms that apply RNNs to graph-level learning. One type transforms graphs into sequential-structured data. The other aggregates neighborhood information about the target node and relabels those aggregated features through an RNN. This is similar to Message Passing Kernels (MPKs, Section 4.1.1).

A natural way to capture the sequential information in graphs is to use a series of random walk paths to represent a graph. For example, GAM [64] employs a long short-term memory (LSTM) model to guide a random walk on graphs. Meanwhile, the LSTM model generates a representation for the walk sequence to describe the graph. In addition, Zhao *et al.* [65] proposed an RNN-based graph classification algorithm called SAN. Starting from a node, SAN employs an RNN model that adds nodes and edges to form an informative substructure whose representation is progressively generated by the RNN model. A graph-level representation that can be used for graph classification tasks is then generated by summing all the representations of the formed substructures. Given a graph generation task, NetGAN [66] uses an LSTM model as a generator to yield fake walk sequences, while a discriminator disambiguates the graph's real walk sequences from the generated fake ones to reverse-train the generator. Another graph generation model Graphrnn [67] creates various permutations of graphs, with various combinations of nodes and edges as sequential data to be input into an RNN model.

The second category of RNN-based GL-DNNs implements neural networks version of MPKs through RNN models. As such, the algorithms in this category can be viewed as the predecessors of MPNNs. Scarselli *et al.* [68] recurrently updated node embeddings until reaching a stable situation, that is:

$$\mathbf{h}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} f_w(\mathbf{x}_v, \mathbf{s}_{u,v}, \mathbf{x}_u, \mathbf{h}_u^{(k-1)}), \quad (14)$$

where \mathbf{h}_u^0 is randomly initialized and $f_w(\cdot)$ is a parametric function that maps vectors into a concentrated space to shorten their distance. To address the graph-level task, a super node connected with all the other nodes is used to output the representation of the whole graph. In addition, Li *et al.* [69] proposed the idea of using a gated recurrent unit (GRU) to relabel the aggregated information from the 1-hop neighborhoods of the center node. This approach reduces the recurrent process for updating node embeddings to a fixed number of steps and avoids control convergence parameters, formulated as:

$$\mathbf{h}_v^{(k)} = \text{GRU}\left(\mathbf{h}_v^{(k-1)}, \text{AGG}^{(k)}\left(\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)\right)\right), \quad (15)$$

where $\mathbf{h}_v^{(k)}$ represents the node representation of v at the k -th iteration, $\mathbf{h}_v^{(0)}$ is the node feature \mathbf{x}_v , and here AGG is a weighted sum aggregation function. This algorithm continues the recurrent process until it hits the predefined K number of iterations needed to form the node representations. A graph-level representation is then produced via:

$$\mathbf{h}_G = \tanh\left(\sum_{v \in \mathcal{V}} f_t\left(\mathbf{h}_v^{(K)}, \mathbf{h}_v^{(0)}\right) \odot \tanh\left(\mathbf{h}_v^{(K)}\right)\right), \quad (16)$$

where $f_t(\cdot)$ is a softmax function guided by an attention mechanism, that preserves and aggregates valuable node representations for specific graph-level tasks. $\tanh(\cdot)$ is an activation function, and \odot is element-wise multiplication.

5.3 CNN-Based GL-DNNs

Another significant deep learning technique that works in the Euclidean domain is CNN. Here, grid-structured data, such as images, are studied. Similar to RNN-based GL-DNNs, there are two main branches of CNN-based graph-level learning. In Appendix C.1, Fig. 9 depicts the details of these two different branches.

The first branch sorts nodes and arranges the node features to form a concentration matrix, of grid-structured data, to train the CNNs. PATCHY-SAN [70] selects a fixed number of neighbors of a central node and sorts neighbors to concatenate their features as the grid-structured feature matrix.

By choosing a series of central nodes, PATCHY-SAN constructs some matched feature matrices. Finally, a graph-level representation is produced by the CNN model from the concatenation matrix of all built feature matrices. In addition, Kernel Convolutional Neural Network (KCNN) [71] sorts all vertices in a graph to form grid-structured data. A neighborhood graph is built for each vertex and a kernel matrix is constructed by implementing the kernel function (i.e., an SPK or an MPK) between all pairwise neighborhood graphs. In this work, the grid-structured data for feeding up CNN is the kernel matrix, where each row is a vector describing the similarities between the neighborhood graph of the matched index vertex and the other neighborhood graphs.

The second branch involves CNN-guided neural network versions of MPKs. These methods have two main steps: aggregating neighborhood information to the central node, and using the convolution operation to relabel the aggregated features. NN4G [72] performs a convolution kernel upon 1-hop neighbors for updating the center node and outputs the graph-level representations based on the node embeddings produced by each convolution layer, which is defined as:

$$\mathbf{h}_v^{(k)} = f\left(\mathbf{w}^{(k-1)\top} \mathbf{x}_v + \sum_{i=1}^{k-1} \mathbf{w}_{k,i}^\top \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i)}\right), \quad \mathbf{h}_{\mathcal{G}} = f\left(\sum_{k=1}^K \mathbf{w}_k \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_v^{(k)}\right), \quad (17)$$

where $f(\cdot)$ is a linear or sigmoidal function and $h_v^{(0)} = 0$. Another related work, ECC [73] concatenates 1-hop neighbor embeddings $(\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v))$ around the central node v to construct a feature matrix by the k -th iteration. Subsequently, a convolution and average operation is executed on the aggregated neighbor feature matrix to obtain a representation for the central node. Then a graph-level representation is produced via max-pooling the node embeddings.

$$\mathbf{H} = [\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)], \quad \mathbf{h}_v^{(k)} = \frac{1}{|\mathcal{N}(v)|} (\mathbf{W} \odot \mathbf{H}) + b^{(k)}, \quad \mathbf{h}_{\mathcal{G}} = \text{MaxPooling}(\mathbf{h}_v^{(K)} : v \in \mathcal{V}). \quad (18)$$

Moreover, Diffusion CNN (DCNN) [74] aggregates multi-hop neighborhood features to the central nodes through a matrix multiplication $\mathbf{P}\mathbf{X}$, where $\mathbf{P} = [\mathbf{A}, \mathbf{A}^2, \dots, \mathbf{A}^h] \in \mathbb{R}^{h \times n \times n}$ is a three-dimensional tensor containing multi-hop (i.e., 1-, 2-, ..., h -hops) adjacent matrices and $\mathbf{X} \in \mathbb{R}^{n \times f}$ is the node features matrix. $\mathbf{P}\mathbf{X} \in \mathbb{R}^{h \times n \times f}$ represents the updated node features after multi-hop aggregation. For graph classification tasks, DCNN permutes the dimensions giving $\mathbf{P}\mathbf{X} \in \mathbb{R}^{n \times h \times f}$ and all node representations are averaged as $\mathbf{P}^* \in \mathbb{R}^{h \times f}$. Subsequently, a convolution operation is implemented on \mathbf{P}^* to produce a graph-level representation. The convolution operation can be defined as $\mathbf{h}_{\mathcal{G}} = f(\mathbf{W} \odot \mathbf{P}^*)$, where $f(\cdot)$ is a nonlinear activation function, and \mathbf{W} is a trainable weight matrix for convolution and summation.

5.4 CapsNet-Based GL-DNNs

CapsNets [75] were originally designed to capture more spatial relationships between the partitions of an entity than CNNs. CapsNets are available to assemble vectorized representations of different features (e.g., colors, textures) to a capsule dealt with by a specific network. Thus, applying a CapsNet to a graph preserves rich features and/or structure information at the graph level.

Graph Capsule Convolutional Neural Networks (GCAPS-CNN) [76] iteratively aggregates neighbor information under different statistical moments (e.g., mean, standard deviation) to form a

capsule representation of the central node, formulated as:

$$\mathbf{h}_v^{(k)} = \frac{1}{|\mathcal{N}(v)|} \begin{bmatrix} \left(\sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(k-1)} \right) \mathbf{W}_1 \text{ (mean)} \\ \left(\sum_{u \in \mathcal{N}(v)} \left(\mathbf{h}_u^{(k-1)} - \mu \right)^2 \right) \mathbf{W}_2 \text{ (std)} \\ \left(\sum_{u \in \mathcal{N}(v)} \left(\frac{\mathbf{h}_u^{(k-1)} - \mu}{\sigma} \right)^3 \right) \mathbf{W}_3 \text{ (skewness)} \\ \vdots \end{bmatrix}, \quad (19)$$

where $(\mathbf{W}_1, \mathbf{W}_2, \dots)$ are the learnable weight matrices for mapping the aggregated features into a uniform hidden space with a dimensionality of h . If the number of statistical moments is p and the final iteration number is K , each node will be represented as $\mathbf{h}_v^{(K)} \in \mathbb{R}^{p \times h}$, and the matrix of all n node embeddings will be $H^{(K)} \in \mathbb{R}^{n \times p \times h}$. This approach employs a covariance function as the permutation-invariant layer to output a graph-level representation, defined as:

$$\mathbf{h}_G = \frac{1}{n} (H^{(K)} - \mu)^\top (H^{(K)} - \mu). \quad (20)$$

CapsGNN [77] iteratively aggregates node features to a center node, and, in turn, adds the aggregation results of each iteration to a capsule representation of the central node. An attention mechanism is then applied to all node capsules so as to generate a graph capsule that can be plugged into a capsule network for graph classification. Mallea *et al.* [78] employs the same approach as PATCHY-SAN [70] to find substructures in graphs, while the feature matrices of searched substructures are assembled in a capsule network for graph classification.

6 GRAPH-LEVEL GRAPH NEURAL NETWORKS (GL-GNNS)

This section focuses on GL-GNNs, which are the most influential graph-level learning techniques at present. The cornerstone branch of GL-GNNs —Message Passing Neural Networks (MPNNs) (see Section 6.1) —are introduced first, followed by. Some emerging methods in GL-GNNs, such as subgraph-based methods (see Section 6.2) and graph kernel-based methods (see Section 6.3). Notably, these emerging approaches take advantage of some of the insights from traditional graph-level learning methods. In addition, we review progress in spectral GL-GNNs (see Section 6.4), which push graph-level learning forward through spectrum properties. There are also some contents related to GL-GNNs in Appendix D, such as contrastive learning-based approaches (see Appendix D.1), the expressivity (see Appendix D.2), generalizability (see Appendix D.3), and explainability (see Appendix D.4) of GL-GNNs. Please refer to Table 4 in Appendix D for the GL-GNNs discussed in this section.

6.1 Message Passing Neural Networks (MPNNs)

As mentioned, researchers have developed RNN- and CNN- based versions of MPKs. However, as the influence of deep learning has expanded, researchers have also developed various feedforward versions of Message Passing Kernels (MPKs, refer to Section 4.1.1). Collectively, these are called MPNNs. MPNNs are similar to RNN-based MPKs in Eq. (15), but MPNNs set different weights in separate layers rather than sharing weights in all layers. Gilmer *et al.* [21] summarizes a collection of MPNNs [79–81] and further proposes a unified framework for this branch of techniques, as

shown in Fig. 5 (A) and denoted as:

$$\mathbf{h}_v^{(k)} = U^{(k)} \left(\mathbf{h}_v^{(k-1)}, \sum_{u \in \mathcal{N}(v)} M^{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathcal{E}_{v,u} \right) \right), \quad (21)$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $M^{(k)}$ is a function that outputs the passed message for the target node based on itself and its neighbors, and $U^{(k)}(\cdot)$ updates the embedding of the target node. After multiple iterations, the node embeddings $\mathbf{h}_v^{(k)}$ learn the local structure information and the graph-level topology has distributed in all nodes. A readout function reads all node embeddings and outputs a graph-level representation, that is:

$$\mathbf{h}_{\mathcal{G}} = \text{readout} \left(\mathbf{h}_v^{(k)} : v \in \mathcal{V} \right). \quad (22)$$

MPNNs have become the mainstream of graph-level studies [21]. They are also representative of spatial-based GL-GNNs since they are easy to use through matrix operations. Lastly, the time and memory complexity of MPNNs only grows linearly with the graph size, making this a very practical approach for large sparse graphs. In recent years, practitioners have developed numerous enhanced versions of MPNNs, including subgraph-enhanced MPNNs (see Section 6.2), and kernel-enhanced MPNNs (see Section 6.3).

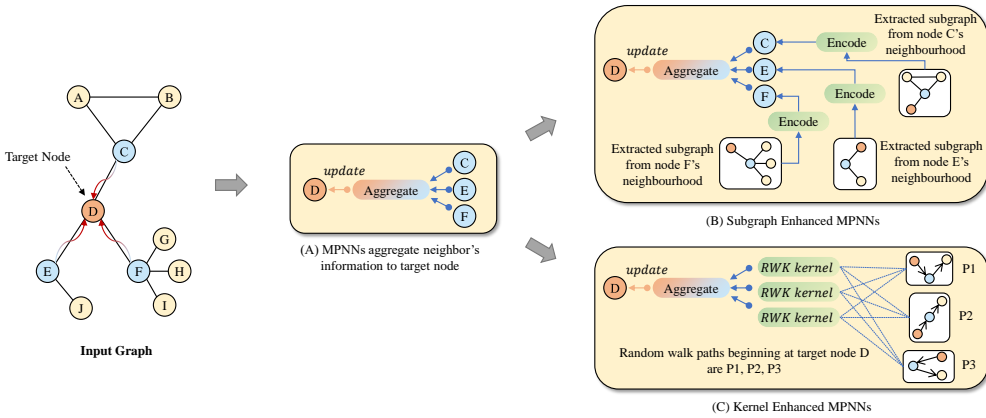


Fig. 5. Different mechanisms of MPNNs, Subgraph Enhanced MPNNs, and Kernel Enhanced MPNNs. In Subgraph Enhanced MPNN, we used 1-hop neighborhoods as the subgraph for easy understanding, but the specific subgraph extraction is up to the article.

6.2 Subgraph-Based GL-GNNs

In recent years, investigating GL-GNNs that are capable of capturing more topological information has been a crucial stream of study. This is especially, since a number of works have uncovered structure-aware flaws in MPNNs. To this end, practitioners have devised subgraph-based GL-GNNs, which leverage the rich structural information in subgraphs. These subgraph-based GL-GNNs can be divided into two branches. The first branch enhances an MPNN by injecting the subgraph information into the aggregation process, as outlined in Fig. 5 (B). The other branch borrows the graphlet idea and decomposes the graph into a few subgraphs, merging multiple subgraph embeddings to produce an embedding of the entire graph.

6.2.1 Subgraph Enhanced MPNNs. As mentioned, MPNNs learn topological information via a neighborhood aggregation process. However, standard MPNNs only aggregate node features, not structural information. Therefore, a straightforward way of strengthening an MPNNs is to enrich the features of the nodes or edges with subgraph information. Graph Substructure Network (GSN) [82], for example, counts the number of occurrences of a predefined subgraph pattern g_1, \dots, g_M (e.g., a cycle or a triangle) that involves the target node v or edge $\mathcal{E}_{v,u}$. From these, subgraph feature vectors are constructed for v as \mathbf{x}_v^g or for $\mathcal{E}_{v,u}$ as $\mathbf{S}_{v,u}^g$, denoted as:

$$\begin{cases} \mathbf{x}_v^{g_m} = |\{g_s \simeq g_m : v \in \mathcal{V}, v \in \mathcal{V}_{g_s}, g_s \subseteq \mathcal{G}\}|, & \mathbf{x}_v^g = [x_v^{g_1}, \dots, x_v^{g_M}]^\top \text{ (Node)}, \\ \mathbf{S}_{v,u}^{g_m} = |\{g_s \simeq g_m : \mathcal{E}_{v,u} \in \mathcal{E}, \mathcal{E}_{v,u} \in \mathcal{E}_{g_s}, g_s \subseteq \mathcal{G}\}|, & \mathbf{S}_{v,u}^g = [S_{v,u}^{g_1}, \dots, S_{v,u}^{g_M}]^\top \text{ (Edge)}, \end{cases} \quad (23)$$

where g_m is a predefined subgraph pattern, and $g_s \simeq g_m$ means g_s is isomorphic to g_m , $x_v^{g_m}$ counts the number of isomorphic subgraphs g_s containing the node v , and $S_{v,u}^{g_m}$ indicates the number of isomorphic subgraphs g_s containing the edge $\mathcal{E}_{v,u}$. As a last step, the subgraph feature vectors for the node \mathbf{x}_v^g and the edge $\mathbf{S}_{v,u}^g$ are injected into the aggregation layer, which is defined as:

$$\mathbf{h}_v^{(k)} = U^{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)} \right), \quad \mathbf{m}_v^{(k)} = \begin{cases} \sum_{u \in \mathcal{N}(v)} M^{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}_v^g, \mathbf{x}_u^g, \mathcal{E}_{v,u} \right) \text{ (Node)}, \\ \sum_{u \in \mathcal{N}(v)} M^{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{S}_{v,u}^g, \mathcal{E}_{v,u} \right) \text{ (Edge)}. \end{cases} \quad (24)$$

GSN is a promising start for subgraph-enhanced MPNNs. However, they have one fatal drawback in that searching for and testing subgraphs for isomorphism is computationally prohibitive. To avoid this high computational bottleneck, GNN-AK [83] samples subgraphs and swift encodes them into node embeddings. Specifically, GNN-AK extracts the neighborhoods of each node as subgraphs (i.e., the neighborhood of node v is a subgraph g_v), and applies a base MPNN to each neighborhood subgraph to obtain the final node embeddings, i.e.:

$$\mathbf{x}_v^g = [\text{Emb}(v|g_v) \mid \sum_{u \in \mathcal{V} \wedge u \neq v} \text{Emb}(u|g_v) \mid \sum_{u \in \mathcal{V} \wedge u \neq v} \text{Emb}(v|g_u)], \quad (25)$$

where $\text{Emb}(v|g_v)$ is the embedding of node v produced by running the base MPNN on subgraph g_v , $\text{Emb}(v|g_u) = 0$ if subgraph g_u does not contain node v (i.e., $v \notin \mathcal{V}_{g_u}$), and \mathbf{x}_v^g is the subgraph feature of node v for MPNN's aggregation.

Analogously, Nested Graph Neural Networks (NGNN) [84] extracts nodes (i.e., $\mathcal{N}(v) \cup v$) and edges (i.e., $\mathcal{E}_{v_1, v_2} \in \mathcal{E}$ & $v_1, v_2 \in \mathcal{N}(v) \cup v$) in the 1-hop neighborhood of node v , as a neighborhood subgraph g_v , to be encoded by a GNN. The subgraph g_v is then encoded as the embedding h_{g_v} , which denotes the subgraph feature of node v .

One thing common to all the above methods is that they dilute or replace the node features. But such feature properties are essential for graph-level learning. Thus, GraphSNN [85] incorporates the idea of encoding the subgraph features into the edge's weight for aggregation without changing the node features. This approach defines the formula for calculating the degree of isomorphism between two subgraphs. The weight of $\mathcal{E}_{v,u}$ is equal to the degree of isomorphism between two specific subgraphs, where one of the subgraphs is the node v 's neighborhood subgraph, and the other subgraph is the overlap between the neighborhood subgraphs of nodes u and v . By normalizing the computed weights at the end, GraphSNN builds a subgraph-guided attention mechanism partaking in the MPNN's aggregation.

6.2.2 Graphlet. In addition to empowering MPNNs through subgraph information, researchers have directly used the embeddings of subgraphs to form a graph-level representation. SUGAR [86], for example, uses GNNs to embed discriminative subgraphs selected through reinforcement

learning. A readout function over all learned subgraph embeddings is then used to build a graph-level representation for classification, which can be used for classification. Correspondingly, the graph classification results are back-propagated to train the GNNs that embed selected subgraphs. Similarly, Subgraph Neural Networks (SubGNN) [87] views the subgraphs of a graph as instances with independent labels. For each instance, SubGNN samples a few finer local structures, and forms embeddings through the GNN. A representation of each instance is generated by aggregating all the embeddings of the sampled local structures. Another approach, Equal Subgraph Aggregation Network (ESAN) [88], enhances this branch by applying two GNNs, one for learning individual embeddings for sampled subgraphs and the other for learning message passing among them. Finally, a universal set encoder [89] compresses all the subgraph embeddings into one graph-level representation.

6.3 Graph Kernel-Based GL-GNNs

Like the revival of the subgraph idea in the deep learning field, graph kernels that incorporate deep learning techniques have also attracted attention. Similar to subgraph-based GL-GNNs, there are generally two branches of graph kernel-based GL-GNNs. As Fig. 5 (C) shows, one branch replaces the 1-hop neighbor aggregation and vertex update functions in MPNNs with a kernel function. This group is, called the kernel-enhanced MPNNs. In the other branch, differentiable and parameterizable kernels are designed to plug kernels into the neural networks so as to form learnable and fast deep graph kernels.

6.3.1 Kernel-enhanced MPNNs. This type of method often uses a graph kernel to update the node embeddings, which in turn are used to recalculate the graph kernel. Kernel-enhanced MPNNs break the local update of the MPNN (i.e., the node features are aggregated via adjacent neighbors) so as to capture more structure information.

For example, Graph Convolutional Kernel Networks (GCKN) [90] and Graph Structured Kernel Networks (GSKN) [91] employ walk-based kernels to iteratively generate node embeddings. Specifically, these methods generate q walking sequences starting from the target node, where each sequence records all node embeddings in the walk. As an example, in the k -th iteration, the one-step walk sequence P_i from node v to u would be represented as $R(P_i) = [\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}]^\top$. By building a kernel function $K(R(P_i), R(P_j))$ (e.g., a random walk kernel) as the similarity measurement for any two walking sequences P_i and P_j , GCKN and GSKN aggregate the kernel values as the updated node embeddings, that is:

$$h_v^{(k)} = \sum_{1 \leq i \leq q} [K(R(P_1), R(P_i)), \dots, K(R(P_q), R(P_i))]^\top. \quad (26)$$

To follow up, the node embeddings updated by the graph kernel are used to obtain the kernel value in the next iteration. Du *et al.* [20] combined a Neural tangent kernel (NTK) [92] with an MPNN, summarizing the advantages of this category of approach. Overall, the technique gives better theoretical explanations, brought about by the graph kernel, and the convex-optimized tasks are easy to train. Thus, kernel-enhanced MPNNs use a kernel function to replace the aggregation and vertex update functions in MPNNs. The walk-based kernels do particularly well at capturing local structures to encode into the node embeddings.

6.3.2 Deep Graph Kernel. Traditional graph kernels are limited by the theoretical computational bottleneck, thus, researchers search for an optimal solution for comparing two graphs by neural networks. Recently, Lei *et al.* [93] discussed deep graph kernels as parameterized learnable graph kernels for deriving neural operations. These deep graph kernels can be optimized for specific tasks with fast computation speeds and good interpretability.

Deep Divergence Graph Kernels (DDGK) [94] takes M base graphs $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_M\}$ to represent a target graph \mathcal{G}_t as M -dimensional vectors $\mathbf{h}_{\mathcal{G}_t} = [K_D(\mathcal{G}_1, \mathcal{G}_t), \dots, K_D(\mathcal{G}_M, \mathcal{G}_t)]^\top$, where $K_D(\mathcal{G}_m, \mathcal{G}_t)$ is a trainable kernel for measuring the distance between \mathcal{G}_m and \mathcal{G}_t . First, DDGK uses each base graph $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_M\}$ to train an encoder $\{\mathcal{Z}_1, \mathcal{Z}_2, \dots, \mathcal{Z}_M\}$. The encoder \mathcal{Z}_m takes the one-hot encoding of nodes (e.g., the first node's encoding is $[1, 0, 0, \dots]^\top$) in \mathcal{G}_m as the input and tries to predict their neighbors (e.g., if a node only links to the second and third nodes, the correct output should be $[0, 1, 1, 0, \dots]^\top$). Then, the trained encoder \mathcal{Z}_m is used for predicting the node's neighbors in \mathcal{G}_t , as the divergence score $K_D(\mathcal{G}_m, \mathcal{G}_t)$ between two graphs. That is:

$$K_D(\mathcal{G}_m, \mathcal{G}_t) = \sum_{v_i, v_j \in \mathcal{V}_t, \mathcal{E}_{i,j} \in \mathcal{E}_t} -\log(v_j | v_i, \mathcal{Z}_m). \quad (27)$$

Random Walk graph Neural Networks (RWNN) [95] also derives a trainable random walk kernel $K_{RW}(\cdot, \cdot)$ through a series of learnable graph patterns $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_M\}$. A learnable graph \mathcal{G}_m has a fixed node set \mathcal{V}_m but a changeable edge set \mathcal{E}_m . RWNN produces graph-level embeddings $\mathbf{h}_{\mathcal{G}_t} = [K_{RW}(\mathcal{G}_1, \mathcal{G}_t), \dots, K_{RW}(\mathcal{G}_M, \mathcal{G}_t)]^\top$ of the target graph \mathcal{G}_t for graph classification tasks. Correspondingly, the classification results are backpropagated to change the adjacency matrix of learnable graph patterns. That is, RWNN uses the prediction results to train the input of the kernel function (i.e., graph patterns) so that the kernel values can be learned according to the downstream task.

6.4 Spectral-Based GL-GNNs

Spectral-based GL-GNNs were started earlier by Bruna *et al.* [96], which designed graph convolutions via the spectral graph theory [53]. Recently, Balcilar *et al.* [97] described spectral and spatial graph convolution in a unified way and performed spectral analysis on convolution kernels. The analysis results demonstrate that a vast majority of MPNNs are low-pass filters in which only smooth graph signals are retained. Graph signals with a low-frequency profile are useful for node-level tasks on assortative networks where nodes have similar features to their neighborhoods [98]. However, with graph-level tasks, graph signals beyond the low frequency may be critical since they can highlight the differences between different graphs [99], and, although MPNNs have been widely used, they overlook the signal frequency of graph data.

In terms of a feature $\mathbf{x} \in \mathbb{R}^n$ (a column vector of $\mathbf{X} \in \mathbb{R}^{n \times f}$) as a graph signal on a graph with n nodes, spectral graph convolution performs graph signal filtering after transforming the graph signals \mathbf{x} in spatial space into the frequency domain. According to spectral graph theory [53], the frequency domain generally takes the eigenvectors of the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$ where \mathbf{D} is the degree matrix (or the normalized version $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$) of a set of space bases. Note, though, that other bases can also be used, such as graph wavelet bases [100, 101]. Specifically, $\{\lambda_1, \dots, \lambda_n\}$ where $0 \leq \lambda_1 \leq \dots \leq \lambda_n \leq 2$, and $\mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_n)$ are the n eigenvalues and n orthogonal eigenvectors of \mathbf{L} , respectively. λ_i represents the smoothness degree of \mathbf{u}_i about \mathbf{L} . Based on the graph Fourier transformation $\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}$, the graph signal \mathbf{x} is mapped to the frequency domain. And $\mathbf{x} = \mathbf{U} \hat{\mathbf{x}}$ is the graph Fourier inverse transformation that can restore the graph signal in spectral domain to the spatial domain. The polynomial filter is adopted by most of spectral graph convolution methods, for example, ChebNet [102] defines the spectral graph convolution as $\text{Udiag}(\Phi(\Lambda)) \mathbf{U}^\top \mathbf{x}$, where $\Lambda = \text{diag}(\{\lambda_i\}_{i=1}^n)$, $\Phi(\Lambda) = \sum_{k=0}^K \theta_k \Lambda^k$ is the polynomial filtering function, K are the hyper-parameters that realize the localized spectral graph convolution, and θ_k is the polynomial coefficient.

Spectral graph convolution can be task-agnostic when graph signals with any frequency profiles are filtered. Conversely, they can also be task-specific—for example, a band-pass filter can highlight graph signals that are strongly relate to downstream tasks [97]. However, only a few practitioners

have designed graph-level neural networks from the perspective of spectral graph convolution [103, 104]. The main problem with applying spectral convolution in graph-level tasks is the transferability of the spectral filter coefficients from the training graph set to the unseen graphs. The spectral filters depend on the graph Laplacian decomposition, but different graph structures have different graph Laplacian decomposition results. Most recently, Levie *et al.* [105] theoretically proved the transferability of spectral filters on multigraphs. Balcilar *et al.* [103] proposed a custom filter function that could output frequency components from low to high to better distinguish between graphs. Due to the limitation of polynomial filters in modeling sharp changes in the frequency response, Bianchi *et al.* [104] employed an auto-regressive moving average (ARMA) filter to perform graph-level tasks. The ARMA filter is more robust to the changes or perturbations on graph structures as it does not depend on the eigen-decomposition of the graph Laplacian explicitly. In addition, Zheng *et al.* [106] proposed a graph convolution based on graph Framelet transforms instead of graph Fourier transform with a shrinkage activation to decompose graphs into both low-pass and high-pass frequencies. However, there is no theoretical proof of the transferability of framelet decomposition.

7 GRAPH POOLING

Generally, deep graph-level learning methods encode graphs based on node representations. Graph pooling is a technique that integrates node embeddings into a graph embedding. In this section, we introduce two mainstream types of graph pooling techniques, i.e., global and hierarchical graph pooling (see Section 7.1 and 7.2). We summarize all discussed pooling approaches in Table 5 in Appendix E. Moreover, we discuss the effectivity of graph pooling (see Section E.1).

7.1 Global Graph Pooling

There are four different types of global graph pooling –numeric operation, attention-based, CNN-based, and global top- K –all of which aggregate all node embeddings at once to build a graph-level representation.

7.1.1 Numeric Operation. Adopting a simple numeric operation for all node embeddings is a common graph pooling method [81, 107], since it is easy to use and obeys the permutation invariant. An illustration of a type of numeric operation (i.e., a summation) for all node embeddings is shown in Fig. 6 (A). It is common to see practitioners aggregating node embeddings via summation, maximization, minimization, mean, and concatenation functions. For example:

$$\mathbf{h}_{\mathcal{G}} = \sum_{v \in \mathcal{V}} \mathbf{h}_v / \max_{v \in \mathcal{V}} / \min_{v \in \mathcal{V}} (\mathbf{h}_v) / \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_v / [\mathbf{h}_{v_1} | \dots | \mathbf{h}_{v_{|\mathcal{V}|}}]. \quad (28)$$

Duvenaud *et al.* [81] empirically proved that, in graph-level learning, summation has no weaker an outcome than a hash function. Similarly, GIN [107] shows us that the injective relabeling function in the WL algorithm can be replaced with a simple numeric operation. Further, GIN also allows us to analyze the efficacy of different functions: summation, maximization, and mean functions. Summation comprehensively summarizes the full features and structure of a graph. Maximization emphasizes significant node embeddings, and mean learns the distribution of labels. Inspired by GIN, Principal Neighbourhood Aggregation (PNA) [108] employs all three of these functions to pool the node embeddings, while TextING [109] includes both mean and maximization pooling to capture the label distribution and strengthen the keyword features. A few variants of graph pooling have also been developed. For example, Deep Tensor Neural Network (DTNN) [80] applies a neural layer that processes the node embeddings before the summation function and second-order pooling (SOPOOL) [110] is executed as $\mathbf{h}_{\mathcal{G}} = [\mathbf{h}_{v_1}^T \mathbf{h}_{v_1} | \dots | \mathbf{h}_{v_{|\mathcal{V}|}}^T \mathbf{h}_{v_{|\mathcal{V}|}}]$.

7.1.2 Attention-based. The contributions of node embeddings to graph-level representations may not be equal, as some of them contain may more important information than others. Hence, some researchers have tried using an attention mechanism to aggregate the node embeddings based on their particular contribution, as outlined in Fig. 6 (C). Li *et al.* [69] and Duvenaud *et al.* [81], for example, both employ a softmax function as an attention-based global pooling for aggregation. This can be written as:

$$\mathbf{h}_G = \sum_{v,k} \text{softmax} \left(w_v^k, \mathbf{h}_v^k \right), \quad (29)$$

where w_v^k is a trainable weight for the embedding h_v^k of node v in iteration k . Note that w_v^k will be large if h_v^k is important to the downstream task. Set2Set [111] is a more complicated attention-based graph pooling model. It learns the attention coefficients of all node embeddings from an ordered sequence generated by LSTM. Although Set2Set handles sequential node embeddings, the order of nodes is determined by an LSTM model without affecting permutation invariance.

7.1.3 CNN-based. PATCHY-SAN [70] and KCNN [71] are based on the idea of ordering vertices and applying a 1-D convolutional layer to pool the ordered vertices features. These two models are permutation invariant because they order vertices according to certain rules regardless of the input order.

7.1.4 Global Top-K. Global top- K graph pooling sorts all nodes and selects the first K node embeddings for aggregation, as shown in Fig. 6 (B). In this way, the pooling layer only preserves K significant vertices and drops out others. SortPool [112] employs graph convolution operations to project each node into a one-dimensional vector as the ranking score for selecting the K vertices with the highest scores. Subsequently, a GL-GNN is used to produce the node embeddings of the selected K nodes, which come together to form the graph-level representation. Graph Self-Adaptive Pooling (GSAPool) [113] is another global top- K graph pooling model that ranks nodes based on the summing of feature and structure scores. The node structure scores are 1-dimensional vectors projected by the graph convolution operations as same as SortPool, while the feature scores are learned by feeding the node features into an MLP.

7.2 Hierarchical Graph Pooling

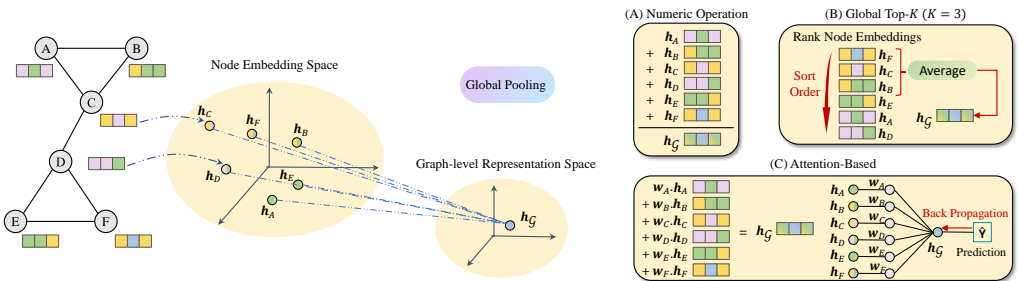


Fig. 6. Toy examples of Global Pooling methods.

Global graph pooling ignores the hierarchical structures in graphs. The evolution of a graph is to collect nodes into hierarchical structures (e.g., communities), then to form the graph. Hence, researchers tend to capture hierarchical information through an aggregation process that has multiple parses, which coarsens the graph each time. We have divided hierarchical graph pooling techniques into three branches: clustering-based, hierarchical top- K , and tree-based.

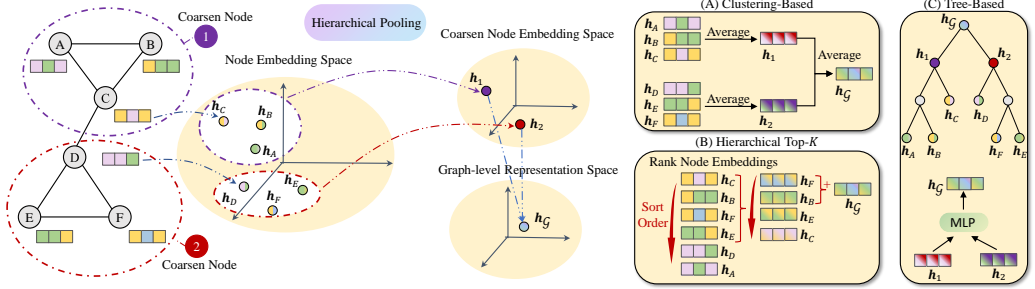


Fig. 7. Toy examples of Hierarchical Pooling methods.

7.2.1 Clustering-based. Clustering methods were originally designed to capture the hidden hierarchical structures in graphs, but these techniques can be incorporated into the pooling process. Fig. 7 (A) demonstrates clustering-based graph pooling, which has been the focus of many studies. For instance, Henaff *et al.* [114] implemented multi-resolution spectral clustering [115] which assigns each node to a matched cluster. Subsequently, the clusters in the input graph are treated as new nodes of the new coarsened graph. The embedding of the new node is obtained by averaging all node embeddings in the cluster. This coarsening process is iterative and operates until only one or very few vertices remain in the most recent coarsened graph. Similarly, Bruna *et al.* [116] adopted hierarchical agglomerative clustering [117] to coarsen graphs, while StructPool [118] employs conditional random fields [119] to cluster each node by considering the assignments of other vertices.

However, clustering-based graph pooling cannot optimize the clustering process for downstream tasks given just any old off-the-shelf clustering method. Rather, the clustering method must be designed to consider downstream tasks. For example, Graph Multiset Transformer (GMT) [120] uses a multi-head self-attention mechanism to cluster nodes into different sets according to the final task and a graph-level representation is therefore derived through these sets. MinCutPool [121] assigns each node to a cluster via an MLP, which is optimized by two goals: first that the clusters are similar in size, and, second, that the clusters' embeddings are separable. Finally, the graph-level representation is obtained by pooling the substructure-level embeddings. EigenPool [122] involves a spectral clustering method that coarsens graphs and pools node embeddings into cluster-level embeddings by converting spectral-domain signals. These clustering-based algorithms assume that each node belongs to a certain cluster, yet DiffPool [123] assigns each node to multiple clusters through a trainable soft assignment matrix $\mathbf{S}^{(k)} \in \mathbb{R}^{n^{(k)} \times n^{(k+1)}}$, where $n^{(k)}$ is the number of vertices in the input graph at the k -th layer, and $n^{(k+1)}$ represents the cluster's number in the input graph or the node's number in the coarsened graph. To be specific, at the k -th layer, each row of $\mathbf{S}^{(k)}$ corresponds to a node in the input graph, and each column of $\mathbf{S}^{(k)}$ corresponds to a new node in the coarsened graph (i.e., a cluster in the input graph). The assignment matrix $\mathbf{S}^{(k)}$ is trained by a graph convolutional layer, which is defined as:

$$\mathbf{S}^{(k)} = \text{softmax} \left(\text{Conv}^{(k)} \left(\mathbf{A}^{(k)}, \mathbf{H}^{(k)}, \mathbf{W}^{(k)} \right) \right), \quad (30)$$

where $\mathbf{A}^{(k)} \in \mathbb{R}^{n^{(k)} \times n^{(k)}}$ and $\mathbf{H}^{(k)} \in \mathbb{R}^{n^{(k)} \times f}$ are the adjacent matrix and node embedding matrix of the input graph at the k -th layer, respectively. $\mathbf{W}^{(k)} \in \mathbb{R}^{f \times n^{(k+1)}}$ is the trainable weight matrix, and $\text{softmax}(\cdot)$ function is applied to each row.

7.2.2 Hierarchical Top- k . The high complexity of the clustering process exacerbates the computational cost burden of cluster-based hierarchical graph pooling. For example, the DiffPool [123] is extremely costly in terms of time and memory because the assignment matrices need to be trained. So, to speed up the process of hierarchical graph pooling, researchers have looked to replace the clustering process with a scheme that coarsens the graph according to the top- K idea, as shown in Fig. 7 (B). Graph U-nets [124], for example, projects each node feature into a 1-dimensional vector Y , as the rank score. Subsequently, the K nodes with the highest score are selected to form the new coarsened graph, which is defined as:

$$Y = \frac{Z^{(l)}P^{(l)}}{\|P^{(l)}\|}, \quad \text{idx} = \text{Top } K(Y), \quad Z^{(l+1)} = \left(Z^{(l)} \odot (\text{sigmoid}(Y_{\text{idx}}) \mathbf{1}_Z^T) \right), \quad A^{(l+1)} = A_{\text{idx}, \text{idx}}^{(l)} \quad (31)$$

where $Z^{(l)}$ is the input node features at the layer l , $P^{(l)}$ is a learnable projection matrix, $\text{Top } K(\cdot)$ is a function that returns the index of the top- K nodes, and all the elements are 1 in the vector $\mathbf{1}_Z$ which has the same dimension as the node feature. Cangea *et al.* [125] employed the Graph U-nets to coarsen graphs and concatenated the mean and maximum values of node embeddings on the coarsened graphs as graph-level representations. Further, SAGPool [126] chooses the top- K nodes to generate the coarsened graph by adopting a graph convolution operation to project node features as scores.

All these methods generate a coarsened graph by preserving the top- K nodes. However, Ranjan *et al.* [127] presented the novel idea of ranking the clusters and preserving on the top- K of them. The clusters were ranked by employing a self-attention algorithm called Master2Token [128] that scores each cluster based on the node embeddings within it.

7.2.3 Tree-based. Tree-based hierarchical graph pooling implements the coarsening process via an encoding tree, where the input graph is coarsened layer by layer to the ultimate node from the leaf layer to the root layer, as shown in Fig. 7 (C). ChebNet [102] and MoNet [129] use the Graclus [130] algorithm to pair nodes in the graph based on the graph spectrum and merge the pair-wise nodes as a new node in the coarsened graph. That is to say, these two methods build a balanced binary tree to coarsen the graph, and each father node on the tree is obtained by coarsening its two child nodes. Wu *et al.* [131] uses a structure encoding tree [132] for tree-based hierarchical graph pooling. Structural coding trees compress the hierarchy of a graph into a tree. Here, the leaves are the nodes, the root represents the whole graph, and the other non-leaf nodes are the hierarchical structures (e.g., the communities). An MLP merges the features of the child nodes in the structure encoding tree, to generate an embedding of the father node. The result is an embedding of the root node, which serves as a graph-level representation. Moreover, Wu *et al.* [133] empirically verified that the hierarchical tree pooling guided by structure entropy can preserve higher-quality structural information than U-Nets and MinCutPool. Alternatively, EdgePool [134] scores edges based on the features of the nodes the edges link, eliminating the highest-ranked edge by merging its two end nodes. The features of the newly generated node, which maintains all the neighbors of the original two nodes, are obtained by summing the features of the two merged nodes. EdgePool falls into the category of being a tree-based hierarchical graph pooling method because it merges two child nodes in a tree into a father node.

8 BENCHMARKS

8.1 Datasets

Table 1 summarizes a selection of benchmark graph-level datasets, including TUDataset [136], Open Graph Benchmark (OGB) [141], MOLECULENET [2], MALNET [157], and others [150]. The graph

Table 1. Summary of Selected Benchmark Datasets

Category	Dataset	Size	#Graphs	Average #Nodes	Average #Edges	Node Attr.	Edge Attr.	#Classes	Source
Biology	ENZYMES	Small	600	32.6	62.1	✓	-	6	[135, 136]
	PROTEINS	Small	1113	39.1	72.8	✓	-	2	[135, 136]
	D&D	Small	1178	284.3	715.7	✓	-	2	[136, 137]
	BACE	Small	1513	34.1	36.9	✓	✓	2	[2, 138]
	MUV	Medium	93087	24.2	26.3	✓	✓	2	[2, 139]
	ppa	Medium	158100	243.4	2266.1	-	✓	37	[140, 141]
Chemistry	MUTAG	Small	188	17.9	19.8	✓	✓	2	[41, 136]
	SIDER	Small	1427	33.6	35.4	✓	✓	2	[2, 142]
	ClinTox	Small	1477	26.2	27.9	✓	✓	2	[2, 143]
	BBBP	Small	2039	24.1	26.0	✓	✓	2	[2, 144]
	Tox21	Small	7831	18.6	19.3	✓	✓	2	[2, 145]
	ToxCast	Small	8576	18.8	19.3	✓	✓	2	[2, 146]
	MolHIV	Small	41127	25.5	27.5	✓	✓	2	[2, 141]
	MolPCBA	Medium	437929	26.0	28.1	✓	✓	2	[2, 141]
	FreeSolv	Small	642	8.7	8.4	✓	✓	-	[2, 147]
	ESOL	Small	1128	13.3	13.7	✓	✓	-	[2, 148]
	Lipophilicity	Small	4200	27.0	29.5	✓	✓	-	[2, 149]
	AQSOL	Small	9823	17.6	35.8	✓	✓	-	[150, 151]
	ZINC	Small	12000	23.2	49.8	✓	✓	-	[150, 152]
	QM9	Medium	129433	18.0	18.6	✓	✓	-	[2, 136]
Social Networks	IMDB-BINARY	Small	1000	19.8	96.5	-	-	2	[136, 153]
	IMDB-MULTI	Small	1500	13.0	65.9	-	-	3	[136, 153]
	DBLP_v1	Small	19456	10.5	19.7	✓	✓	2	[136]
	COLLAB	Medium	5000	74.5	2457.8	-	-	3	[136, 153]
	REDDIT-BINARY	Small	2000	429.6	497.8	-	-	2	[136, 153]
	REDDIT-MULTI-5K	Medium	4999	508.5	594.9	-	-	5	[136, 153]
Computer Science	REDDIT-MULTI-12K	Medium	11929	11.0	391.4	-	-	11	[136, 153]
	CIFAR10	Medium	60000	117.63	941.1	✓	-	10	[150, 154]
	MNIST	Medium	70000	70.57	564.53	✓	-	10	[150, 155]
	code2	Medium	452741	125.2	124.2	✓	✓	-	[141, 156]
	MALNET	Large	1262024	15378	35167	-	-	696	[157]

* The category of computer science includes computer vision, cybersecurity, and program coding datasets.

* Node Attr. and Edge Attr. indicates the labels or features of nodes and edges, respectively.

* The size of datasets follows the setting of OGB [141], medium datasets have more than 1 million nodes or more than edges, and large datasets own over 100 million nodes or 1 billion edges.

datasets collected by the group at TUDataset [136] have been widely used to evaluate graph-level learning approaches. These graph datasets consist of molecules, proteins, images, social networks, synthetic graphs, and data from many other domains. However, despite their wide use, they have attracted criticism from some practitioners. For example, Ivanov *et al.* [158] contends that the sets suffer from isomorphism bias, i.e., they contain isomorphic graphs with different labels, which may hinder model training—a claim based on the analysis of 54 widely-used graph datasets. They also note that some of the datasets are too small to train a data-hungry deep learning model. For example, Dwivedi *et al.* [150] presented that most GL-GNNs have a close performance to others in the small dataset. Further, some topology-agnostic baselines yield a performance that is competitive to GL-GNNs.

Developing practical and large-scale benchmark datasets has become an important issue for the graph-level learning community. To this end, Wu *et al.* [2] proposed a benchmark named MOLECULENET that contains a set of large-scale graph datasets of molecules. The dataset is

designed to be used for graph regression and classification tasks. Dwivedi *et al.* [150] transformed images into graphs for classification, in which a group of pixels is clustered as a node. Based on real-world cybersecurity scenarios, Freitas *et al.* [157] proposed a large-scale graph dataset of over 1.2 million graphs with imbalanced labels. Furthermore, OGB [141] has published application-oriented large-scale graph datasets of molecules, proteins, and source code cooperation networks.

8.2 Evaluations

The development of graph-level learning has been impeded by unfair evaluations. For example, Ruffinelli *et al.* [159] argue that some graph-level learning models only produce state-of-the-art performance because of tricks with the model’s training, not because of the novel ideas proposed in the articles. However, there is no consensus on which evaluation to use with the most widely used graph datasets, such as TUDatasets, nor is there even a universally-accepted data split [160]. Hence, to evaluate the graph-level learning models in a unified and fair way, some researchers have attempted to establish a standard model evaluation protocol. For example, Dwivedi *et al.* [150] built a benchmark framework based on PyTorch and DGL⁷ that evaluates models on graph classification and graph regression tasks with a unified model evaluation protocol. They do apply training tricks, such as batch normalization, residual connections, and graph size normalization, to GL-GNNs to measure their effects. But all models being evaluated with the protocol are subject to the same training regime. Similarly, in addition to the large-scale graph datasets, OGB [141] provides a standard model evaluation protocol that includes a unified way to load and split data, the model evaluation itself, plus the cross-validations. Recently, Zhu *et al.* [161] provides a benchmark framework for graph contrastive learning.

9 DOWNSTREAM TASKS AND APPLICATIONS

This section introduces the mainstream downstream tasks of graph-level learning and their corresponding applications.

Graph Generation. This task aims to generate new graphs that have specific properties based on a series of graphs. Graph generation has a broad application in the field of biochemistry. For instance, *drug development* involves experimenting with a tremendous number of molecule arrangements, but, through graph generation, the overall time and investment required to do this work can be reduced [10]. Similarly, *molecule generation* [162, 163] has been used to explore new catalysts [164]. Sanchez *et al.* [165] applied graph generation into *physical systems modeling* to simulate real-world particle motions. *Scene graph generation* [166, 167] can be used to understand the scene of images and generate abstraction for images to summarize the relationship among objects in an image. Most recently, a few works [168, 169] have employed graph generation for *program debugging*, which modifies the nodes (i.e., variables or functions) and links in the program flow graph to fix bugs.

Graph Classification. The goal of graph classification is to learn the mapping relationship between graphs and corresponding class labels and predict the labels of unseen graphs. Graph classification is a critical graph-level learning task with a range of applications. For example, classifying molecular graphs [9, 170] can be used to determine anti-cancer activity, toxicity, or the mutagenicity of molecules. Classifying protein graphs [135] can help to identify proteins with specific functions, such as enzymes. By converting texts to graphs in which nodes denote words and edges are the relationships between words, *text categorization* [171, 172] can distinguish documents with different topics. By the same token, pixels in images can be regarded as nodes and adjacent pixels are linked to yield graphs for *image recognition* [173, 174]. This task can be extended to *medical diagnosis* to deal with computed tomography scans [175] and clinical images [176]. In addition,

⁷DEEP GRAPH LIBRARY: <https://www.dgl.ai>

graph classification can also be used for *online product recommendation* [177] and *fake news detection* [178, 179]. Recently, it has been impressive to see that graph-level learning can deliver *IQ tests* [180] that select graphs with a specific style from a group of graphs based on the style learning on the other group of graphs.

Graph Comparison. This task involves measuring the distance or similarity of pair-wise graphs in a graph dataset. The applications of this task include: *semantic inference*, which infers text-document affiliations [181]; matching images with texts describing the same thing [182]; *semantic metrics*, which measures the semantic similarity between texts [183, 184]; and *cross-language information retrieval*, which seeks information in a language context that is different from the query [185, 186].

Graph Regression. This task aims to predict the continuous proprieties of graphs. Taking molecules as examples, graph regression can predict different molecular proprieties related to the tightness of chemical bounds, fundamental vibrations, the state of electrons in molecules, the spatial distribution of electrons in molecules, and so on [21, 187, 188]. Hence, the most promising application of graph regression is *drug discovery*. In addition, employing graph regression to predict ratings or avenues of films is feasible.

Subgraph Discovery. This is the task of detecting discriminative substructures in a graph dataset. Subgraph Discovery can be applied to *molecular structure search* [81, 189], which explores the functional structures in chemical compounds, or to *social event detection* [190], where subgraph discovery can be used to detect the substructures that represent great events in a series of social networks.

Applying Complex Scenarios. In addition to simple downstream tasks, researchers have extended graph-level learning to some complex scenarios. For instance, *multi-view GL* targets learning in scenarios where an object is described by multiple graphs (i.e., multi-graph-views). In multi-view GL, practitioners mine information from each single-graph-view and then strategically fuse information from all graph-views [191, 192]. *Multi-task GL* [193, 194] is generally used to optimize multiple related tasks; hence, it focuses on detecting the discriminative features across all the different tasks. In real-world scenarios, there are a vast number of unlabeled graphs that go unused since most GL techniques require learning from labeled information. Consequently, *semi-supervised GL* [49] was developed, which can learn from a dataset containing only a few labeled graphs and very many unlabeled graphs. Likewise, *positive and unlabeled GL* [195, 196] only requires a few labeled graphs in one class along with other unlabeled graphs. In terms of dynamic scenarios, there are also applications that record changing graphs over time as graph streams. For example, a paper and its references can be regarded as a citation graph, and a graph stream can be produced of citations in chronological order of the corresponding papers. *Graph stream GL* [197, 198], for example, is specifically designed for graph stream data and mines valuable patterns from dynamic graph records.

10 FUTURE DIRECTIONS

Although graph-level learning has gone through a long journey, there are still open issues that have been less explored. In this section, we spotlight 12 future directions involving technical challenges and application issues of graph-level learning for readers to refer to.

10.1 Neural Architecture Search (NAS) for GL-GNNs

Existing GL-GNNs often have a complex architecture, consisting of a number of different components, e.g., multiple graph convolutions and graph pooling layers. GL-GNNs require careful parameter tuning to achieve optimal performance since most of them are non-convex. Hence, it is expensive to search for a well-performing architecture from among the bulk of optional components and their numerous parameters.

Opportunities: Developing effective NAS methods to free researchers from the task of repeatedly searching for good architectures manually and, in turn, tuning the parameters is an urgent goal. By minimizing the entropy, Yang *et al.* [199] raised a dimension estimator, which can empower the GL-GNNs to automatically encode graphs into suitable dimensional embeddings. Moreover, Knyazev *et al.* [200] modeled the search for an architecture as a graph in which each node represents a neural network layer or operation (e.g., a convolution layer) and each edge represents the connectivity between a pair of operations. Subsequently, GNNs can work on the constructed graphs to seek the optimal architecture. We argue that constructing an optimization goal based on knowledge of deep learning might be a practical way of providing an automatic NAS for various GL-GNNs.

10.2 Geometrically Equivariant GL-GNNs

In geometric graphs [201], each node is described by two vectors, i.e., a feature vector and a geometric vector. For example, in 3D molecule graphs, atoms are assigned geometric information such as speeds, coordinates, and spins which together comprise the geometric vector. Constructing GL-GNNs that can learn geometric graphs is an important part of modeling in chemistry and physics.

Opportunities: GL-GNNs that can predict a set of geometric graphs need to be equivariant. For example, when inputting a geometric graph with a specific rotation into a GL-GNN, the corresponding output should reflect the same rotation. There are some algorithms about geometrically equivariant GL-GNNs. For example, Satorras *et al.*'s [202] Equivariant Graph Neural Networks (EGNN) expands MPNNs aggregating both feature vectors and geometric vectors, while GemNet [203] infuses more geometric information into the message passing mechanism, like dihedral angles. Both of these methods achieve state-of-the-art performance with 3D molecule prediction tasks. For more details on this topic, we refer readers to [204].

10.3 Self-explainable GL-GNNs

Most algorithms for explaining the predictions of GL-GNNs are post-hoc (e.g., PGExplainer [205]), where the aim is to train a model to interpret a pre-trained GL-GNN. In other words, the training and explaining processes in GL-GNNs are independent.

Opportunities: Miao *et al.* [206] proposed that the separate prediction and explanation processes will inevitably lead to sub-optimal model performance. For example, the explanation model may detect substructures that have spurious correlations to the graph labels when interpreting predictions [207]. Designing self-explaining GL-GNNs where the prediction and explanation components enhance each other should therefore be a fruitful future direction of research for the graph-level learning community.

10.4 Informative Graph Pooling

We categorized the existing pooling techniques into two families, i.e., global and hierarchical pooling (see Section 7). The aim of the top- k approaches [112, 113], which are among the most representative global pooling methods, is to select some nodes for the pooled graph. However, one cannot ensure that the redundancy of the selected nodes will be low. Further, the mechanism of the hierarchical family tends to smooth the node representations, which means the uninformative nodes tend to be selected for the pooled graphs [208].

Opportunities: Existing state-of-the-art graph pooling methods are not able to coarsen the original graph into a pooled graph with nodes of low redundancy. However, a pooled graph consisting of dissimilar nodes is critical for graph-level learning. For example, an atomic pair composed of different atoms can empower different proprieties to molecules. Traditional subgraph mining methods [48, 177] can then be used to identify the discriminative subgraphs of low redundancy as

representative graphs. Hence, referring to the ideas of traditional subgraph mining for graph pooling methods that can identify informative nodes and/or subgraphs might yield feasible solutions to this problem.

10.5 Graph-level Federated Learning

Graph data are generally sourced from information collected by institutions. However, due to privacy considerations, graph data from different institutions is generally not used to jointly train graph-level learning models. In practice, numerous graph-level learning techniques are data-hungry, especially the currently mainstream GL-GNNs. Therefore, it is a practical topic to promote the joint training of graph-level learning models by different institutions using their respective graph data.

Opportunities: Federated learning solves the data isolation problem, feeding data-driven machine learning models from different sources with rich amounts of data while maintaining privacy. For example, Xie *et al.* [209] proposed a federated learning framework specifically for GL-GNNs, where different GL-GNNs are trained based on different graph sets and sharing weights are learned by the GL-GNNs. Graph-level federated learning is an emerging topic with great challenges. In fact, a benchmark for this task has recently been released [210].

10.6 Graph-level Imbalance Learning

A machine learning model trained on the data with an imbalanced label distribution might be biased towards the majority classes. That is, with many samples and the minority classes consisting only of a small number of samples, the model may be under-fit. Representative tasks that need imbalance learning and must distinguish between samples from the majority and minority classes include anomaly detection [211] and long-tail event detection [212].

Opportunities: Although imbalanced learning has been a long-standing issue in deep learning, graph-level imbalance learning, especially with deep models, is underexplored. Wang *et al.* [213] over-sampled graphs in the minority class to relieve imbalance distributions between the majority and minority classes. They also appended a self-consistency between the original and the augmented graphs. Over-sampling the minority samples is a traditional solution to imbalanced learning. However, this approach has been criticized for some shortcomings, such as over-fitting and changing the original distribution of the dataset. Additionally, minority graphs generally contain special substructures that are different from those in the majority graphs. Strengthening the structural awareness of the current graph-level learning tools could be a feasible way of overcoming this problem.

10.7 Graph-level Learning on Complex Graphs

In this survey, almost all the investigated graph-level learning methods are assumed to work on fundamental graphs (i.e., unweighted and undirected graphs and their nodes and edges are homogeneous). This is because fundamental graphs are easy to understand and easy for models to handle. However, realistic graphs are usually complex. For example, the edges between actors and movies have a different meaning to the edges between two movies in multi-relational graphs. Collaborators on a paper can be linked together by a hyperedge (i.e., hypergraphs), while authors, papers, and venues can all be nodes in a citation network, even though they are distinct taxonomic entities (i.e., heterogeneous graphs), etc.

Opportunities: Compared to highly developed graph-level learning on fundamental graphs, mining complex graphs still requires further development. For instance, most GL-GNNs for heterogeneous graphs rely on manually-defined meta-paths (i.e., a sequence of relations between nodes or edges) that are based on domain knowledge. However, defining meta-paths is not only expensive, it will not capture comprehensive semantic relationships [214, 215]. Lv *et al.* [216] also raised the issue

that, empirically, some heterogeneous GL-GNNs do not perform as well as simple GL-GNNs. In addition, it is hard to fairly evaluate hypergraph GL-GNNs since the hypergraphs are acquired from a range of different sources and built by a range of different construction approaches. In conclusion, there are numerous worthwhile directions to explore when it comes to graph-level learning with complex graphs, such as benchmarking evaluation [216] and datasets.

10.8 Graph-level Interaction Learning

Almost all the literature on graph-level learning treats each graph in a dataset as an independent sample. However, considering the interactions between graphs should lead to challenging and highly novel research. For example, learning the interactions between graphs might be used to predict the chemical reactions when two compounds meet or to explore the effect of taking two or more drugs at the same time.

Opportunities: Although this topic has strong practical implications for graph-level learning applications in biochemistry, it is still understudied. So far, only a few GL-GNNs have been designed to tackle this topic and its related tasks. DSS-GNN [88], for instance, predicts the interactions between subgraphs located in a single graph, while Graph of Graphs Neural Network (GoGNN) [217] predicts chemical-chemical and drug-drug interactions. These two tasks own the off-the-shelf datasets, DDI [218], CCI [219], and SE [218].

10.9 Graph-level Anomaly Detection

The aim of anomaly detection is to identify objects that significantly deviate from the majority of other objects. However, when it comes to graph-structured data, almost all graph anomaly detection research focuses on detecting anomalous nodes in a single graph [15].

Opportunities: Graph-level anomaly detection that identifies anomalous graphs in a graph dataset is a research topic of great value application-wise. For example, such a method could help to detect proteins with special functions from a large number of common protein structures. Some pioneering studies [220–222] combine state-of-the-art GL-GNNs with traditional anomaly detection methods (e.g., one-class classification [223]) to detect anomalous graphs in a graph dataset. However, these graph convolution operations were not specifically designed to detect anomalous graphs. Most graph convolution works like a low-pass filter [97] that smooths the anomalous information in a graph [224]. Hence, more analysis of the reasons behind anomalous graphs is needed and specific graph convolutions need to be proposed that are purposefully designed to detect anomalous graph information manifested in graph structures and/or attributes.

10.10 Out-of-Distribution Generalization

Out-of-distribution (OOD) learning improves a model's generalization ability. It applies to scenarios where the test data does not have the same distribution as the training data. OOD settings can have two types of distribution shift, concept shift and covariate shift. Concept shift refers to situations where the conditional distribution between the inputs and outputs differs from the training data to the test data. Covariate shift means that the test data has some certain features not shown in the training data.

Opportunities: Almost all the graph-level learning algorithms assume that the training and the test data will have the same distribution. However, this I.I.D. (independent, identically distributed) assumption may be violated in some scenarios. For example, molecules with the same function may contain some different scaffolds. When the test data have a scaffold that has never appeared in training data, graph-level learning methods models will not perform nearly as well. The graph-level learning community has recently noticed this issue and has embarked on related research in response. Gui *et al.* [225] proposed a graph OOD learning benchmark. Inspired by invariant learning,

Wu *et al.* [207] identified the casual subgraphs that are invariant across different distributions to improve the OOD generalization ability of GL-GNNs. Similarly, Bevilacqua *et al.* [226] employed an inference model to capture approximately invariant causal graphs to improve the extrapolation abilities of GL-GNNs. In addition to invariant learning, many techniques such as meta-learning, data augmentation, and disentanglement learning are feasible for OOD learning. Combining these techniques with GL-GNNs is likely to be the future of achieving graph-level learning models with a strong OOD generalization capacity.

10.11 Brain Graphs Analytics

Brain networks, also known as connectomes, are maps of the brain where the nodes denote the brain regions of interest (ROIs) in the brain and the edges denote the neural connections between these ROIs. An important application of machine learning models pertaining to brain networks is to distinguish brains with neurological disorders from normal individuals and identify those regions of the brain that are the cause of brain disease.

Opportunities: Existing graph-level learning algorithms especially GL-DNNs and GL-GNNs, tend to be over-parameterized for learning brain networks, which are usually sparse. Further, obtaining a brain network usually comes at a high cost, because it involves scanning an individual's brain and converting the neuro-image into a brain network. In addition, existing GL-DNNs and GL-GNNs cannot handle the correspondence of nodes between different graphs. However, different brain networks have the same ROIs, and node identities and ROIs are one-to-one correspondence [227]. In summary, graph-level learning with brain networks requires models that are lightweight and can identify corresponding nodes between different graphs.

10.12 Multi-Graph-Level Learning

Standard graph-level learning views each graph as an instance, which can be restrictive in practical applications. Considering a product that has multiple reviews on an online shopping page. Each review can be represented as a graph of the textual semantics among the words. To predict any properties of that online product, one needs to learn from review-based multi-graphs—that is, multi-graph-level learning.

Opportunities: To the best of our knowledge, the current multi-graph-level learning algorithms are all traditional. For example, Boosting based Multi-graph Graph Classification (bMGC) [228] and Multi-Instance Learning Discriminative Mapping (MILDM) [229] are both subgraph mining methods that classify multi-graph objects by extracting informative subgraphs. However, both two methods cannot use label information to guide the feature selection process. Developing deep learning models can better extract features for multi-graph-level learning via the label information.

11 CONCLUSIONS

This survey paper provides a comprehensive review of graph-level learning methods. Due to the irregular structure of graphs, graph-level learning has long been a non-trivial task with related research spanning the traditional to the deep learning era. However, the community is eager for a comprehensive taxonomy of this complex field. In this paper, we framed the representative graph-level learning methods into four categories based on different technical directions. In each category, we provided a detailed discussion on, and comparison of, the representative methods. We also discussed open-source materials to support research in this field, including datasets, algorithm implementations, and benchmarks, along with the most graph-level learning tasks and their potential industrial applications. Lastly, we raised 12 future directions based on currently open issues that would make valuable contributions to the graph-level learning community.

REFERENCES

- [1] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Comment. Acad. Sci. Imp. Petropol.*, pp. 128–140, 1741.
- [2] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chem. Sci.*, vol. 9, no. 2, pp. 513–530, 2018.
- [3] F. Harary, "The four color conjecture and other graphical diseases," *Proof Techniques in Graph Theory*, pp. 1–9, 1969.
- [4] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI Series*, vol. 2, no. 9, pp. 12–16, 1968.
- [5] B. D. McKay and A. Piperno, "Practical graph isomorphism, ii," *J. Symb. Comput.*, vol. 60, pp. 94–112, 2014.
- [6] L. Babai, "Graph isomorphism in quasipolynomial time," in *Proc. STOC*, pp. 684–697, 2016.
- [7] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proc. LNAI*, pp. 129–143, 2003.
- [8] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Proc. ECML-PKDD*, pp. 13–23, 2000.
- [9] Y. Wang, J. Wang, Z. Cao, and A. Barati Farimani, "Molecular contrastive learning of representations via graph neural networks," *Nat. Mach. Intell.*, vol. 4, no. 3, pp. 279–287, 2022.
- [10] J. Vamathevan, D. Clark, P. Czodrowski, I. Dunham, E. Ferran, G. Lee, B. Li, A. Madabhushi, P. Shah, M. Spitzer, et al., "Applications of machine learning in drug discovery and development," *Nat. Rev. Drug Discov.*, vol. 18, no. 6, pp. 463–477, 2019.
- [11] T. Lanciano, F. Bonchi, and A. Gionis, "Explainable classification of brain networks via contrast subgraphs," in *Proc. KDD*, pp. 3308–3318, 2020.
- [12] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, 2018.
- [13] X. Su, S. Xue, F. Liu, J. Wu, J. Yang, C. Zhou, W. Hu, C. Paris, S. Nepal, D. Jin, et al., "A comprehensive survey on community detection with deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–21, 2022.
- [14] F. Liu, S. Xue, J. Wu, C. Zhou, W. Hu, C. Paris, S. Nepal, J. Yang, and P. S. Yu, "Deep learning for community detection: Progress, challenges and opportunities," in *Proc. IJCAI*, pp. 1–7, 2021.
- [15] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu, "A comprehensive survey on graph anomaly detection with deep learning," *IEEE Trans. Knowl. Data Eng.*, pp. 1–32, 2021.
- [16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2020.
- [17] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, pp. 249–270, 2020.
- [18] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, pp. 1–42, 2020.
- [19] C. Liu, Y. Zhan, C. Li, B. Du, J. Wu, W. Hu, T. Liu, and D. Tao, "Graph pooling for graph neural networks: Progress, challenges, and opportunities," *arXiv preprint arXiv:2204.07321*, 2022.
- [20] S. S. Du, K. Hou, R. Salakhutdinov, B. Póczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," in *Proc. NeurIPS*, pp. 5724–5734, 2019.
- [21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. ICML*, pp. 1263–1272, 2017.
- [22] D. Haussler et al., "Convolution kernels on discrete structures," tech. rep., Citeseer, 1999.
- [23] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, no. 9, 2011.
- [24] S. Hido and H. Kashima, "A linear-time graph kernel," in *Proc. ICDM*, pp. 179–188, 2009.
- [25] C. Morris, K. Kersting, and P. Mutzel, "Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs," in *Proc. ICDM*, pp. 327–336, 2017.
- [26] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Mach. Learn.*, vol. 102, no. 2, pp. 209–245, 2016.
- [27] B. Rieck, C. Bock, and K. Borgwardt, "A persistent weisfeiler-lehman procedure for graph classification," in *Proc. ICML*, pp. 5448–5458, 2019.
- [28] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. ICDM*, pp. 1–8, 2005.
- [29] R. W. Floyd, "Algorithm 97: shortest path," *Commun. ACM*, vol. 5, no. 6, pp. 345–350, 1962.
- [30] E. W. Dijkstra et al., "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.
- [31] G. Nikolentzou, P. Meladianos, F. Rousseau, Y. Stavarakas, and M. Vazirgiannis, "Shortest-path graph kernels for document similarity," in *Proc. EMNLP*, pp. 1890–1900, 2017.
- [32] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.

- [33] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, “Extensions of marginalized graph kernels,” in *Proc. ICML*, pp. 70–78, 2004.
- [34] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, “Optimal assignment kernels for attributed molecular graphs,” in *Proc. ICML*, pp. 225–232, 2005.
- [35] F. D. Johansson and D. Dubhashi, “Learning with similarity functions on graphs using matchings of geometric embeddings,” in *Proc. KDD*, pp. 467–476, 2015.
- [36] M. Schiavinato, A. Gasparetto, and A. Torsello, “Transitive assignment kernels for structural classification,” in *Proc. SIMBAD*, pp. 146–159, 2015.
- [37] D. Pachauri, R. Kondor, and V. Singh, “Solving the multi-way matching problem by permutation synchronization,” in *Proc. NeurIPS*, pp. 1860–1868, 2013.
- [38] A. Woźnica, A. Kalousis, and M. Hilario, “Adaptive matching based kernels for labelled graphs,” in *Proc. PAKDD*, pp. 374–385, 2010.
- [39] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, “Efficient graphlet kernels for large graph comparison,” in *Proc. AISTATS*, pp. 488–495, 2009.
- [40] N. Wale, I. A. Watson, and G. Karypis, “Comparison of descriptor spaces for chemical compound retrieval and classification,” *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 347–375, 2008.
- [41] N. Kriege and P. Mutzel, “Subgraph matching kernels for attributed graphs,” in *Proc. ICML*, p. 291–298, 2012.
- [42] F. Costa and K. De Grave, “Fast neighborhood subgraph pairwise distance kernel,” in *Proc. ICML*, pp. 255–262, 2010.
- [43] M. Kuramochi and G. Karypis, “Frequent subgraph discovery,” in *Proc. ICDM*, pp. 313–320, 2001.
- [44] X. Yan and J. Han, “gspan: Graph-based substructure pattern mining,” in *Proc. ICDM*, pp. 721–724, 2002.
- [45] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt, “Near-optimal supervised feature selection among frequent subgraphs,” in *Proc. SDM*, pp. 1076–1087, 2009.
- [46] X. Yan, H. Cheng, J. Han, and P. S. Yu, “Mining significant graph patterns by leap search,” in *Proc. SIGMOD*, pp. 433–444, 2008.
- [47] X. Kong and S. Y. Philip, “Multi-label feature selection for graph classification,” in *Proc. ICDM*, pp. 274–283, 2010.
- [48] J. Wu, Z. Hong, S. Pan, X. Zhu, Z. Cai, and C. Zhang, “Multi-graph-view learning for graph classification,” in *Proc. ICDM*, pp. 590–599, 2014.
- [49] X. Kong and P. S. Yu, “Semi-supervised feature selection for graph classification,” in *Proc. KDD*, pp. 793–802, 2010.
- [50] Y. Zhao, X. Kong, and S. Y. Philip, “Positive and unlabeled learning for graph classification,” in *Proc. ICDM*, pp. 962–971, 2011.
- [51] C. Cai and Y. Wang, “A simple yet effective baseline for non-attribute graph classification,” *arXiv preprint arXiv:1811.03508*, 2018.
- [52] S. Ivanov and E. Burnaev, “Anonymous walk embeddings,” in *Proc. ICML*, pp. 2186–2195, 2018.
- [53] F. R. Chung and F. C. Graham, *Spectral graph theory*, vol. 92. American Mathematical Soc., 1997.
- [54] S. Verma and Z.-L. Zhang, “Hunt for the unique, stable, sparse and fast feature learning on graphs,” in *Proc. NeurIPS*, pp. 87–97, 2017.
- [55] S. Sawlani, L. Zhao, and L. Akoglu, “Fast attributed graph embedding via density of states,” in *Proc. ICDM*, pp. 559–568, 2021.
- [56] A. Tsitsulin, M. Munkhoeva, and B. Perozzi, “Just slaq when you approximate: Accurate spectral distances for web-scale graphs,” in *Proc. WWW*, pp. 2697–2703, 2020.
- [57] S. L. Braunstein, S. Ghosh, and S. Severini, “The laplacian of a graph as a density matrix: a basic combinatorial approach to separability of mixed states,” *Ann. Comb.*, vol. 10, no. 3, pp. 291–317, 2006.
- [58] P.-Y. Chen, L. Wu, S. Liu, and I. Rajapakse, “Fast incremental von neumann graph entropy computation: Theory, algorithm, and applications,” in *Proc. ICML*, pp. 1091–1101, 2019.
- [59] X. Liu, L. Fu, and X. Wang, “Bridging the gap between von neumann graph entropy and structural information: theory and applications,” in *Proc. WWW*, pp. 3699–3710, 2021.
- [60] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *Proc. ICLR*, pp. 1–12, 2013.
- [61] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, “subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs,” in *Proc. KDD Workshop on Mining and Learning with Graphs*, 2016.
- [62] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” in *Proc. KDD Workshop on Mining and Learning with Graphs*, 2017.
- [63] D. Nguyen, W. Luo, T. D. Nguyen, S. Venkatesh, and D. Phung, “Learning graph representation via frequent subgraphs,” in *Proc. SDM*, pp. 306–314, 2018.
- [64] J. B. Lee, R. Rossi, and X. Kong, “Graph classification using structural attention,” in *Proc. KDD*, pp. 1666–1674, 2018.

- [65] X. Zhao, B. Zong, Z. Guan, K. Zhang, and W. Zhao, "Substructure assembling network for graph classification," in *Proc. AAAI*, vol. 32, 2018.
- [66] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *Proc. ICML*, pp. 610–619, 2018.
- [67] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *Proc. ICML*, pp. 5708–5717, 2018.
- [68] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.
- [69] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proc. ICLR*, pp. 1–20, 2016.
- [70] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. ICML*, pp. 2014–2023, 2016.
- [71] G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis, "Kernel graph convolutional neural networks," in *Proc. ICANN*, pp. 22–32, 2018.
- [72] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, 2009.
- [73] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, pp. 3693–3702, 2017.
- [74] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. NeurIPS*, pp. 1993–2001, 2016.
- [75] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proc. ICANN*, pp. 44–51, 2011.
- [76] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," in *Proc. ICML-IJCAL workshop on Computational Biology*, pp. 1–12, 2018.
- [77] Z. Xinyi and L. Chen, "Capsule graph neural network," in *Proc. ICLR*, pp. 1–16, 2018.
- [78] M. D. G. Mallea, P. Meltzer, and P. J. Bentley, "Capsule neural networks for graph classification using explicit tensorial graph representations," *arXiv preprint arXiv:1902.08399*, 2019.
- [79] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Proc. NeurIPS*, pp. 4502–4510, 2016.
- [80] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nat. Commun.*, vol. 8, no. 1, pp. 1–8, 2017.
- [81] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. NeurIPS*, pp. 2224–2232, 2015.
- [82] G. Bouritsas, F. Frasca, S. P. Zafeiriou, and M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–12, 2022.
- [83] L. Zhao, W. Jin, L. Akoglu, and N. Shah, "From stars to subgraphs: Uplifting any GNN with local structure awareness," in *Proc. ICLR*, pp. 1–22, 2022.
- [84] M. Zhang and P. Li, "Nested graph neural networks," in *Proc. NeurIPS*, pp. 1–14, 2021.
- [85] A. Wijesinghe and Q. Wang, "A new perspective on" how graph neural networks go beyond weisfeiler-lehman?," in *Proc. ICLR*, pp. 1–23, 2022.
- [86] Q. Sun, J. Li, H. Peng, J. Wu, Y. Ning, P. S. Yu, and L. He, "Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism," in *Proc. WWW*, pp. 2081–2091, 2021.
- [87] E. Alsentzer, S. G. Finlayson, M. M. Li, and M. Zitnik, "Subgraph neural networks," in *Proc. NeurIPS*, pp. 1–13, 2020.
- [88] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *Proc. ICLR*, pp. 1–46, 2022.
- [89] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proc. CVPR*, pp. 652–660, 2017.
- [90] D. Chen, L. Jacob, and J. Mairal, "Convolutional kernel networks for graph-structured data," in *Proc. ICML*, pp. 1576–1586, 2020.
- [91] Q. Long, Y. Jin, Y. Wu, and G. Song, "Theoretically improving graph neural networks via anonymous walk graph kernels," in *Proc. WWW*, pp. 1204–1214, 2021.
- [92] A. Jacot, C. Hongler, and F. Gabriel, "Neural tangent kernel: Convergence and generalization in neural networks," in *Proc. NeurIPS*, pp. 8580–8589, 2018.
- [93] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola, "Deriving neural architectures from sequence and graph kernels," in *Proc. ICML*, pp. 2024–2033, 2017.
- [94] R. Al-Rfou, B. Perozzi, and D. Zelle, "Ddgg: Learning graph representations for deep divergence graph kernels," in *Proc. WWW*, pp. 37–48, 2019.
- [95] G. Nikolentzos and M. Vazirgiannis, "Random walk graph neural networks," in *Proc. NeurIPS*, pp. 16211–16222, 2020.

- [96] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *Proc. ICML*, pp. 1–14, 2014.
- [97] M. Balcilar, G. Renton, P. Héroux, B. Gaüzère, S. Adam, and P. Honeine, "Analyzing the expressive power of graph neural networks in a spectral perspective," in *Proc. ICLR*, 2020.
- [98] D. Bo, X. Wang, C. Shi, and H. Shen, "Beyond low-frequency information in graph convolutional networks," in *Proc. AAAI*, pp. 3950–3957, 2021.
- [99] M. Balcilar, G. Renton, P. Héroux, B. Gauzere, S. Adam, and P. Honeine, "Bridging the gap between spectral and spatial domains in graph neural networks," *arXiv preprint arXiv:2003.11702*, 2020.
- [100] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *Proc. ICLR*, pp. 1–13, 2019.
- [101] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, 2011.
- [102] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NeurIPS*, pp. 3837–3845, 2016.
- [103] M. Balcilar, P. Héroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine, "Breaking the limits of message passing graph neural networks," in *Proc. ICML*, pp. 599–608, 2021.
- [104] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional arma filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 3496–3507, 2021.
- [105] R. Levie, W. Huang, L. Bucci, M. M. Bronstein, and G. Kutyniok, "Transferability of spectral graph convolutional neural networks," *J. Mach. Learn. Res.*, vol. 22, pp. 272–331, 2021.
- [106] X. Zheng, B. Zhou, J. Gao, Y. Wang, P. Lió, M. Li, and G. Montufar, "How framelets enhance graph neural networks," in *Proc. ICML*, pp. 1–10, 2021.
- [107] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. ICLR*, pp. 1–17, 2018.
- [108] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," in *Proc. NeurIPS*, pp. 1–11, 2020.
- [109] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, "Every document owns its structure: Inductive text classification via graph neural networks," in *Proc. ACL*, pp. 334–339, 2020.
- [110] Z. Wang and S. Ji, "Second-order pooling for graph neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–12, 2020.
- [111] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *Proc. ICLR*, pp. 1–11, 2016.
- [112] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI*, pp. 1–8, 2018.
- [113] L. Zhang, X. Wang, H. Li, G. Zhu, P. Shen, P. Li, X. Lu, S. A. A. Shah, and M. Bennamoun, "Structure-feature based graph self-adaptive pooling," in *Proc. WWW*, pp. 3098–3104, 2020.
- [114] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.
- [115] U. Von Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, no. 4, pp. 395–416, 2007.
- [116] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. ICLR*, pp. 1–14, 2014.
- [117] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, 2009.
- [118] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *Proc. ICLR*, pp. 1–12, 2020.
- [119] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.
- [120] J. Baek, M. Kang, and S. J. Hwang, "Accurate learning of graph representations with graph multiset pooling," in *Proc. ICLR*, pp. 1–22, 2021.
- [121] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proc. ICML*, pp. 874–883, 2020.
- [122] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proc. KDD*, pp. 723–731, 2019.
- [123] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. NeurIPS*, pp. 4805–4815, 2018.
- [124] H. Gao and S. Ji, "Graph u-nets," in *Proc. ICML*, pp. 2083–2092, 2019.
- [125] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," *arXiv preprint arXiv:1811.01287*, 2018.
- [126] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. ICML*, pp. 3734–3743, 2019.
- [127] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proc. AAAI*, pp. 5470–5477, 2020.

- [128] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," in *Proc. AAAI*, vol. 32, 2018.
- [129] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. CVPR*, pp. 5115–5124, 2017.
- [130] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, 2007.
- [131] J. Wu, S. Li, J. Li, Y. Pan, and K. Xu, "A simple yet effective method for graph classification," in *Proc. IJCAI*, pp. 1–7, 2022.
- [132] A. Li and Y. Pan, "Structural information and dynamical complexity of networks," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 3290–3339, 2016.
- [133] J. Wu, X. Chen, K. Xu, and S. Li, "Structural entropy guided graph hierarchical pooling," in *Proc. ICML*, pp. 24017–24030, 2022.
- [134] F. Diehl, "Edge contraction pooling for graph neural networks," *arXiv preprint arXiv:1905.10990*, 2019.
- [135] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. 1, pp. 47–56, 2005.
- [136] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "Tudataset: A collection of benchmark datasets for learning with graphs," *arXiv preprint arXiv:2007.08663*, 2020.
- [137] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *J. Mol. Biol.*, pp. 771–783, 2003.
- [138] G. Subramanian, B. Ramsundar, V. Pande, and R. A. Denny, "Computational modeling of β -secretase 1 (bace-1) inhibitors using ligand based approaches," *J. Chem. Inf. Model.*, vol. 56, no. 10, pp. 1936–1949, 2016.
- [139] S. G. Rohrer and K. Baumann, "Maximum unbiased validation (muv) data sets for virtual screening based on pubchem bioactivity data," *J. Chem. Inf. Model.*, vol. 49, no. 2, pp. 169–184, 2009.
- [140] M. Zitnik, R. Sosič, M. W. Feldman, and J. Leskovec, "Evolution of resilience in protein interactomes across the tree of life," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 116, no. 10, pp. 4426–4433, 2019.
- [141] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. NeurIPS*, pp. 22118–22133, 2020.
- [142] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande, "Low data drug discovery with one-shot learning," *ACS Cent. Sci.*, vol. 3, no. 4, pp. 283–293, 2017.
- [143] K. M. Gayvert, N. S. Madhukar, and O. Elemento, "A data-driven approach to predicting successes and failures of clinical trials," *Cell Chem. Biol.*, vol. 23, no. 10, pp. 1294–1301, 2016.
- [144] I. F. Martins, A. L. Teixeira, L. Pinheiro, and A. O. Falcao, "A bayesian approach to in silico blood-brain barrier penetration modeling," *J. Chem. Inf. Model.*, vol. 52, no. 6, pp. 1686–1697, 2012.
- [145] T. Challenge, "Tox21 data challenge 2014," <https://tripod.nih.gov/tox/challenge>, 2014.
- [146] A. M. Richard, R. S. Judson, K. A. Houck, C. M. Grulke, P. Volarath, I. Thillainadarajah, C. Yang, J. Rathman, M. T. Martin, J. F. Wambaugh, et al., "Toxcast chemical landscape: paving the road to 21st century toxicology," *Chem. Res. Toxicol.*, vol. 29, no. 8, pp. 1225–1251, 2016.
- [147] D. L. Mobley and J. P. Guthrie, "Freesolv: a database of experimental and calculated hydration free energies, with input files," *J. Comput.-Aided Mol. Des.*, vol. 28, no. 7, pp. 711–720, 2014.
- [148] J. S. Delaney, "Esol: estimating aqueous solubility directly from molecular structure," *J. Chem. Inf. Comput.*, vol. 44, no. 3, pp. 1000–1005, 2004.
- [149] M. Wenlock and N. Tomkinson, "Experimental in vitro dmpk and physicochemical data on a set of publicly disclosed compounds," 2015.
- [150] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.
- [151] M. C. Sorkun, A. Khetan, and S. Er, "Aqsolddb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds," *Sci. Data*, vol. 6, no. 1, pp. 1–8, 2019.
- [152] J. J. Irwin, T. Sterling, M. M. Mysinger, E. S. Bolstad, and R. G. Coleman, "Zinc: a free tool to discover chemistry for biology," *J. Chem. Inf. Model.*, vol. 52, no. 7, pp. 1757–1768, 2012.
- [153] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proc. KDD*, p. 1365–1374, 2015.
- [154] A. Krizhevsky, G. Hinton, et al., "Learning multiple layers of features from tiny images," 2009.
- [155] Y. LeCun, "The mnist database of handwritten digits," <http://yann.lecun.com/exdb/mnist/>, 1998.
- [156] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.
- [157] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," in *Proc. NeurIPS Track Datasets and Benchmarks*, pp. 1–13, 2021.

- [158] S. Ivanov, S. Sviridov, and E. Burnaev, "Understanding isomorphism bias in graph data sets," *arXiv preprint arXiv:1910.12091*, 2019.
- [159] D. Ruffinelli, S. Broscheit, and R. Gemulla, "You can teach an old dog new tricks! on training knowledge graph embeddings," in *Proc. ICLR*, pp. 1–20, 2019.
- [160] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proc. ICLR*, pp. 1–16, 2020.
- [161] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, "An empirical study of graph contrastive learning," in *Proc. NeurIPS Track Datasets and Benchmarks*, pp. 1–25, 2021.
- [162] C. Shi*, M. Xu*, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "Graphaf: a flow-based autoregressive model for molecular graph generation," in *Proc. ICLR*, pp. 1–18, 2020.
- [163] H. L. Morgan, "The generation of a unique machine description for chemical structures—a technique developed at chemical abstracts service," *J. Chem. Doc.*, vol. 5, no. 2, pp. 107–113, 1965.
- [164] L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, *et al.*, "Open catalyst 2020 (oc20) dataset and community challenges," *ACS Catal.*, vol. 11, no. 10, pp. 6059–6072, 2021.
- [165] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *Proc. ICML*, pp. 8459–8468, 2020.
- [166] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. CVPR*, pp. 5410–5419, 2017.
- [167] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. NeurIPS*, pp. 1–9, 2015.
- [168] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proc. ICLR*, pp. 1–17, 2018.
- [169] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, and K. Wang, "Hoppity: Learning graph transformations to detect and fix bugs in programs," in *Proc. ICLR*, pp. 1–17, 2020.
- [170] X. Fang, L. Liu, J. Lei, D. He, S. Zhang, J. Zhou, F. Wang, H. Wu, and H. Wang, "Geometry-enhanced molecular representation learning for property prediction," *Nat. Mach. Intell.*, vol. 4, no. 2, pp. 127–134, 2022.
- [171] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem," in *Proc. ACL-IJCNLP*, pp. 1702–1712, 2015.
- [172] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-cnn," in *Proc. WWW*, pp. 1063–1072, 2018.
- [173] Z.-M. Chen, X.-S. Wei, P. Wang, and Y. Guo, "Multi-label image recognition with graph convolutional networks," in *Proc. CVPR*, pp. 5177–5186, 2019.
- [174] J. Wu, S. Pan, X. Zhu, Z. Cai, and C. Zhang, "Multi-graph-view learning for complicated object classification," in *Proc. IJCAI*, p. 3953–3959, 2015.
- [175] J. Hao, J. Liu, E. Pereira, R. Liu, J. Zhang, Y. Zhang, K. Yan, Y. Gong, J. Zheng, J. Zhang, *et al.*, "Uncertainty-guided graph attention network for parapneumonic effusion diagnosis," *Med. Image Anal.*, vol. 75, pp. 102217–102229, 2022.
- [176] J. Wu, H. Jiang, X. Ding, A. Konda, J. Han, Y. Zhang, and Q. Li, "Learning differential diagnosis of skin conditions with co-occurrence supervision using graph convolutional networks," in *Proc. MICCAI*, pp. 335–344, Springer, 2020.
- [177] J. Wu, X. Zhu, C. Zhang, and S. Y. Philip, "Bag constrained structure pattern mining for multi-graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 10, pp. 2382–2396, 2014.
- [178] Y. Dou, K. Shu, C. Xia, P. S. Yu, and L. Sun, "User preference-aware fake news detection," in *Proc. SIGIR*, pp. 2051–2055, 2021.
- [179] A. Silva, L. Luo, S. Karunasekera, and C. Leckie, "Embracing domain differences in fake news: Cross-domain fake news detection using multi-modal data," in *Proc. AAAI*, vol. 35, pp. 557–565, 2021.
- [180] D. Wang, M. Jammik, and P. Lio, "Abstract diagrammatic reasoning with multiplex graph networks," in *Proc. ICLR*, pp. 1–20, 2020.
- [181] A. Haghighi, A. Y. Ng, and C. D. Manning, "Robust textual inference via graph matching," in *Proc. HLT-EMNLP*, pp. 387–394, 2005.
- [182] C. Liu, Z. Mao, T. Zhang, H. Xie, B. Wang, and Y. Zhang, "Graph structured network for image-text matching," in *Proc. CVPR*, pp. 10921–10930, 2020.
- [183] D. Ramage, A. N. Rafferty, and C. D. Manning, "Random walks for text semantic similarity," in *Proc. ACL-IJCNLP workshop on graph-based methods for natural language processing (TextGraphs-4)*, pp. 23–31, 2009.
- [184] T. Hughes and D. Ramage, "Lexical semantic relatedness with random graph walks," in *Proc. EMNLP-CoNLL*, pp. 581–589, 2007.
- [185] K. Xu, L. Wang, M. Yu, Y. Feng, Y. Song, Z. Wang, and D. Yu, "Cross-lingual knowledge graph alignment via graph matching neural network," in *Proc. ACL*, pp. 3156–3161, 2019.

- [186] C. Monz and B. J. Dorr, "Iterative translation disambiguation for cross-language information retrieval," in *Proc. SIGIR*, pp. 520–527, 2005.
- [187] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, and T. Hou, "Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models," *J. Cheminformatics*, vol. 13, no. 1, pp. 1–23, 2021.
- [188] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.
- [189] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, "Graph kernels for chemical informatics," *Neural Netw.*, vol. 18, no. 8, pp. 1093–1110, 2005.
- [190] M. Shao, J. Li, F. Chen, H. Huang, S. Zhang, and X. Chen, "An efficient approach to event detection and forecasting in dynamic multivariate social media networks," in *Proc. WWW*, pp. 1631–1639, 2017.
- [191] J. Wu, Z. Hong, S. Pan, X. Zhu, Z. Cai, and C. Zhang, "Multi-graph-view subgraph mining for graph classification," *Knowl. Inf. Syst.*, vol. 48, no. 1, pp. 29–54, 2016.
- [192] J. Wu, S. Pan, X. Zhu, C. Zhang, and S. Y. Philip, "Multiple structure-view learning for graph classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 7, pp. 3236–3251, 2017.
- [193] S. Pan, J. Wu, X. Zhu, C. Zhang, and S. Y. Philip, "Joint structure feature exploration and regularization for multi-task graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 3, pp. 715–728, 2015.
- [194] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Task sensitive feature exploration and learning for multitask graph classification," *IEEE Trans. Cybern.*, vol. 47, no. 3, pp. 744–758, 2016.
- [195] J. Wu, Z. Hong, S. Pan, X. Zhu, C. Zhang, and Z. Cai, "Multi-graph learning with positive and unlabeled bags," in *Proc. SDM*, pp. 217–225, 2014.
- [196] J. Wu, S. Pan, X. Zhu, C. Zhang, and X. Wu, "Positive and unlabeled multi-graph learning," *IEEE Trans. Cybern.*, vol. 47, no. 4, pp. 818–829, 2016.
- [197] S. Pan, J. Wu, X. Zhu, and C. Zhang, "Graph ensemble boosting for imbalanced noisy graph stream classification," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 954–968, 2014.
- [198] C. C. Aggarwal, "On classification of graph streams," in *Proc. SDM*, pp. 652–663, 2011.
- [199] Z. Yang, G. Zhang, J. Wu, J. Yang, Q. Z. Sheng, H. Peng, A. Li, S. Xue, and J. Su, "Minimum entropy principle guided graph neural networks," in *Proc. WSDM*, pp. 114–122, 2023.
- [200] B. Knyazev, M. Drozdal, G. W. Taylor, and A. Romero Soriano, "Parameter prediction for unseen deep architectures," in *Proc. NeurIPS*, pp. 1–16, 2021.
- [201] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.
- [202] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *Proc. ICML*, pp. 9323–9332, 2021.
- [203] J. Gasteiger, F. Becker, and S. Günnemann, "Gemnet: Universal directional graph neural networks for molecules," in *Proc. NeurIPS*, pp. 6790–6802, 2021.
- [204] J. Han, Y. Rong, T. Xu, and W. Huang, "Geometrically equivariant graph neural networks: A survey," *arXiv preprint arXiv:2202.07230*, 2022.
- [205] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," in *Proc. NeurIPS*, pp. 19620–19631, 2020.
- [206] S. Miao, M. Liu, and P. Li, "Interpretable and generalizable graph learning via stochastic attention mechanism," in *Proc. ICML*, pp. 1–20, 2022.
- [207] Y.-X. Wu, X. Wang, A. Zhang, X. He, and T.-S. Chua, "Discovering invariant rationales for graph neural networks," in *Proc. ICLR*, pp. 1–22, 2022.
- [208] D. P. P. Mesquita, A. H. S. Jr., and S. Kaski, "Rethinking pooling in graph neural networks," in *Proc. NeurIPS*, pp. 2220–2231, 2020.
- [209] H. Xie, J. Ma, L. Xiong, and C. Yang, "Federated graph classification over non-IID graphs," in *Proc. NeurIPS*, pp. 18839–18852, 2021.
- [210] C. He, K. Balasubramanian, E. Ceyani, C. Yang, H. Xie, L. Sun, L. He, L. Yang, S. Y. Philip, Y. Rong, *et al.*, "Fedgraphnn: A federated learning benchmark system for graph neural networks," *arXiv preprint arXiv:2104.07145*, 2021.
- [211] G. Zhang, Z. Yang, J. Wu, J. Yang, X. Shan, H. Peng, J. Su, C. Zhou, Q. Z. Sheng, L. Akoglu, and C. C. Aggarwal, "Dual-discriminative graph neural network for imbalanced graph-level anomaly detection," in *Proc. NeurIPS*, pp. 1–12, 2022.
- [212] P. Agarwal, R. Vaitithyanathan, S. Sharma, and G. Shroff, "Catching the long-tail: Extracting local news events from twitter," in *Proc. AAAI*, pp. 379–382, 2012.
- [213] Y. Wang, Y. Zhao, N. Shah, and T. Derr, "Imbalanced graph classification via graph-of-graph neural networks," *arXiv preprint arXiv:2112.00238*, 2021.

- [214] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary? revisiting heterogeneous graph embeddings," in *Proc. CIKM*, pp. 437–446, 2018.
- [215] Y. Yang, Z. Guan, J. Li, W. Zhao, J. Cui, and Q. Wang, "Interpretable and efficient heterogeneous graph convolutional network," *IEEE Trans. Knowl. Data Eng.*, 2021.
- [216] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks," in *Proc. KDD*, pp. 1150–1160, 2021.
- [217] H. Wang, D. Lian, Y. Zhang, L. Qin, and X. Lin, "Gognn: Graph of graphs neural network for predicting structured entity interactions," in *Proc. IJCAI*, pp. 1–7, 2021.
- [218] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *Bioinformatics*, vol. 34, no. 13, p. 457–466, 2018.
- [219] M. Kuhn, C. von Mering, M. Campillos, L. J. Jensen, and P. Bork, "Stitch: interaction networks of chemicals and proteins," *Nucleic Acids Res.*, vol. 36, no. 1, pp. 684–688, 2007.
- [220] R. Ma, G. Pang, L. Chen, and A. van den Hengel, "Deep graph-level anomaly detection by glocal knowledge distillation," in *Proc. WSDM*, pp. 704–714, 2022.
- [221] C. Qiu, M. Kloft, S. Mandt, and M. Rudolph, "Raising the bar in graph-level anomaly detection," in *Proc. IJCAI*, pp. 1–8, 2022.
- [222] L. Zhao and L. Akoglu, "On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights," *Big Data*, pp. 1–30, 2021.
- [223] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proc. ICML*, pp. 4393–4402, 2018.
- [224] J. Tang, J. Li, Z. Gao, and J. Li, "Rethinking graph neural networks for anomaly detection," in *Proc. ICML*, pp. 21076–21089, 2022.
- [225] S. Gui, X. Li, L. Wang, and S. Ji, "Good: A graph out-of-distribution benchmark," *arXiv preprint arXiv:2206.08452*, 2022.
- [226] B. Bevilacqua, Y. Zhou, and B. Ribeiro, "Size-invariant graph representations for graph classification extrapolations," in *Proc. ICML*, pp. 1–14, 2021.
- [227] O. Sporns, "Graph theory methods: applications in brain networks," *Dialogues Clin. Neurosci.*, 2022.
- [228] J. Wu, S. Pan, X. Zhu, and Z. Cai, "Boosting for multi-graph classification," *IEEE Trans. Cybern.*, vol. 45, no. 3, pp. 416–429, 2014.
- [229] J. Wu, S. Pan, X. Zhu, C. Zhang, and X. Wu, "Multi-instance learning with discriminative bag mapping," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 6, pp. 1065–1080, 2018.

A TAXONOMY OF GRAPH-LEVEL LEARNING TECHNIQUES

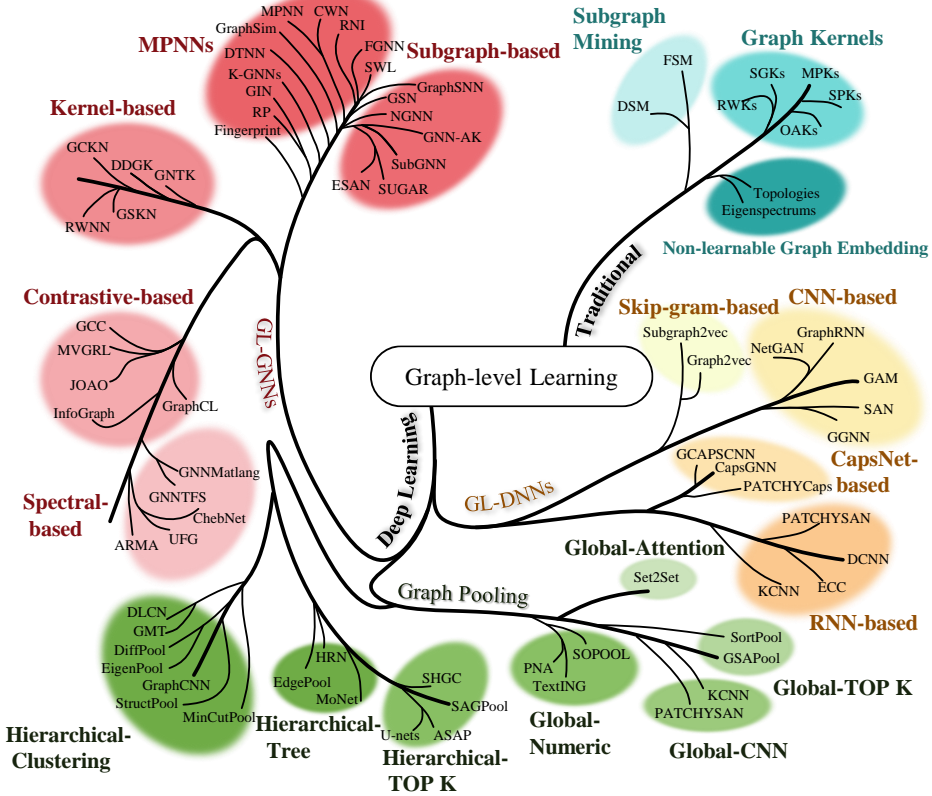


Fig. 8. The taxonomy tree of graph-level learning techniques.

The taxonomy tree in Fig. 8 depicts these four branches of graph-level learning with selected algorithms highlighted.

B TRADITIONAL LEARNING

All traditional graph-level learning publications discussed in this section are summarized in Table 2.

C GRAPH-LEVEL DEEP NEURAL NETWORKS (GL-DNNs)

The representative GL-DNNs mentioned in this section are summarized in Table 3.

C.1 CNN-based GL-DNNs

There are two main branches of CNN-based graph-level learning. The first branch is illustrated in Fig. 9 (A), which sorts nodes and arranges the node features to form a concentration matrix as the grid-structured data for training the CNNs. As a second branch, researchers have developed a CNN-guided neural network version of an MPK, which is shown in Fig. 9 (B).

Table 2. Summary of Traditional Graph-level Learning Methods.

Subsection	Model	Year	Method	Venue	Language	Code Repository
Graph Kernels	Message Passing Kernels	2009	NHK[1]	ICDM	Python	https://github.com/ysig/GraKeL
		2011	WL[2]	JMLR	C++	https://github.com/BorgwardtLab/graph-kernels
		2016	PK[3]	ML	MATLAB	https://github.com/marionmari/propagation_kernels
	ShortestPath Kernels	2017	Global-WL[4]	ICDM	C++	https://github.com/chrmrrs/glocalwl
		2019	P-WL[5]	ICML	Python	https://github.com/BorgwardtLab/P-WL
	Random Walk Kernels	2005	SPK[6]	ICDM	Python	https://github.com/ysig/GraKeL
		2017	SPK-DS[7]	EMNLP	-	-
		2003	RWK[8]	LNAI	Python	https://github.com/jajupmochi/graphkit-learn
		2004	ERWK[9]	ICML	Python	https://github.com/jajupmochi/graphkit-learn
		2010	SOMRWK[10]	JMLR	Python	https://github.com/ysig/GraKeL
		2005	OAK[11]	ICML	-	-
		Optimal Assignment Kernels	2013	PS-OAK[12]	NeurIPS	Python
	2015		GE-OAK[13]	KDD	-	-
	2015		TAK[14]	SIMBAD	-	-
	Subgraph Kernels	2009	Graphlet[15]	AISTATS	Python	https://github.com/ysig/GraKeL
		2010	NSPDK[16]	ICML	Python	https://github.com/fabriziocosta/EDeN
		2012	SMK[17]	ICML	C++	https://github.com/fapaul/GraphKernelBenchmark
	Subgraph Mining	Frequent Subgraph Mining	2000	AGM[18]	ECML PKDD	C++
2001			FSG[19]	ICDM	C++	https://github.com/NikhilGupta1997/Data-Mining-Algorithms
Discrimina- tive Subgraph Mining		2002	gSpan[20]	ICDM	Python	https://github.com/betterenvi/gSpan
		2008	LEAP[21]	SIGMOD	-	-
		2009	CORK[22]	SDM	-	-
		2010	gMLC[23]	ICDM	-	-
		2010	gSSC[24]	KDD	-	-
		2011	gPU[25]	ICDM	-	-
Non-Learnable Graph Embedding	2014	gCGVFL[26]	ICDM	-	-	
	2017	FGSD[27]	NeurIPS	Python	https://github.com/vermaMachineLearning/FGSD	
	2018	AWE[28]	ICML	Python	https://github.com/nd7141/AWE	
	2019	LDP[29]	ICLR RLGM	Python	https://github.com/Chen-Cai-OSU/LDP	
	2020	SLAQ[30]	WWW	Python	https://github.com/google-research/google-research/tree/master/graph_embedding/slaq	
	2021	VNGE[31]	WWW	Python	https://github.com/xuecheng27/WWW21-Structural-Information	
2021	A-DOGE[32]	ICDM	Python	https://github.com/sawlani/A-DOGE		

Table 3. Summary of Graph-Level Deep Neural Networks (GL-DNNs).

Model	Year	Method	Venue	Language	Code Repository
Skipgram -Based	2016	Subgraph2vec[33]	KDD MLG	Python	https://github.com/MLDroid/subgraph2vec_tf
	2017	Graph2vec[34]	KDD MLG	Python	https://github.com/MLDroid/graph2vec_tf
	2018	GE-FSG[35]	SDM	Python	https://github.com/nphdang/GE-FSG
RNN- Based	2016	GGNN[36]	ICLR	Python-Tensorflow	https://github.com/Microsoft/gated-graph-neural-network-samples
	2018	GAM[37]	KDD	Python-Pytorch	https://github.com/benedekrozemberczki/GAM
	2018	SAN[38]	AAAI	-	-
	2018	NetGAN[39]	ICML	Python-Tensorflow	https://github.com/danielzuegner/netgan
CNN- Based	2018	GraphRNN[40]	ICML	Python-Pytorch	https://github.com/snap-stanford/GraphRNN
	2016	PATCHYSAN[41]	ICML	Python	https://github.com/tvayer/PSCN
	2016	DCNN[42]	NeurIPS	Python	https://github.com/jcatw/dcnm
	2017	ECC[43]	CVPR	Python-Pytorch	https://github.com/mys007/ecc
CapsNet- Based	2018	KCNN[44]	ICANN	Python-Pytorch	https://github.com/giannisnik/cnn-graph-classification
	2018	GCAPSCNN[45]	WCB	Python	https://github.com/vermaMachineLearning/Graph-Capsule-CNN-Networks
	2019	CapsGNN[46]	ICLR	Python-Pytorch	https://github.com/benedekrozemberczki/CapsGNN
2019	PatchyCaps[47]	Arxiv	Python	https://github.com/BraintreeLtd/PatchyCapsules	

D GRAPH-LEVEL GRAPH NEURAL NETWORKS (GL-GNNS)

We also illustrate the contrastive learning-based approaches (see Section D.1). In addition, we investigate the expressivity (see Section D.2), generalizability (see Section D.3), and explainability (see Section D.4) of GL-GNNs. Please refer to Table 4 for the GL-GNNs discussed in this section.

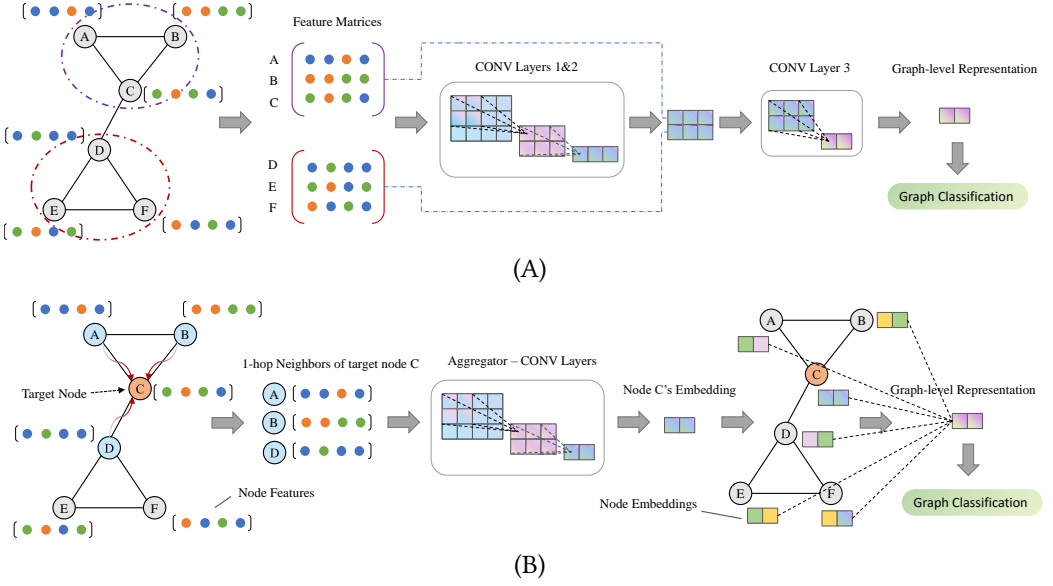


Fig. 9. This figure shows two ways of tentative exploration by CNN-based GL-DNNs on graph-structured data.

D.1 Contrastive Learning-Based GL-GNNs

Contrastive learning [82] is a data augmentation method that creates new, plausible instances by transposing existing data without affecting the semantics. Investigating contrastive learning for GL-GNNs is significant since GL-GNNs are data-driven models that will always encounter bottlenecks given insufficiently labeled graphs.

Graph Contrastive Learning (GraphCL) [72] defines four approaches to creating new instances as augmentation data: (1) node dropping, which randomly removes a proportion of nodes from the graph; (2) edge perturbation, which randomly adds or removes a certain percentage of edges from the graph; (3) feature masking, where some of the features of some nodes are randomly masked; and (4) subgraphs, where subgraphs are taken from the graph. To be noticed, the newly-produced instances must be labeled as the same class as the source graph. InfoGraph [73], for example, samples subgraphs g_m from a source graph \mathcal{G} as new instances. A GL-GNN encoder \mathcal{H}^ϕ with some parameters ϕ is then used to generate graph-level representations of g_m and \mathcal{G} , denoted as $\mathbf{h}_{g_m}^\phi$ and $\mathbf{h}_{\mathcal{G}}^\phi$. InfoGraph's learning objective is to maximize the mutual information between $\mathbf{h}_{\mathcal{G}}^\phi$ and all $\mathbf{h}_{g_m}^\phi$, $g_m \in \mathcal{G}$. This can be brought of as an evaluation of the statistical dependencies between two variables. Formally:

$$\mathcal{H}^\phi, \mathcal{H}^\psi = \operatorname{argmax}_{\phi, \psi} \sum_{\mathcal{G} \in \mathbb{G}} \frac{1}{|\{g_m\}|} \sum_{g_m \in \mathcal{G}} I_{\phi, \psi}(\mathbf{h}_{g_m}^\phi; \mathbf{h}_{\mathcal{G}}^\phi), \quad (32)$$

where \mathcal{H}^ψ is the mutual information estimator with the parameters ψ , and $I_{\phi, \psi}(\cdot, \cdot)$ measures the mutual information.

Similarly, Graph Contrastive Coding (GCC) [74] samples subgraphs g_1, \dots, g_M from the graph dataset $\mathbb{G} = \{\mathcal{G}_1, \dots, \mathcal{G}_N\}$ as new instances. The embeddings of the subgraphs g_m and graph \mathcal{G}_n produced by the GL-GNN are denoted as \mathbf{h}_{g_m} and $\mathbf{h}_{\mathcal{G}_n}$, respectively. In the GL-GNN, a GCC employs

Table 4. Summary of Graph-Level Graph Neural Networks (GL-GNNs).

Model	Year	Method	Venue	Language	Code Repository
Message Passing Neural Networks	2015	Fingerprint[48]	NeurIPS	Python-Tensorflow	https://github.com/HIPS/neural-fingerprint
	2016	GraphSim[49]	NeurIPS	Python-Tensorflow	https://github.com/clvrai/Relation-Network-Tensorflow
	2017	MPNN[50]	ICML	Python-Pytorch	https://github.com/priba/nmp_qc
	2017	DTNN[51]	NC	Python-Tensorflow	https://github.com/atomistic-machine-learning/dttnn
	2019	GIN[52]	ICLR	Python-Pytorch	https://github.com/weihua916/powerful-gnns
	2019	K-GNNs[53]	AAAI	Python-Pytorch	https://github.com/chsmrrs/k-gnn
	2019	PPGN[54]	NeurIPS	Python-Tensorflow	https://github.com/hadarser/ProvablyPowerfulGraphNetworks
	2019	RP[55]	ICML	Python-Pytorch	https://github.com/PurdueMINDS/RelationalPooling
	2021	FGNN[56]	ICLR	Python-Pytorch	https://github.com/mlelarge/graph_neural_net
	2021	SWL[57]	ICML	Python-Pytorch	https://github.com/twitter-research/cwn
Subgraph-Based	2021	CWN[58]	NeurIPS	Python-Pytorch	https://github.com/twitter-research/cwn
	2021	RNI[59]	IJCAL	-	-
	2020	SubGNN[60]	NeurIPS	Python-Pytorch	https://github.com/mims-harvard/SubGNN
	2021	SUGAR[61]	WWW	Python-Tensorflow	https://github.com/RingBDSStack/SUGAR
	2021	NGNN[62]	NeurIPS	Python-Pytorch	https://github.com/muhanzhang/nestdgnn
	2022	GNN-AK[63]	ICLR	Python-Pytorch	https://github.com/LingxiaoShawn/GNNAsKernel
	2022	GraphSNN[64]	ICLR	Python-Pytorch	https://github.com/wokas36/GraphSNN
Kernel-Based	2022	ESAN[65]	ICLR	Python-Pytorch	https://github.com/beabevi/esan
	2022	GSN[66]	TPAMI	Python-Pytorch	https://github.com/gbouritsas/GSN
	2019	GNTR[67]	NeurIPS	Python	https://github.com/KangchengHou/gntk
	2019	DDGK[68]	WWW	Python-Tensorflow	https://github.com/google-research/google-research/tree/master/graph_embedding/ddgk
	2020	GCKN[69]	ICML	Python-Pytorch	https://github.com/claying/GCKN
Contrastive-Based	2020	RWNN[70]	NeurIPS	Python-Pytorch	https://github.com/giannisnik/rwgnn
	2021	GSKN[71]	WWW	Python-Pytorch	https://github.com/YimiAChack/GSKN
	2020	GraphCL[72]	NeurIPS	Python-Pytorch	https://github.com/Shen-Lab/GraphCL
	2020	InfoGraph[73]	ICLR	Python-Pytorch	https://github.com/fanyun-sun/InfoGraph
	2020	GCC[74]	KDD	Python-Pytorch	https://github.com/THUDM/GCC
	2020	MVGR[75]	ICML	Python-Pytorch	https://github.com/kavehassani/mvgrl
Spectral-Based	2021	JOAO[76]	ICML	Python-Pytorch	https://github.com/Shen-Lab/GraphCL_Automated
	2016	ChebNet[77]	NeurIPS	Python-Tensorflow	https://github.com/mdeff/cnn_graph
	2021	GNN-TFS[78]	JMLR	-	-
	2021	GNNMatlang[79]	ICML	Python-Tensorflow	https://github.com/balcilar/gnn-matlang
	2021	ARMA[80]	TPAMI	Python-Pytorch	https://github.com/dmlc/dgl/tree/master/examples/pytorch/arma
2021	UFG[81]	ICML	Python-Pytorch	https://github.com/YuGuangWang/UFG	

InfoNCE loss [83] as the learning objective, that is:

$$\mathcal{L} = \sum_{\mathcal{G}_n \in \mathcal{G}} -\log \frac{\sum_{g_m \in \mathcal{G}_n} \exp(\mathbf{h}_{\mathcal{G}_n}^\top \mathbf{h}_{g_m} / \tau)}{\sum_{i=0}^M \exp(\mathbf{h}_{\mathcal{G}_n}^\top \mathbf{h}_{g_i} / \tau)}, \quad (33)$$

where τ is the temperature hyper-parameter. If $g_m \in \mathcal{G}_n$, the InfoNCE aims to maximize the similarity between \mathbf{h}_{g_m} and $\mathbf{h}_{\mathcal{G}_n}$. Otherwise, it separates \mathbf{h}_{g_m} and $\mathbf{h}_{\mathcal{G}_n}$ as far away in the semantic space as possible. Hassani *et al.* [75] extended graph contrastive learning to the multi-view scenario. Here, each graph view is regarded as an independent instance. This work maximizes the mutual information between a graph view and other views of the same graph. Recently, a contrastive-based GL-GNN MolCLR [84] adopts three augmentation strategies (i.e., node drops, edge drops, and subgraph removal) for achieving benchmark results on 10 million unique molecules.

There are several ways to generate new instances for graph contrastive learning, which raises the question of how to choose the most suitable method for the dataset one is working with. Joint augmentation optimization (JOAO) [76] was developed to address this challenge by automating the search for a proper graph data augmentation method. JOAO trains a probability matrix that can be iteratively updated to select the optimal data augmentation approach. Its performance is competitive.

D.2 The expressivity of GL-GNNs

As the cutting-edge technology for graph-level learning, people want to explore the power of GL-GNNs for distinguishing graphs —namely, they want to investigate the expressivity of GL-GNNs. Practitioners generally employ a representative MPK of the 1-WL algorithm [2, 85] to evaluate the expressivity of standard GL-GNNs (i.e., MPNNs) since MPNNs are the neural network versions of MPKs. The intimate connection between GL-GNNs and 1-WL is exploited in the Graph Isomorphism Network (GIN) [52]. This framework shows the upper expressivity bound of an MPNN equals the 1-WL algorithm. Several research teams have subsequently proved that MPNNs equivalent to 1-WL can not distinguish some substructures in graphs (e.g., cycles, triangles, and Circulant Skip Links) [55, 86, 87]. However, these indistinguishable substructures play a significant role in learning social network and chemical compounds graphs [88]. To break the 1-WL expressivity limitation, the expressivity of GL-GNNs has been empowered through K -WL, convolution enhancement, and feature enrichment.

D.2.1 K -WL. A complex variant of 1-WL is the K -WL algorithm, which identifies more substructures in graphs by relabeling a set of K vertices. Morris *et al.* [53] employed MPNNs dealing with K -dimensional tensors to apply K -WL by neural networks, that is K -GNN. K -GNN achieved the expressivity approximately near but slightly weaker than the K -WL, but its computational cost increases exponentially with K since it needs to calculate K -ranked tensors. To avoid processing high-dimensional tensors, Provably Powerful Graph Networks (PPGN) [54] adopts a variant of the 2-WL algorithm (i.e., 2-FWL [89]) for designing GL-GNNs and achieves the expressivity over 3-WL. Further, PPGN replaces the relabel function in 2-FWL with a matrix multiplication based on a single quadratic operation. Similarly, Folklore Graph Neural Networks (FGNN) [56] implements 2-FWL through matrix operations on tensors, pursuing the expressive power as 3-WL. Despite these common efforts on K -WL equivalence GL-GNNs, the majority of them theoretically exceed 1-WL but do not empirically exceed 1-WL [90]. This weak performance by K -WL equivalent GL-GNNs is due to two main reasons which are explained next.

D.2.2 Convolution Enhancement. One reason for the failure of the K -WL approaches is that they break the local updates of MPNNs [91], i.e., they no longer update vertices based on neighborhood information. In practice, GL-GNNs require local updates to preserve the inductive bias property of the graph convolutions [91]. Therefore, some researchers have explored more powerful GL-GNNs by upscaling the graph convolutions yet preserving the local updates. Alon and Yahav [92] noticed that the majority of GL-GNNs do not capture the long-range interactions between nodes because the number of convolutional layers is limited by over-smoothing issues —that is, the node embeddings tend to be similar after multiple aggregations. However, long-range interactions can influence the discriminativeness of graphs. For example, methylnonane is identified by the atoms posited in the compound's two end sides. To address this issue, these researchers appended a fully linked adjacency matrix to the convolutional layer which aggregates the long-range information without violating any local updates. Another powerful tool for enhancing the convolution layer is the matrix query language (MATLANG) [93, 94]. MATLANG strengthens a GL-GNN so that it can recognize more special substructures through its matrix operations, thereby reaching 3-WL expressivity. Inspired by this work, Balcilar *et al.* [79] added MATLANG to the convolutional layer, while Grees and Reutter [95] evaluated the expressiveness of GL-GNNs through MATLANG instead of the 1-WL algorithm.

D.2.3 Feature Enrichment. Another reason that K -WL methods outperform 1-WL in theory but do not achieve superior performance in experiments is that they ignore the role of node features. As complementary information for graph structures, node features allow almost all graphs to

be discriminated by 1-WL GL-GNNs. Some practitioners have emphasized that considering node features can also improve the expressiveness of GL-GNNs, rather than just focusing on graph structures. Murphy *et al.* [55] annotated a unique position descriptor for each node, that is, sorting all nodes. Adopting these position descriptors as node features can help a 1-WL GL-GNN to better handle featureless graphs and identify more structures. To maintain permutation-invariant of graph-level learning, all permutations of node order should be enumerated and the average results should be taken. Similarly, Colored Local Iterative Procedure (CLIP) [96] sorts the nodes in a substructure and gives them a local position descriptor for feature enrichment. In addition, both Sato *et al.* [97] and Abboud *et al.* [59] insert random features into nodes giving rise to stable and powerful GL-GNNs.

D.2.4 High-order Neural Networks. Recently, researchers have tried to improve the expressivity of GL-GNNs through algebraic topology. This is because equipping graphs as a geometric structure can preserve more valuable properties. Cellular GL-GNNs [57, 58] perform MPNNs on cell complexes, an object including hierarchical structures (e.g., vertices, edges, triangles, tetrahedra). By replacing graphs with cell complexes, cellular GL-GNNs benefits from the better computational fabric for larger expressivity. Furthermore, sheaf neural networks [98, 99] decorate a graph with a geometrical structure, sheaf, which constructs vector space for each node and edge and applies linear transformations among these spaces. A correct sheaf setting will allow an MPNN to pass messages along a richer structure. Thus, linearly separate embeddings can be learned, which will enhance the expressive power of GL-GNNs.

D.3 The generalizability of GL-GNNs

Real-world applications with graph data tend to involve complex scenarios, such as needing to train a model with only a small amount of labeled data that can ultimately perform well with a large-scale unlabeled test (i.e., size shift) or using only a few labeled training graphs to fit the bulk of unlabeled test graphs. The ability to generalize GL-GNNs is hence a crucial aspect of dealing with these challenges.

D.3.1 Size Generalization. Sinha *et al.* [100] stress the importance of generalizing GL-GNNs and presented evaluation criteria for this. Xu *et al.* [101] theoretically explain that GL-GNNs have better size generalization capabilities than MLPs and can extrapolate trained models to test data that is different from the training set. To this end, they presented a trick for MPNNs where the graph's vertices are updated by minimizing the aggregated information instead of through summation. This trick improves generalization ability by altering the learning process from one that is non-linear to one that is linear. Yehudai *et al.* [102] theoretically and empirically found the generalization ability of GL-GNNs as the discrepancies in substructures between large and small graphs grows. To solve this problem, they forced the GL-GNN to pay more attention to the substructures that are hidden in large unlabeled graphs but rarely appear in small labeled graphs. SizeShiftReg [103] constrains GNNs to be robust to size-shift through a regularization approach. SizeShiftReg coarsens the input graph and minimizes the discrepancy between the distribution of the original and coarsened graph embeddings.

D.3.2 Few-shot Learning. In considering few-shot learning scenarios, Ma *et al.* [104] found that there are also differences in the substructures between a few labeled graphs and a large number of unlabeled graphs. This is because a statistical sample of the training data is too small to represent the substructural distributions of the whole dataset. Thus, they paid more attention to capturing substructures in unseen unlabeled graphs. Chauhan *et al.* [105] clusters graphs based on their

spectral properties, to produce super-class graphs. Graph-level representations can be learned from super-class graphs as they have excellent generalization.

D.4 The explainability of GL-GNNs

The black-box nature of deep neural networks limits the applicability of GL-GNNs to situations where trust is not an absolutely crucial requirement. Making GL-GNNs explain their predictions in a way that is more interpretable to humans is therefore of great significance to extending the research of GL-GNNs. Studies on GL-GNNs need to shed insights into how they handle node features and topologies when it comes to predictions. They also need to more clearly demonstrate how the models identify significant subgraphs and features. Methods to explain GL-GNNs can be roughly divided into two categories. One group involves methods that explain the prediction of each input graph; the other group of methods captures common patterns in the predictions of a set of graphs as explanations.

D.4.1 A Single Graph. There are three ways to understand GL-GNNs predictions based on a single graph: they can be perturbation-based, model-proxy-based, or gradient-based. Perturbation-based methods mask nodes, edges, or substructures in the input graph to generate new predictions. These are then compared to the original input prediction to highlight the important features or structures influencing the GL-GNNs.

For example, GNNExplainer [106] masks nodes and edges by changing the feature and adjacency matrices, to form masked graphs. An input graph and its masked graphs are predicted by a trained GL-GNN, while GNNExplainer aims to find the masked graphs with maximized mutual information between its' prediction and the input's prediction. This found masked graph is the one that preserves the most significant substructures to the GL-GNN's given prediction. Alternatively, SubgraphX [107] samples a group of nodes' neighborhoods as subgraphs. A trained GL-GNN is then used to compute Shapley values [108] for all the sampled subgraphs. These values represent each subgraph's contribution to the GL-GNN's prediction. PGExplainer [109] trains an MLP to determine which edges are valuable to a GNN's prediction and then removes any irrelevant edges to form a new graph. Subsequently, the original and the newly-formed graph are fed into a trained GL-GNN so as to optimize an MLP by maximizing the mutual information between their predictions.

Model-proxy-based methods utilize a simpler surrogate model to approximate the predictions of GL-GNNs. PGM-Explainer [110] adopts an explainable Bayesian network [111] to calculate the relationship dependencies between nodes, so as to generate a probability graph that describes the input graph.

Gradient-based approaches measure the importance of different input features by back-propagating the gradients of the neural networks. Gradient-weighted Class Activation Mapping (Grad-CAM) [112], for example, takes the gradient value of each node embedding in a graph classification task as a measurement of the nodes' significance to the GL-GNN's prediction. Grad-CAM then measures this subgraph's importance to the prediction by averaging the gradient values of all node embeddings from the subgraph.

D.4.2 A Set of Graphs. What is common to all the above methods is that they can only learn independent explanations for each instance of a graph [106, 113]. However, often the predictions of GL-GNNs made by GL-GNNs are based on a set of graphs. Thus, understanding the rules or graph patterns that a GL-GNN mines from a set of graphs can provide high-level and generic insights into the explainability of GL-GNNs. XGNN [114] employs a reinforcement learning guided graph generator that generates a graph pattern for different graphs in the same class. The graph generator is trained via policy gradient to maximize the certain label prediction [115]. Recently, Azzolin *et al.* [116] set a kernel function between extracted subgraphs and trainable prototypes and feed the

Table 5. Summary of Graph Pooling.

Model	Year	Method	Venue	Language	Code Repository
Global-Numeric	2020	PNA[117]	NeurIPS	Python-Pytorch	https://github.com/lukevabarret/pna
	2020	TextING[118]	ACL	Python-Tensorflow	https://github.com/CRIPAC-DIG/TextING
	2020	SOPOOL[119]	TPAMI	Python-Pytorch	https://github.com/divelab/sopool
Global-Attention	2016	Set2Set[120]	ICLR	Python-Pytorch	https://github.com/pyg-team/pytorch_geometric
Global-CNN	2016	PATCHYSAN[41]	ICML	Python	https://github.com/tvayer/PSCN
	2018	KCNN[44]	ICANN	Python-Pytorch	https://github.com/giannisnik/cnn-graph-classification
Global-Top K	2018	SortPool[121]	AAAI	Python-Pytorch	https://github.com/muhanzhang/pytorch_DGCNN
	2020	GSAPOOL[122]	WWW	Python-Pytorch	https://github.com/psp3dgc/gsapool
Hierarchical-Clustering	2014	DLCN[123]	ICLR	-	-
	2015	GraphCNN[124]	Arxiv	Python-Tensorflow	https://github.com/mdeff/cnn_graph
	2018	DiffPool[125]	NeurIPS	Python-Pytorch	https://github.com/RexYing/diffpool
	2019	EigenPool[126]	KDD	Python-Pytorch	https://github.com/alge24/eigenpooling
	2020	StructPool[127]	ICLR	Python-Pytorch	https://github.com/Nate1874/StructPool
	2020	MinCutPool[128]	ICML	Python-Tensorflow	https://github.com/FilippoMB/Spectral-Clustering-with-Graph-Neural-Networks-for-Graph-Pooling
	2021	GMT[129]	ICLR	Python-Pytorch	https://github.com/JinheonBaek/GMT
Hierarchical-Top K	2018	SHGC[130]	Arxiv	Python-Tensorflow	https://github.com/HeapHop30/hierarchical-pooling
	2019	U-Nets[131]	ICML	Python-Pytorch	https://github.com/HongyangGao/Graph-U-Nets
	2019	SAGPool[132]	ICML	Python-Pytorch	https://github.com/inyeoplee77/SAGPool
	2020	ASAP[133]	AAAI	Python-Pytorch	https://github.com/mallabaisc/ASAP
Hierarchical-Tree	2017	MoNet[134]	CVPR	Python-Pytorch	https://github.com/dmlc/dgl/tree/master/examples/mxnet/monet
	2019	EdgePool[135]	Arxiv	Python-Pytorch	https://github.com/pyg-team/pytorch_geometric/blob/master/torch_geometric/nn/pool/edge_pool.py
	2022	HRN[136]	IJCAL	Python	https://github.com/Wu-Junran/HierarchicalReporting
	2022	SEP-G[137]	ICML	Python	https://github.com/Wu-Junran/SEP

kernel value into an MLP for prediction. Finally, the trainable prototypes are assumed as the global explanation of a set of graphs.

E GRAPH POOLING

Table 5 summarizes the graph pooling approaches introduced in this section. Moreover, we introduce some recent investigations about the efficacy of this newly-emerging technique (see Section E.1).

E.1 The effectivity of Graph Pooling

As a downstream summarization component of GNNs, graph pooling has attracted a surge of research interest. However, since graph pooling is so new, much work is required to investigate the effectiveness of all the various graph pooling algorithms. Mesquita *et al.* [138] conducted controlled experiments to empirically evaluate the effectiveness of clustering-based hierarchical graph pooling. First, they adopted two opposite strategies for guiding some clustering-based hierarchical graph pooling processes —specifically, clustering each of non-adjacent and adjacent nodes. The final results not only show that the two strategies are comparable, they also indicate that off-the-shelf clustering algorithms, which tend to cluster adjacent nodes, fail to improve graph pooling. As part of the experiments, Mesquita and colleagues also replaced the learnable assignment matrix in DiffPool [125] with an immutable probability assignment matrix: uniform, normal and Bernoulli distributions were selected. The experimental results verify that the performance of fixed-probability-assignment-matrix-guided graph pooling is not weaker than that of DiffPool. Overall, they concluded that the current clustering-based hierarchical pooling may not be particularly effective and matched this will a call for more sanity checks and ablation studies of the current graph pooling algorithms to fairly evaluate their contributions. Another study on Pooled Architecture Search (PAS) [139] was dedicated to investigating the effectiveness of graph pooling —this time with different datasets. The results of the study show that the effectiveness of graph pooling algorithms is data-specific, that

is to say, different input data needs to be handled by a suitable graph pooling algorithm. For this reason, PAS includes a differentiable search method to select the most appropriate graph pooling algorithm for the given input data.

F DOWNSTREAM TASKS AND APPLICATIONS

Fig. 10 summarizes some common downstream tasks and applications in graph-level learning.

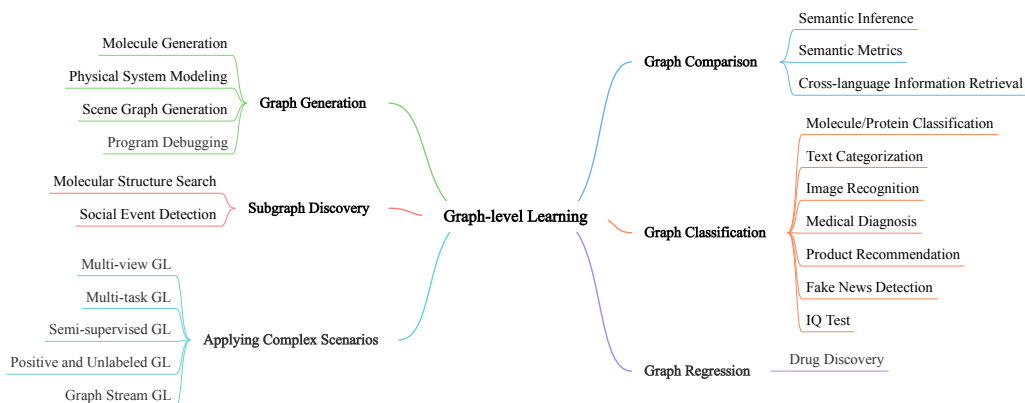


Fig. 10. An overview of graph-level learning downstream tasks and their practical applications.

G FUTURE DIRECTIONS

G.1 Graph-level Fairness Learning

The bias in data can easily lead to issues with fairness, where machine learning models make discriminatory predictions towards certain demographic groups based on sensitive attributes such as race. One feasible solution to debiasing the data is to conduct a competitive game between a biased and a debiased encoder. The game is won when the fairness-aware debiased is able to cheat its competitor [140, 141]. Other algorithms add constraints to the loss function to counterbalance model performance with fairness [142, 143].

Opportunities: Most work on improving the fairness of models have involved node-level tasks and single graphs [144]. However, injecting an awareness of fairness into graph-level learning algorithms is also critical work. Some graph-level learning tasks, such as disease prediction and fraud detection, demand fair results if they are to accurately guide people's decision-making. One challenge to be overcome in attempting to make graph-level learning fair is that the representative GL-GNNs, i.e., MPNNs, will tend to produce unfair predictions in the face of data bias because the message passing mechanisms actually spread the bias via neighborhood structures [145]. We refer readers who are interested in this topic to [146], which gives an exhaustive introduction to fairness learning with graph-structured data.

REFERENCES

- [1] S. Hido and H. Kashima, "A linear-time graph kernel," in *Proc. ICDM*, pp. 179–188, 2009.
- [2] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels.," *J. Mach. Learn. Res.*, vol. 12, no. 9, 2011.

- [3] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: efficient graph kernels from propagated information," *Mach. Learn.*, vol. 102, no. 2, pp. 209–245, 2016.
- [4] C. Morris, K. Kersting, and P. Mutzel, "Globalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs," in *Proc. ICDM*, pp. 327–336, 2017.
- [5] B. Rieck, C. Bock, and K. Borgwardt, "A persistent weisfeiler-lehman procedure for graph classification," in *Proc. ICML*, pp. 5448–5458, 2019.
- [6] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Proc. ICDM*, pp. 1–8, 2005.
- [7] G. Nikolentzos, P. Meladianos, F. Rousseau, Y. Stavarakas, and M. Vazirgiannis, "Shortest-path graph kernels for document similarity," in *Proc. EMNLP*, pp. 1890–1900, 2017.
- [8] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Proc. LNAI*, pp. 129–143, 2003.
- [9] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, "Extensions of marginalized graph kernels," in *Proc. ICML*, pp. 70–78, 2004.
- [10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, 2010.
- [11] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *Proc. ICML*, pp. 225–232, 2005.
- [12] D. Pachauri, R. Kondor, and V. Singh, "Solving the multi-way matching problem by permutation synchronization," in *Proc. NeurIPS*, pp. 1860–1868, 2013.
- [13] F. D. Johansson and D. Dubhashi, "Learning with similarity functions on graphs using matchings of geometric embeddings," in *Proc. KDD*, pp. 467–476, 2015.
- [14] M. Schiavinato, A. Gasparetto, and A. Torsello, "Transitive assignment kernels for structural classification," in *Proc. SIMBAD*, pp. 146–159, 2015.
- [15] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Proc. AISTATS*, pp. 488–495, 2009.
- [16] F. Costa and K. De Grave, "Fast neighborhood subgraph pairwise distance kernel," in *Proc. ICML*, pp. 255–262, 2010.
- [17] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *Proc. ICML*, p. 291–298, 2012.
- [18] A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in *Proc. ECML-PKDD*, pp. 13–23, 2000.
- [19] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proc. ICDM*, pp. 313–320, 2001.
- [20] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proc. ICDM*, pp. 721–724, 2002.
- [21] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining significant graph patterns by leap search," in *Proc. SIGMOD*, pp. 433–444, 2008.
- [22] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *Proc. SDM*, pp. 1076–1087, 2009.
- [23] X. Kong and S. Y. Philip, "Multi-label feature selection for graph classification," in *Proc. ICDM*, pp. 274–283, 2010.
- [24] X. Kong and P. S. Yu, "Semi-supervised feature selection for graph classification," in *Proc. KDD*, pp. 793–802, 2010.
- [25] Y. Zhao, X. Kong, and S. Y. Philip, "Positive and unlabeled learning for graph classification," in *Proc. ICDM*, pp. 962–971, 2011.
- [26] J. Wu, Z. Hong, S. Pan, X. Zhu, Z. Cai, and C. Zhang, "Multi-graph-view learning for graph classification," in *Proc. ICDM*, pp. 590–599, 2014.
- [27] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Proc. NeurIPS*, pp. 87–97, 2017.
- [28] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," in *Proc. ICML*, pp. 2186–2195, 2018.
- [29] C. Cai and Y. Wang, "A simple yet effective baseline for non-attribute graph classification," *arXiv preprint arXiv:1811.03508*, 2018.
- [30] A. Tsitsulin, M. Munkhoeva, and B. Perozzi, "Just slaq when you approximate: Accurate spectral distances for web-scale graphs," in *Proc. WWW*, pp. 2697–2703, 2020.
- [31] X. Liu, L. Fu, and X. Wang, "Bridging the gap between von neumann graph entropy and structural information: theory and applications," in *Proc. WWW*, pp. 3699–3710, 2021.
- [32] S. Sawlani, L. Zhao, and L. Akoglu, "Fast attributed graph embedding via density of states," in *Proc. ICDM*, pp. 559–568, 2021.
- [33] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," in *Proc. KDD Workshop on Mining and Learning with Graphs*, 2016.
- [34] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," in *Proc. KDD Workshop on Mining and Learning with Graphs*, 2017.

- [35] D. Nguyen, W. Luo, T. D. Nguyen, S. Venkatesh, and D. Phung, "Learning graph representation via frequent subgraphs," in *Proc. SDM*, pp. 306–314, 2018.
- [36] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proc. ICLR*, pp. 1–20, 2016.
- [37] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. KDD*, pp. 1666–1674, 2018.
- [38] X. Zhao, B. Zong, Z. Guan, K. Zhang, and W. Zhao, "Substructure assembling network for graph classification," in *Proc. AAAI*, vol. 32, 2018.
- [39] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *Proc. ICML*, pp. 610–619, 2018.
- [40] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *Proc. ICML*, pp. 5708–5717, 2018.
- [41] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. ICML*, pp. 2014–2023, 2016.
- [42] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. NeurIPS*, pp. 1993–2001, 2016.
- [43] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, pp. 3693–3702, 2017.
- [44] G. Nikolentzos, P. Meladianos, A. J.-P. Tixier, K. Skianis, and M. Vazirgiannis, "Kernel graph convolutional neural networks," in *Proc. ICANN*, pp. 22–32, 2018.
- [45] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," in *Proc. ICML-IJCAL workshop on Computational Biology*, pp. 1–12, 2018.
- [46] Z. Xinyi and L. Chen, "Capsule graph neural network," in *Proc. ICLR*, pp. 1–16, 2018.
- [47] M. D. G. Mallea, P. Meltzer, and P. J. Bentley, "Capsule neural networks for graph classification using explicit tensorial graph representations," *arXiv preprint arXiv:1902.08399*, 2019.
- [48] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. NeurIPS*, pp. 2224–2232, 2015.
- [49] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Proc. NeurIPS*, pp. 4502–4510, 2016.
- [50] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. ICML*, pp. 1263–1272, 2017.
- [51] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nat. Commun.*, vol. 8, no. 1, pp. 1–8, 2017.
- [52] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. ICLR*, pp. 1–17, 2018.
- [53] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proc. AAAI*, pp. 4602–4609, 2019.
- [54] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Proc. NeurIPS*, pp. 1–12, 2019.
- [55] R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Relational pooling for graph representations," in *Proc. ICML*, pp. 4663–4673, 2019.
- [56] W. Azizian and marc lelarge, "Expressive power of invariant and equivariant graph neural networks," in *Proc. ICLR*, pp. 1–39, 2021.
- [57] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lio, and M. Bronstein, "Weisfeiler and lehman go topological: Message passing simplicial networks," in *Proc. ICML*, pp. 1026–1037, 2021.
- [58] C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Lio, G. F. Montufar, and M. Bronstein, "Weisfeiler and lehman go cellular: Cw networks," in *Proc. NeurIPS*, pp. 2625–2640, 2021.
- [59] R. Abboud, I. I. Ceylan, M. Grohe, and T. Lukasiewicz, "The surprising power of graph neural networks with random node initialization," in *Proc. IJCAI*, pp. 2112–2118, 2021.
- [60] E. Alsentzer, S. G. Finlayson, M. M. Li, and M. Zitnik, "Subgraph neural networks," in *Proc. NeurIPS*, pp. 1–13, 2020.
- [61] Q. Sun, J. Li, H. Peng, J. Wu, Y. Ning, P. S. Yu, and L. He, "Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism," in *Proc. WWW*, pp. 2081–2091, 2021.
- [62] M. Zhang and P. Li, "Nested graph neural networks," in *Proc. NeurIPS*, pp. 1–14, 2021.
- [63] L. Zhao, W. Jin, L. Akoglu, and N. Shah, "From stars to subgraphs: Uplifting any GNN with local structure awareness," in *Proc. ICLR*, pp. 1–22, 2022.
- [64] A. Wijesinghe and Q. Wang, "A new perspective on" how graph neural networks go beyond weisfeiler-lehman?," in *Proc. ICLR*, pp. 1–23, 2022.
- [65] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *Proc. ICLR*, pp. 1–46, 2022.

- [66] G. Bouritsas, F. Frasca, S. P. Zafeiriou, and M. Bronstein, “Improving graph neural network expressivity via subgraph isomorphism counting,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–12, 2022.
- [67] S. S. Du, K. Hou, R. Salakhutdinov, B. Póczos, R. Wang, and K. Xu, “Graph neural tangent kernel: Fusing graph neural networks with graph kernels,” in *Proc. NeurIPS*, pp. 5724–5734, 2019.
- [68] R. Al-Rfou, B. Perozzi, and D. Zelle, “Ddgg: Learning graph representations for deep divergence graph kernels,” in *Proc. WWW*, pp. 37–48, 2019.
- [69] D. Chen, L. Jacob, and J. Mairal, “Convolutional kernel networks for graph-structured data,” in *Proc. ICML*, pp. 1576–1586, 2020.
- [70] G. Nikolentzos and M. Vazirgiannis, “Random walk graph neural networks,” in *Proc. NeurIPS*, pp. 16211–16222, 2020.
- [71] Q. Long, Y. Jin, Y. Wu, and G. Song, “Theoretically improving graph neural networks via anonymous walk graph kernels,” in *Proc. WWW*, pp. 1204–1214, 2021.
- [72] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, “Graph contrastive learning with augmentations,” in *Proc. NeurIPS*, pp. 5812–5823, 2020.
- [73] F.-Y. Sun, J. Hoffman, V. Verma, and J. Tang, “Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization,” in *Proc. ICLR*, pp. 1–16, 2020.
- [74] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, “Gcc: Graph contrastive coding for graph neural network pre-training,” in *Proc. KDD*, pp. 1150–1160, 2020.
- [75] K. Hassani and A. H. Khasahmadi, “Contrastive multi-view representation learning on graphs,” in *Proc. ICML*, pp. 4116–4126, 2020.
- [76] Y. You, T. Chen, Y. Shen, and Z. Wang, “Graph contrastive learning automated,” in *Proc. ICML*, pp. 12121–12132, 2021.
- [77] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Proc. NeurIPS*, pp. 3837–3845, 2016.
- [78] R. Levie, W. Huang, L. Bucci, M. M. Bronstein, and G. Kutyniok, “Transferability of spectral graph convolutional neural networks,” *J. Mach. Learn. Res.*, vol. 22, pp. 272–331, 2021.
- [79] M. Balcilar, P. Héroux, B. Gauzere, P. Vasseur, S. Adam, and P. Honeine, “Breaking the limits of message passing graph neural networks,” in *Proc. ICML*, pp. 599–608, 2021.
- [80] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, “Graph neural networks with convolutional arma filters,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 3496–3507, 2021.
- [81] X. Zheng, B. Zhou, J. Gao, Y. Wang, P. Lió, M. Li, and G. Montufar, “How framelets enhance graph neural networks,” in *Proc. ICML*, pp. 1–10, 2021.
- [82] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, “Learning deep representations by mutual information estimation and maximization,” in *Proc. ICLR*, pp. 1–24, 2019.
- [83] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *arXiv preprint arXiv:1807.03748*, 2018.
- [84] Y. Wang, J. Wang, Z. Cao, and A. Barati Farimani, “Molecular contrastive learning of representations via graph neural networks,” *Nat. Mach. Intell.*, vol. 4, no. 3, pp. 279–287, 2022.
- [85] B. Weisfeiler and A. Leman, “The reduction of a graph to canonical form and the algebra which appears therein,” *NTI Series*, vol. 2, no. 9, pp. 12–16, 1968.
- [86] V. Arvind, F. Fuhlbrück, J. Köbler, and O. Verbitsky, “On weisfeiler-leman invariance: Subgraph counts and related graph properties,” *J. Comput. Syst. Sci.*, vol. 113, pp. 42–59, 2020.
- [87] C. Vignac, A. Loukas, and P. Frossard, “Building powerful and equivariant graph neural networks with structural message-passing,” in *Proc. NeurIPS*, pp. 14143–14155, 2020.
- [88] Z. Chen, L. Chen, S. Villar, and J. Bruna, “Can graph neural networks count substructures?,” in *Proc. NeurIPS*, pp. 10383–10395, 2020.
- [89] J.-Y. Cai, M. Fürer, and N. Immerman, “An optimal lower bound on the number of variables for graph identification,” *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.
- [90] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, “Benchmarking graph neural networks,” *arXiv preprint arXiv:2003.00982*, 2020.
- [91] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., “Relational inductive biases, deep learning, and graph networks,” *arXiv preprint arXiv:1806.01261*, 2018.
- [92] U. Alon and E. Yahav, “On the bottleneck of graph neural networks and its practical implications,” in *Proc. ICLR*, pp. 1–16, 2021.
- [93] R. Brijder, F. Geerts, J. V. D. Bussche, and T. Weerwag, “On the expressive power of query languages for matrices,” *ACM Trans. Database Syst.*, vol. 44, no. 4, pp. 1–31, 2019.
- [94] F. Geerts, “On the expressive power of linear algebra on graphs,” *Theory Comput. Syst.*, vol. 65, no. 1, pp. 179–239, 2021.

- [95] F. Geerts and J. L. Reutter, “Expressiveness and approximation properties of graph neural networks,” in *Proc. ICLR*, pp. 1–43, 2022.
- [96] G. Dasoulas, L. D. Santos, K. Scaman, and A. Virmaux, “Coloring graph neural networks for node disambiguation,” in *Proc. IJCAI*, pp. 1–17, 2021.
- [97] R. Sato, “A survey on the expressive power of graph neural networks,” *arXiv preprint arXiv:2003.04078*, 2020.
- [98] C. Bodnar, F. Di Giovanni, B. Chamberlain, P. Lio, and M. Bronstein, “Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in GNNs,” *Proc. NeurIPS*, vol. 35, pp. 18527–18541, 2022.
- [99] F. Barbero, C. Bodnar, H. S. d. O. Borde, M. Bronstein, P. Veličković, and P. Liò, “Sheaf neural networks with connection laplacians,” in *Proc. ICML Workshop on Topology, Algebra, and Geometry in Machine Learning*, pp. 1–8, 2022.
- [100] K. Sinha, S. Sodhani, J. Pineau, and W. L. Hamilton, “Evaluating logical generalization in graph neural networks,” *arXiv preprint arXiv:2003.06560*, 2020.
- [101] K. Xu, M. Zhang, J. Li, S. S. Du, K.-I. Kawarabayashi, and S. Jegelka, “How neural networks extrapolate: From feedforward to graph neural networks,” in *Proc. ICLR*, pp. 1–52, 2021.
- [102] G. Yehudai, E. Fetaya, E. Meiroim, G. Chechik, and H. Maron, “From local structures to size generalization in graph neural networks,” in *Proc. ICML*, pp. 11975–11986, 2021.
- [103] D. Buffelli, P. Liò, and F. Vandin, “Sizeshiftreg: a regularization method for improving size-generalization in graph neural networks,” in *Proc. NeurIPS*, pp. 1–12, 2022.
- [104] N. Ma, J. Bu, J. Yang, Z. Zhang, C. Yao, Z. Yu, S. Zhou, and X. Yan, “Adaptive-step graph meta-learner for few-shot graph classification,” in *Proc. CIKM*, pp. 1055–1064, 2020.
- [105] J. Chauhan, D. Nathani, and M. Kaul, “Few-shot learning on graphs via super-classes based on graph spectral measures,” in *Proc. ICLR*, pp. 1–19, 2020.
- [106] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, “Gnnexplainer: Generating explanations for graph neural networks,” in *Proc. NeurIPS*, pp. 1–12, 2019.
- [107] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, “On explainability of graph neural networks via subgraph explorations,” in *Proc. ICML*, pp. 12241–12252, 2021.
- [108] H. W. Kuhn and A. W. Tucker, *Contributions to the Theory of Games*. No. 28, Princeton University Press, 1953.
- [109] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, “Parameterized explainer for graph neural network,” in *Proc. NeurIPS*, pp. 19620–19631, 2020.
- [110] M. Vu and M. T. Thai, “Pgm-explainer: Probabilistic graphical model explanations for graph neural networks,” in *Proc. NeurIPS*, pp. 12225–12235, 2020.
- [111] J. Pearl, “Markov and bayesian networks: Two graphical representations of probabilistic knowledge,” *Probabilistic Reasoning in Intelligent Systems*, pp. 77–141, 1988.
- [112] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, “Explainability methods for graph convolutional neural networks,” in *Proc. CVPR*, pp. 10772–10781, 2019.
- [113] H. Yuan, H. Yu, S. Gui, and S. Ji, “Explainability in graph neural networks: A taxonomic survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 5782–5799, 2022.
- [114] H. Yuan, J. Tang, X. Hu, and S. Ji, “Xggn: Towards model-level explanations of graph neural networks,” in *Proc. KDD*, pp. 430–438, 2020.
- [115] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proc. NeurIPS*, pp. 1–7, 1999.
- [116] S. Azzolin, A. Longa, P. Barbiero, P. Lio, and A. Passerini, “Global explainability of GNNs via logic combination of learned concepts,” in *Proc. ICLR*, pp. 1–16, 2023.
- [117] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, “Principal neighbourhood aggregation for graph nets,” in *Proc. NeurIPS*, pp. 1–11, 2020.
- [118] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, “Every document owns its structure: Inductive text classification via graph neural networks,” in *Proc. ACL*, pp. 334–339, 2020.
- [119] Z. Wang and S. Ji, “Second-order pooling for graph neural networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–12, 2020.
- [120] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” in *Proc. ICLR*, pp. 1–11, 2016.
- [121] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in *Proc. AAAI*, pp. 1–8, 2018.
- [122] L. Zhang, X. Wang, H. Li, G. Zhu, P. Shen, P. Li, X. Lu, S. A. A. Shah, and M. Bennamoun, “Structure-feature based graph self-adaptive pooling,” in *Proc. WWW*, pp. 3098–3104, 2020.
- [123] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and deep locally connected networks on graphs,” in *Proc. ICLR*, pp. 1–14, 2014.
- [124] M. Henaff, J. Bruna, and Y. LeCun, “Deep convolutional networks on graph-structured data,” *arXiv preprint arXiv:1506.05163*, 2015.

- [125] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. NeurIPS*, pp. 4805–4815, 2018.
- [126] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proc. KDD*, pp. 723–731, 2019.
- [127] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *Proc. ICLR*, pp. 1–12, 2020.
- [128] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *Proc. ICML*, pp. 874–883, 2020.
- [129] J. Baek, M. Kang, and S. J. Hwang, "Accurate learning of graph representations with graph multiset pooling," in *Proc. ICLR*, pp. 1–22, 2021.
- [130] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," *arXiv preprint arXiv:1811.01287*, 2018.
- [131] H. Gao and S. Ji, "Graph u-nets," in *Proc. ICML*, pp. 2083–2092, 2019.
- [132] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. ICML*, pp. 3734–3743, 2019.
- [133] E. Ranjan, S. Sanyal, and P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proc. AAAI*, pp. 5470–5477, 2020.
- [134] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. CVPR*, pp. 5115–5124, 2017.
- [135] F. Diehl, "Edge contraction pooling for graph neural networks," *arXiv preprint arXiv:1905.10990*, 2019.
- [136] J. Wu, S. Li, J. Li, Y. Pan, and K. Xu, "A simple yet effective method for graph classification," in *Proc. IJCAI*, pp. 1–7, 2022.
- [137] J. Wu, X. Chen, K. Xu, and S. Li, "Structural entropy guided graph hierarchical pooling," in *Proc. ICML*, pp. 24017–24030, 2022.
- [138] D. P. P. Mesquita, A. H. S. Jr., and S. Kaski, "Rethinking pooling in graph neural networks," in *Proc. NeurIPS*, pp. 2220–2231, 2020.
- [139] L. Wei, H. Zhao, Q. Yao, and Z. He, "Pooling architecture search for graph classification," in *Proc. CIKM*, pp. 2091–2100, 2021.
- [140] A. Bose and W. Hamilton, "Compositional fairness constraints for graph embeddings," in *Proc. ICML*, pp. 715–724, PMLR, 2019.
- [141] F. Masrour, T. Wilson, H. Yan, P.-N. Tan, and A. Esfahanian, "Bursting the filter bubble: Fairness-aware network link prediction," in *Proc. AAAI*, pp. 841–848, 2020.
- [142] J. Kang, J. He, R. Maciejewski, and H. Tong, "Inform: Individual fairness on graph mining," in *Proc. KDD*, pp. 379–389, 2020.
- [143] P. Li, Y. Wang, H. Zhao, P. Hong, and H. Liu, "On dyadic fairness: Exploring and mitigating bias in graph connections," in *Proc. ICLR*, pp. 1–18, 2020.
- [144] Y. Dong, N. Liu, B. Jalaian, and J. Li, "Edits: Modeling and mitigating data bias for graph neural networks," in *Proc. WWW*, pp. 1259–1269, 2022.
- [145] E. Dai and S. Wang, "Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information," in *Proc. WSDM*, pp. 680–688, 2021.
- [146] E. Dai, T. Zhao, H. Zhu, J. Xu, Z. Guo, H. Liu, J. Tang, and S. Wang, "A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability," *arXiv preprint arXiv:2204.08570*, 2022.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009